SSD $\neq$ SSD – An Empirical Study to Identify Common Properties and Type-specific Behavior

Volker Hudlet, Daniel Schall
Databases and Information Systems Group
Department of Computer Science
University of Kaiserslautern
D-67653 Kaiserslautern, Germany
{hudlet, schall}@cs.uni-kl.de

Abstract: Solid-state disks are promising high access speed at low energy consumption. While the basic technology for SSDs – flash memory – is well established, new product models are constantly emerging. With each new SSD generation, their behavior pattern changes significantly and it is therefore difficult to make out characteristics for SSDs in general. In this paper, we accomplish empirical, database-centric performance measurements for SSDs, explain the results, and try to derive common characteristics. By comparing our measurement results, we detect no ground truth valid for all solid-state disks. Furthermore, we show that a number of prevalent assumptions about SSDs, which several SSD-specific DBMS optimizations are based on, are questionable by now. As a consequence of these findings, tailor-made DBMS algorithms for specific SSD types may be unsuitable and optimal use of SSD technology in an DBMS context may require careful design and rather adaptive algorithms.

1 Introduction

Solid-state disks gained a lot of attention lately. While many research papers present new algorithms to exploit performance of SSDs, enterprises are thinking of ways to incorporate them in their own products. Hence, flash chips and solid-state disks have become established products by now. SSDs offer dramatically improved random access behavior compared to conventional spinning disks. Due to their lack of mechanical parts, they are able to answer requests much faster than disks. That makes them perfect candidates for incorporating into database systems to speed up data processing. Due to the absence of spinning platters, SSDs also promise to come with a smaller energy footprint. Still, SSDs are not the swiss army knife of storage devices, they come with some limitations we have to deal with, for example read/write asymmetry. The market for solid-state disks is constantly changing and newer SSD generations are steadily improving their performance. With every new SSD generation, new product characteristics are emerging. Some drawbacks of earlier SSDs have been resolved in recent models. The constant change in the devices' properties makes it difficult for upper-layer algorithms to exploit the underlying storage. Optimizations tailored to a dedicated SSD model can have even negative effects on differently behaving models.

In this paper, we measure performance and energy consumption of five solid-state disks and compare them to manufacturer-provided data sheets. By pinpointing the specific characteristics of each drive, we claim that it is unwise for research to reveal the DB performance benefits of this disruptive technology change by relying on a single SSD type. We also derive common characteristics of flash drives in order to support research of the database community and leverage the exploration of flash-specific algorithms. This paper is structured as follows: In Section 2, we briefly summarize the internal structures of SSDs. Next, we outline important related work in Section 3. In Section 4, we are presenting our measurement methodology and the solid-state disks used for our performance study. Section 5 shows our experimental results. We interpret our results and compare them to the properties and facts listed in the data sheets of the manufacturers. In Section 6 we derive common characteristics and review some assumptions made about SSDs in research papers. In Section 7, we reflect our measurement setup and point out some possible limitations to our approach. Finally, we draw our conclusion in Section 8.

2 General SSD Characteristics

Solid-state disks are using flash chips for persistently storing data. An abstraction layer (*FTL*) on top of the chips provides a block device interface and hides the specific flash chip characteristics.

Flash Chips Flash chips are used to store persistent data on SSDs in a matrix of storage cells. The cells can either embody NOR or NAND gates. Modern cells, called *Multi-Level Cells* (MLC), can store more than one bit per cell. Todays solid-state disks are mostly composed of MLC NAND chips; we focus our description on that technology.

Reading from flash chips can only be done pagewise, where each page contains about 1 – 4 KB. It takes about 50 μs per page. Writing pages requires higher voltages, it takes about 10 times as long as reading ($\sim$500 μs). Furthermore, each cell has to be erased prior to writing new values to it. Erasing is even more cost-intensive and can only be done in larger blocks, not by one page at a time. Typically, a block is 32 – 256 KB in size. It takes about 20 times as long to erase a block as reading a page ($\sim$1000 μs). Erasing flash blocks leads to a slow but constant destruction of the cells. After about 10^5 erase cycles, cells will start to wear out and the block is no longer able to retain data.

Flash chips can be grouped together in so-called *planes* to increase storage capacity. Multiple planes can be accessed in parallel to enhance data throughput.

Flash Translation Layer To cope with the limitations of bare-metal flash chips, a mitigation layer is used on top of the chips/planes, called *Flash Translation Layer* (FTL). It provides a block-device interface to the upper layers, making the SSD look like a common storage disk. Therefore, the SSD user does not have to worry about erasure-before-write and handling worn-out blocks; these jobs are handled by the FTL. Because overwriting data on flash chips requires special treatment, the FTL must provide an erase-before-write mechanism that is able to save neighboring flash pages from being erased. Further, the FTL has to take care of worn-out cells.

Based on these basic functionalities, todays FTLs provide a lot more logic to further improve SSD performance. First of all, to avoid the need for erasing hot-spot areas over and over again, a page mapping is introduced to redirect logical page accesses to different physical locations on every new write. This helps to save erase cycles and, therefore, also improves performance. To free up unused areas of the SSD, a garbage collection schema can be implemented allowing the SSD to asynchronously perform erase and cleanup operations when the device is idle. Further reorganization tasks such as summarizing sparsely filled blocks can be employed. Like conventional hard disks, SSDs usually have an internal DRAM cache to buffer write requests or store prefetched pages. This buffer enables solid-state disks to backup and restore pages during erase cycles and to keep in-memory information, e.g., page-mapping structures. By using an FTL, it is possible to avoid most drawbacks of flash chips while making use of the advantages. Therefore, the FTL is a major performance-critical part of every SSD and manufacturers are eager to keep the implementation details a secret. A more detailed explanation of flash memory, SSDs, and their internal structures can be found in [RKM09, CPP$^+$06].

3 Related Work

SSDs have been intensively explored in recent years with a focus on the characteristics of SSDs, on its integration into (existing) hardware systems and on its effective exploitation.

Operating Systems At the device level, Wang et al. [WGK09] propose non-in-place updates when writing to SSDs, which results in a performance improvement in their work, but which is automatically performed inside modern SSDs to mitigate wear-out. The same holds for flash-optimized file systems (e.g., *YAFFS* [Man02]) which employ journaled writes to avoid random and in-place updates. FTL implementation proposals [CPP$^+$06, LPC$^+$07, KKN$^+$02] show different techniques that yield a logical-to-physical block mapping. As the FTL is embedded inside the device firmware, it is unknown whether any of theses techniques or which particular technique has been adopted by the SSD manufacturers.

System Architecture Approaches concerning the system architecture try to find suitable solutions where to place SSDs in computer systems and how to incorporate them. Three main strategies [RKM09] are feasible using SSDs: as extended system memory, as storage accelerator, or as alternative storage device. Other works use a hybrid approach of SSDs and conventional HDDs where the SSD serves as persistent buffer for HDDs in order to mitigate I/O latency [CMB$^+$10, KJKM09] or – depending on their workload – to adaptively place pages on one of these device types [KV08].

DBMS-specific Optimizations The exploitation of SSDs to increase the performance of data-intensive workloads still is in focus of the database systems community. An overview of the knobs and layers which can be made SSD-aware in a database system is given by Graefe [Gra09]. Of course, several components which interact and rely on external storage have been incorporated to be aligned to the characteristics of flash memory, so a lot of different proposals have been made in recent years. This includes amongst others SSD-tailored DB buffer replacement algorithms [OHJ09], page layouts [THS$^+$09], and index structures [AGS$^+$09, KJKK07, WKC07]. Do et al. [DP09] show the impact of SSDs on join processing, mainly the tendency to become CPU-bound rather than being I/O-bound when HDDs are used. Tsirogiannis et al. [THS$^+$09] recommend late materialization for speeding up join processing on SSDs.

SSD Measurements Papers in this area try to find out more about the SSD behavior and how to react to certain situations in reality. Most of them are based on micro-benchmarks used to reveal internal characteristics. Bouganim et al. [BJB09] and Chen et al. [CKZ09] were the first who tried to derive the intrinsic characteristics of different SSDs. They conclude that SSDs have to be considered as black boxes, as they follow no common rule. SSDs in RAID configurations have been examined in [BdNSS10] and [PABG10], where the latter one states that some effects of SSDs, e.g., the read-write asymmetry, are amplified due to the RAID mechanism. Overall, some characteristics are differently interpreted. [BJB09] state that they did not observe any performance improvements from submitting I/Os in parallel, [BdNSS10] use long queue depths and asynchronous I/O in order to increase the bandwidth.

As SSD advance, aspects covered by some approaches are already mitigated by the FTL. Others base their finding on a theoretical flash model, mostly by applying the metrics for read, write, and erase for raw flash chips or derive their results purely based on simulation. Even though theoretical effectiveness can be proven in this way, there can be quite a substantial discrepancy with regard to real-life SSDs. This can be seen in some papers, where the approaches are also verified on real SSDs, but the results are not as good as anticipated or derived by simulation. Moreover most papers base their experiments on only one type of SSD, neglecting the fact that their results could differ attributed to the employed SSD.

4 Methodology

For getting insights into SSD behavior, the read and write performance of the solid-state disks was measured. To stress all devices with the same access patterns, we developed a tool (similar to uFlip[1] and IOmeter[2]) that allows us to perform benchmarks on the devices. The tool is able to read and write different access patterns from/to the devices. The page size the tool uses is adjustable. Figure 1 outlines the access patterns we used to benchmark the SSDs. The first test pattern is sequentially accessing n pages. This pattern is

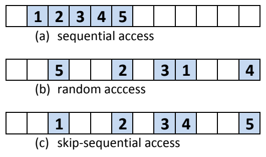


Figure 1: Access Patterns

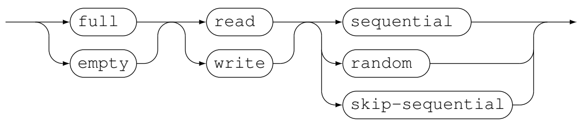


Figure 2: Measurement Combinations

[1]http://uflip.inria.fr/∼uFLIP/
[2]http://www.iometer.org

common in database servers when scanning through a table where no tailored access paths are available. Similarly, sequential writes are typical during log-flush operations. The second pattern is randomly accessing all pages of the test file. We ensured that each page gets accessed only once and that the pattern is repeatable. This pattern is often seen in databases when accessing pre-selected leaf pages in a B-tree. Random writes occur when updating several database tuples at a time or when flushing modified pages back to disk. The final pattern we tested is called *skip-sequential*, which accesses pages sequentially, but skips randomly over some pages. This pattern can be observed when a set of pages needs to be accessed, e.g., when – upon reading scattered database pages – the I/O requests are performed using ascending/descending physical addresses. Hence, this pattern evolves from the random pattern by pre-sorting the page numbers. This helps conventional disks to minimize head movement and, thus, reduces access times.

We set up a testing environment to benchmark all SSDs using the same hardware platform. To measure the device's energy consumption, we attached the computer to a measurement device. This enables us to keep detailed track of the devices' power consumption in addition to the performance measurements. A detailed description of the measurement setup can be found in [SHH10]. Using the previously described setup, we ran several tests against all SSDs and recorded their performance and energy consumption. The tests were run on a 1GB file filled with random data. We repeated the tests using varying page sizes and queue depths. First, we measured read and write patterns on a nearly empty drive (denoted by *empty* in Figure 2). After these measurements, we filled the drive with random data, while preserving the 1GB test file, and ran the same tests again. Figure 2 shows the benchmark combinations we ran for each device. We verified our results for sequential and random access by comparing it to results obtained by IOmeter and got more or less the same performance figures ($\pm 10\%$).

5 Experimental Results

In this section, we will present our measurement results for each SSD individually and discuss the observations. We measured using 32K pages for bandwith and 2 – 8K pages for IOPS measurements.[3] After we have shown all results, we will compare them and derive common patterns.

SSD1 – SuperTalent FSD32GC35M 32GB This SSD is the oldest one we tested. Results clearly show slow performance under all tested patterns as well as heavily degraded write performance. On the other hand, as the results show, random read is as fast as sequential read. SuperTalent states in the data sheet that this device can perform over 58,000 IOPS, a number we could not even get close to. Unfortunately, no further information about page sizes or queue depths is given.

SSD2 – Mtron MSP-SATA7525 The next SSD shows improved performance compared to SSD1, as depicted in Figure 5. Still, random writing is tremendously slower than other access methods. The SSD's data sheet tells a lot more about the parameters used for

[3] According to [BJB09], 32KB is the preferable page size for SSDs.

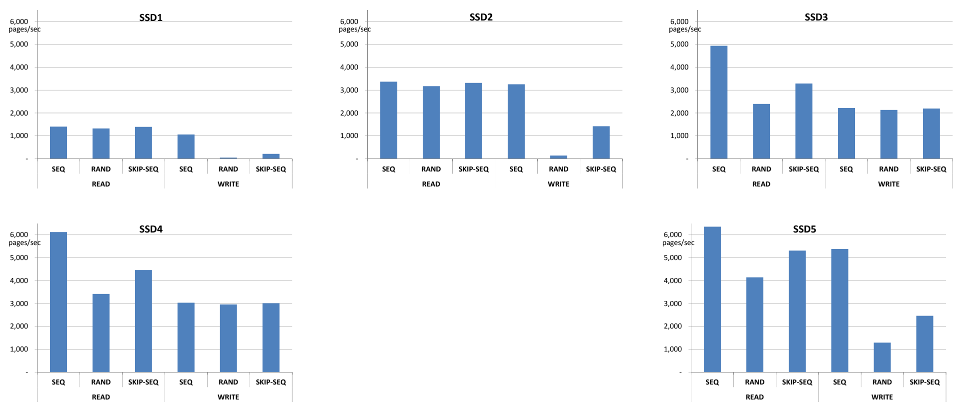


Figure 3: Performance Measurements Using Different SSD Types

testing. It documents page sizes, queue depths, and access patterns used; therefore, the measurements are far more comprehensible. Though, we could not reproduce the stated performance, but get closer to it than we could for the first SSD we tested.

SSD3 – Intel X25-M G1 SSD3 represents the first Intel generation. As shown in Figure 5, this drive is really good at sequential reading, while random reading is comparatively slow. Then again, all write patterns are performing equally well. This is a significant operational difference compared to the first two SSDs. Intel's data sheet documents the queue depth, but not the page size used for benchmarking sequential access patterns. For random accesses, the page size is mentioned. We were unable to get the same performance, even with the same parameters as in the data sheet.

SSD4 – Intel X25-M G2 The next generation of Intel SSDs came with the additional feature *TRIM support*[4]. Figure 5 indicates improved overall performance for all patterns. Nevertheless, the disk is showing the same challenges as the first generation. We could get closer to the performance reported in the data sheet, but were still unable to reach the advertised 35,000 IOPS. At least, we were able to measure the same sequential read bandwidth as stated (not shown in this figure).

SSD5 – Crucial RealSSD According to the manufacturer's data sheet, the device can read up to 60,000 and write up to 45,000 pages/second. Our own measurements show quite a different picture. While reading on SSD5 is faster than on all other SSDs we tested, random writing stresses this device remarkably. Although the data sheet promises 60,000 IOPS for random read, we could not get even close to this number.

6 Results Interpretation

After we presented individual measurement results for each SSD, we are going to examine some common patterns observed on more than one device. In this section, we will also

[4]`http://t13.org/Documents/MinutesDefault.aspx?keyword=trim`

examine common assumptions regarding SSDs and show that not all of them are true.

Random Access Interestingly, though SSDs do not have moving parts and therefore should not suffer from random access, in fact, they do. Our measurements show that random access may be substantially slower than sequential access. Dependent on the device, this effect ranges from -5% performance on SSD1 and SSD2 up to -50% on SSD3 and SSD4. Therefore, sequential accesses should still be preferred over random accesses, although it is not as vital as on hard disks.

Considering the break-even point for selecting index-based access over sequential table scan, there is now a shift on SSDs. For conventional hard disks, a rule of thumb says to use an index-based scan only if the selectivity is below 1 - 3%, otherwise to scan the whole table sequentially. On solid-state disks, the selectivity factor can be shifted to higher percentages. Because of the different performance characteristics of solid-state disks, it is not possible to spot a clear break-even point. For example for SSD1 and SSD2, break-even would be at ∼90%, while on SSD3 and SSD4, break-even is at only ∼50%, due to their worse random access performance. On SSD5, break-even lies at ∼75%, thus a considerable divergence is visible for SSDs. Our results show that there is a trend towards faster write support on SSDs. Clearly, SSD1 and SSD2 suffer heavily from random writes, whereas the newer SSDs from Intel cope with them much better. SSD5, on the other hand, is again performing badly at random writing while providing fastest sequential writes.

Database query optimizers can decide between random and sequential access based on configurable disk parameters.Hence, the same care that has to be taken for conventional disks also has to be applied when using SSDs. Unfortunately, performance characteristics of SSDs are much harder to quantify than for spinning disks: On conventional disks, RPM, cache size, and bus delay are the only vital characteristics to estimate performance. On SSDs, there are no key performance indicators and characteristics can only be derived from measurements. The decision whether to write random pages (logically) in-place or to employ log-structured sequential writes strongly relies on the behavior of the underlying SSD. Therefore, optimizing algorithms for wrong device models can make overall performance even worse. Especially developers for flash-aware buffer algorithms have to consider that device-specific tweaks might be obsolete in no time.

Unstable Behavior While verifying the results using IOmeter, we observed another effect on SSD3. We did some changes to the source code of IOmeter to enable per-second tracking of measurement values. Using this tweaked version, we were able to get more detailed performance data from our devices. Figure 4 visualizes the write performance of SSD3 in pages/second on a per-second basis. As illustrated, every 4 to 5 seconds, performance is heavily degraded for about 3 seconds. We conclude, the drive is performing internal re-organization like freeing up flash blocks or searching for another writable block. We measured the same behavior for sequential writes, though the timespan between drop-offs was about 3 times longer. On SSD4 – the successor of SSD3 –, we measured similar behavior, although the performance drops during writes were not that severe.

While benchmarking SSD4, we had a look at the TRIM command introduced for this model and observed an interesting behavior. Figure 5 depicts our write-performance mea-

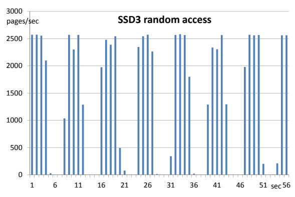


Figure 4: IOmeter per Second

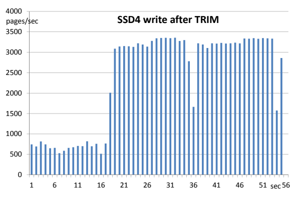


Figure 5: Write after TRIM

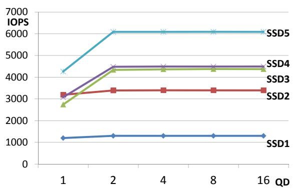


Figure 6: Queue Depth

surement right after deleting ∼130 GB of files on the drive and issuing corresponding TRIM commands to the drive. In this graph, a heavily degraded performance in the first half of the measurement is evident. Apparently, the SSD tries to free up flash blocks while we were simultaneously applying a write load to it. The proprietary FTL mapping particularly concerns device caching, block allocation, and garbage collection. All these mechanisms are software controlled and entirely hidden to the upper software layers. Hence, optimization decisions in the OS or DBMS may be counterproductive and sometimes even worsen the time-consuming house-keeping activities. As inferred from Fig. 4, write latency may extremely vary. While less than ∼400 μs in the best case, we have observed outliers of more than some hundred ms, that is a device-dependent variance of more than ∼200 – 500.[5] Compared to that, a magnetic disk with a device-dependent variance of ∼2 - 5 exhibits quite a stable access behavior and lends itself to reliable optimizer decisions.

Another aspect is a kind of heterogeneity among the SSD types present in a DBMS environment, where several heterogeneous SSDs may coexist in an application (or they may be dynamically exchanged). As a consequence, tailor-made algorithms for specific SSD types, e.g., concerning indexing or buffer management, are not very useful. The same arguments apply for specific workload optimizations (pure OLTP or OLAP processing, mixed workloads with varying degrees of reads/writes). A continuous adjustment or exchange of algorithms affected is not very practical in productive DBMS applications.

Read/Write Asymmetry As mentioned in the literature, reading flash pages is about 10 times faster than writing them because of intrinsic electrical properties. In conclusion, writing to SSDs should be equally slower than reading. Our measurements show that this is not true in general. SSD1 and SSD2, for example, do not exhibit degraded performance for (sequential) write, they are equally fast as sequential read. On all other SSDs, an asymmetry is measurable, but still not as bad as advertised.

Especially for buffer management algorithms, read/write asymmetries introduce a big potential for optimizations. On conventional disks, reading and writing cost the same; therefore, it does not make a big difference whether a clean or a dirty page gets evicted from the buffer. Considering solid-state disks, dependent on the device, the difference can now be significant. As mentioned earlier, current research papers already gave attention to this and flash-aware access algorithms were introduced. Nevertheless, it is crucial for these algorithms to know the exact properties of the underlying device, i. e., the precise read/write behavior, to enable optimizations.

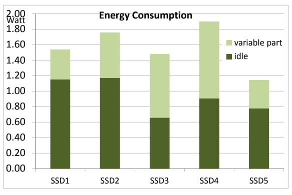
[5] Note, we observed similar variance factors at the level of DBMS operations, e.g., splits in B*-trees, but these were provoked by algorithmic or implementation weaknesses.

Slower When Full? SSDs have to erase flash blocks prior to writing new values to it. As a consequence, overwriting some blocks on a full disk should be much slower than writing to an empty disk. We verified this assumption by filling all drives with random data and repeating our tests afterwards. No significant differences were measurable. Therefore, the common advice to leaving some empty space on the SSD to help the FTL find free/erasable pages can be abandoned. In fact, this is a waste of storage space, since our measurements do not indicate differences between empty and full drives.

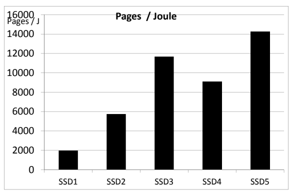
Impact of Queue Depth As mentioned earlier in this paper, the *queue depth (QD)*, that is, the number of concurrent requests needing access to the device, can make a great difference to the overall performance. Due to a technique called *Native Command Queueing (NCQ)*, the device can re-order the sequence of requests in the queue to optimize its internal access path and improve throughput. This was primarily invented for hard disks to optimize their access path along the spinning platter. SSDs can still increase performance by optimizing switches between different flash planes. Furthermore, because access latency of SSDs is very low, bus delays gain greater influence in the overall access delay. To minimize communication overhead, higher queue depths in combination with NCQ can also be used to send bulk requests to the drive [Gas08]. This reduces the overhead to $1/bulk\_size$ of the original overhead. SSD manufacturers know this fact and tweak their performance measurements accordingly. A high queue depth results in increased overall data throughput and higher IOPS, while a queue depth of 1 primarily minimizes access latency.

To gain more insights, we repeatedly measured various queue depths. By using a random read pattern, we give the FTLs a fair chance to optimize the queue. As Figure 6 indicates, the only significant improvement is between QD 1 and QD 2. Beyond this point, extending the QD did not improve data throughput. We did not expect this result, because manufacturers use even higher queue depths for their performance measurements. Also, current database servers do benefit from increased queue depths on conventional hard disks. As mentioned, some papers observed the same behavior [BJB09], while other papers explicitly recommend using longer queues [BdNSS10]. We see that it is not necessary to maintain long request queues for SSDs, thus database applications do not have to worry about getting maximum asynchronous I/O rates. A fair amount of outstanding requests is sufficient to keep an SSD at high bandwidth.

Energy Consumption As energy efficiency is getting a more and more critical factor for large data centers nowadays, we evaluated the SSDs' energy consumption. Figure 7(a) shows the absolute power consumption of the SSDs we tested. For this test, a sequential read pattern is used. Write patterns might consume even more energy. Obviously, the drives do consume energy when being idle; therefore, they are not as energy saving as expected. The SSDs' power profiles are similar to those of conventional hard disks, although their peak power consumption is considerably lower. Power consumption of conventional disks ranges from 4 – 6 Watts for mainstream disks to 9 – 14 Watts for enterprise server hardware. Figure 7(b) shows how many pages can be read by each SSD consuming one Joule of energy. As illustrated, $pages/Joule$ are constantly rising, thus newer SSDs are getting more energy efficient. On conventional disks we measured only 600 – 1800 $pages/Joule$. Anyway, a more differentiated comparison is cumbersome, because



(a) Energy Consumption in Watts



(b) Pages Read per Joule

Figure 7: Energy Consumption Measurements

of the different performance characteristics and their implications on energy efficiency. In general, the best performing SSDs are still the most energy efficient.

7 Limitations

We explored the performance of I/O patterns typically occurring in a DBMS environment. However, we were and still are not aware of the tricks and assumptions, e.g., massive I/O parallelism, of the manufacturers under which their performance behavior was achieved as reported in the data sheets. Our results sometimes indicate substantial deviations confirming that the "promised" device behavior may not be fully exploited by DB applications. Although we took care that our measurements are accurate and reliable, there might be some limitations to our approach. In order to keep up fair publishing policies, we do not want to hide them from the readers. We mainly focused on two ideas, why our measurements might not be 100% reliable.

SSD Choice The solid-state disks we tested were not fresh out-of-the-box, but were rather used by our research group for several benchmarks previously. Ranging from SSD1, which is approximately 3 years old, to our recently bought SSD5, all drives were used in varying degrees. Therefore, the observations and claims we made in this paper might only be true for our particular devices. Since the older drives (SSD1 to SSD3) do not support the TRIM command to free up flash pages, these devices might be worn out exceptionally. Due to a limited budget, we tested only one device for every model; therefore, we cannot not make assumptions for whole product models, but rather for single product instances.

Measurement Platform Choice The original use for our measurement track was to measure and optimize energy consumption. For this reason, we only used a small server board for testing, which might not have a high-performance SATA bus controller. This might bottleneck our measurements. In order to eliminate this possibility, we verified our measurements using a dedicated SAS controller card. Nevertheless, the results stayed the same and we were unable to get the performance promised in the data sheets. We even switched the entire hardware to a different system, which did not lead to an improvement. We can not resolve all doubts for sure, but we assume our measurements were correct and there is in fact a major gap between manufacturer's data sheets and real-world performance.

8 Conclusion

As the measurements clearly discover, each SSD exhibits a differing performance profile. We were able to identify some common patterns and outlined areas that are improving continuously, e.g., write performance. Still, manufacturers' claims about their drives' performance can not be blindly trusted. Because a common benchmarking procedure does not exist, performance claims of data sheets can hardly be reproduced in real-world scenarios. Due to the advancement of the internal FTL, larger write caches, and TRIM support, we believe that the often mentioned drawbacks of SSDs, e.g., slow writes, will soon disappear. Also, write endurance of SSDs is constantly rising and we do no longer have to care about destroying the disk by constantly writing to it. As we have proven by our experiments, every drive has its own characteristics. Optimization towards a single drive or against flash-chip access characteristics is no longer suitable under these circumstances. A lot of literature focuses on improved algorithms for flash chips, although there are no bare-metal flash chips in server systems. As we have shown in Section 6, current solid-state disks embody unpredictable behavior and performance may sometimes drop unexpectedly. In order to use SSDs in time-critical operations, like meeting deadlines in a real-time DBMS, algorithms have to be aware of these characteristics to anticipate even worst-case situations. More generic algorithms – not adjusted to single SSD types, but able to handle a widespread of different device characteristics – would be better suited and rather eligible for DBMS use. To suit a specific device, its characteristics could be determined either offline – prior to using the device in a productive environment – or online – during use. Then, according to the measured properties, the algorithm could automatically tune itself to maximize its performance. We propose that this approach would be more sustainable, even over SSD generations with changing behavior and be, therefore, more useful than highly specialized algorithms fitting particular SSDs only. Our benchmarks of todays solid-state disks unveiled a lot of pitfalls, although these measurements are far from being complete. For example, focusing on smaller grained, longer running benchmarks could help identify a lot more peculiarities of SSDs. For example, long running stress tests could reveal resource exhaustion inside the FTL or, by cutting power to the SSDs, persistence tests could be performed. Nearly all of the measurements we ran have discovered another fact for SSDs; therefore, we think there is a lot more to detect. Hopefully, future SSD generations will no longer need special treatment by the upper layers, because FTLs will contain more and more logic. We propose that SSDs will soon unite the advantages of conventional hard disks combined with faster random access behavior.

References

[AGS$^+$09] Devesh Agrawal, Deepak Ganesan, Ramesh K. Sitaraman, Yanlei Diao, and Shashi Singh. Lazy-Adaptive Tree: An Optimized Index Structure for Flash Devices. *PVLDB*, 2(1):361–372, 2009.

[BdNSS10] Stephan Baumann, Giel de Nijs, Michael Strobel, and Kai-Uwe Sattler. Flashing Databases: Expectations and Limitations. In *DaMoN*, 2010.

[BJB09] Luc Bouganim, Björn Thór Jónsson, and Philippe Bonnet. uFLIP: Understanding Flash IO Patterns. In *CIDR*, 2009.

[CKZ09] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *SIGMETRICS/Performance*, pages 181–192, 2009.

[CMB$^+$10] Mustafa Canim, George A. Mihaila, Bishwaranjan Bhattacharjee, Kenneth A. Ross, and Christian A. Lang. SSD Bufferpool Extensions for Database Systems. *PVLDB*, 3(2):1435–1446, 2010.

[CPP$^+$06] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee, and Ha-Joo Song. System Software for Flash Memory: A Survey. In *EUC*, pages 394–404, 2006.

[DP09] Jaeyoung Do and Jignesh M. Patel. Join Processing for Flash SSDs: Remembering Past Lessons. In *DaMoN*, pages 1–8, 2009.

[Gas08] Geoff Gasior. Intel's X25-E Extreme Solid-state Drive. Technical report, The Tech Report, 2008.

[Gra09] Goetz Graefe. The Five-Minute Rule 20 Years Later (and How Flash Memory Changes the Rules). *Commun. ACM*, 52(7):48–59, 2009.

[KJKK07] Dongwon Kang, Dawoon Jung, Jeong-Uk Kang, and Jin-Soo Kim. mu-Tree : An Ordered Index Structure for NAND Flash. In *EMSOFT*, pages 144–153, 2007.

[KJKM09] S.-H. Kim, D. Jung, J.-S. Kim, and S. Maeng. HeteroDrive: Re-shaping the storage access pattern of OLTP workload using SSD. In *IWSSPS*, pages 13–17, 2009.

[KKN$^+$02] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A space-efficient Flash translation layer for CompactFlash systems. *IEEE Transactions on Consumer Electronics*, 48:366–375, 2002.

[KV08] Ioannis Koltsidas and Stratis Viglas. Flashing up the storage layer. *PVLDB*, 1(1):514–525, 2008.

[LPC$^+$07] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. A log buffer-based flash translation layer using fully-associative sector translation. *TECS*, 6(3), 2007.

[Man02] Charles Manning. YAFFS (Yet Another Flash File System) http://www.yaffs.net/, 2002.

[OHJ09] Yi Ou, Theo Härder, and Peiquan Jin. CFDC: A Flash-aware Replacement Policy for Database Buffer Management. In *DaMoN*, pages 15–20, 2009.

[PABG10] Ilia Petrov, Guillermo G. Almeida, Alejandro P. Buchmann, and Ulrich Gräf. Building Large Storage Based On Flash Disks. In *ADMS*, 2010.

[RKM09] David Roberts, Taeho Kgil, and Trevor N. Mudge. Integrating NAND flash devices onto servers. *Commun. ACM*, 52(4):98–103, 2009.

[SHH10] Daniel Schall, Volker Hudlet, and Theo Härder. Enhancing Energy Efficiency of Database Applications Using SSDs. In *C3S2E*, pages 1–9, 2010.

[THS$^+$09] Dimitris Tsirogiannis, Stavros Harizopoulos, Mehul A. Shah, Janet L. Wiener, and Goetz Graefe. Query Processing Techniques for Solid State Drives. In *SIGMOD*, pages 59–72, 2009.

[WGK09] Yongkun Wang, Kazuo Goda, and Masaru Kitsuregawa. Evaluating Non-In-Place Update Techniques for Flash-Based Transaction Processing Systems. In *DEXA*, pages 777–791, 2009.

[WKC07] Chin-Hsien Wu, Tei-Wei Kuo, and Li-Ping Chang. An efficient B-tree layer implementation for flash-memory storage systems. *TECS*, 6(3), 2007.