Enhancing Energy Efficiency of Database Applications Using SSDs

Daniel Schall
University of Kaiserslautern
Germany
schall@cs.uni-kl.de

Volker Hudlet
University of Kaiserslautern
Germany
hudlet@cs.uni-kl.de

Theo Härder
University of Kaiserslautern
Germany
haerder@cs.uni-kl.de

ABSTRACT

Presently, solid state disks (SSDs) are emerging as a disruptive storage technology and promise breakthroughs for important application properties. They quickly enter the enterprise domain and (partially) replace magnetic disks (HDDs) for database servers. To identify performance and energy use of both types of storage devices, we have built an analysis tool and measured access times and energy needed for them. Associating these measurements to physical IO patterns, we checked and verified the performance claims given by the device manufacturers. Using typical read/write access patterns frequently observed in IO-intensive database applications, we fathomed the performance and energy efficiency potential of a spectrum of differing storage devices (low-end, medium, and high-end SSDs and HDDs).

Cross-comparing measurements of identical experiments, we present indicative parameters concerning IO performance and energy consumption. Furthermore, we reexamine an IO rule of thumb guiding their energy-efficient use in database servers. These findings suggest some database-related optimization areas where they can improve performance while energy is saved at the same time.

1. MOTIVATION

So far, NAND flash memory (denoted as flash disk or solid state disk (SSD)) was primarily considered ideal for storing permanent data in embedded devices, because it is energy efficient, small, light-weight, noiseless, and shock resistant. Therefore, it is used in personal digital assistants (PDAs), pocket PCs, or digital cameras and provides the great advantage of zero-energy needs in idle or turned-off modes.

Nowadays, SSDs promise breakthroughs in energy saving, bandwidth (IOps), reliability, and volumetric capacity [8]; therefore, they seem to be a strong candidate to become the future store for permanent database data. While these properties provide new and highly desired processing qualities compared to magnetic disks (disk or HDD, for short)—today, the prevailing storage technology for database applications—, unit capacities and prices of SSDs also rapidly enter the range where they become competitive to disks. To evaluate their potential when mapping database data to such devices, we briefly sketch typical read/write models of disk and flash for database management systems (DBMSs, DB servers).

C^3S^2E-10, 2010, May 19-21, Montreal[QC, CANADA]
Editor: B. C. Desai
Copyright ©2010 ACM 978-1-60558-901-5/10/05 ...$10.00.

1.1 Read/Write Models

Disks are devices enabling very fast sequential block reads and, at the same time, equally fast writes, whereas random block read/writes are much slower (requiring substantial "mechanical time fractions"). *Blocks as units of physical IOs* can be configured to the needs of the DBMS applications using page sizes typically ranging between 4KB and 64KB. To hide the access gap between memory and disk, DBMSs use a large DB cache in memory (RAM) where (in the simplest case) each cache frame can keep a DB page which, in turn, can be mapped to a disk block. In most DBMSs, propagation of DB pages follows the update-in-place principle applying WAL [9].

Flash storage is divided into m equal blocks typically much larger than DB pages. *Flash blocks* normally contain b (32 – 128) fixed-size pages where a page ranges between 512B and 2KB. Flash pages cannot be written by arbitrary bit patterns, only the transition from 1 to 0 is possible. To set arbitrary pattern, a block must prior be erased, which sets all the bits back to ones. Thus, a written page cannot be *updated in-place* anymore, but only freshly written after the entire block is erased again. Hence, the block is the *unit of erasure* automatically done by the flash device when allocating an empty block. A page is the smallest and a block the largest unit of read, whereas the unit of write is always a page; using chained IO, the DBMS, however, can write $1 < i \leq b$ pages into a block at a time.

A critical concern called *write endurance* is the limited number of erase cycles, between 100,000 (older references) and >1,000,000 (most recent references). When a block reaches this erase cycle limit, it cannot be longer used and has to be marked as corrupted. To maximize the SSD's lifetime, a dedicated software component (using a proprietary algorithm whose exact behavior is unknown), the so-called *flash translation layer* (FTL), optimizes the propagation of page updates thereby uniformly reusing the blocks across the SSD. For this reason, it implements a kind of shadow block mechanism (using DRAM memory for directory information). Whenever pages are written in-place, the SSD device uses shadow blocks to (temporarily) store those modified pages. Finally, upon update propagation, all pages of

Device type	name	seq. read [MB/s]	seq. write [MB/s]	random read (4KB pages/s)	random write (4KB pages/s)	idle power [Watt]	price $/GB
Maxtor IDE 7.2K rpm (80 GB)	HDD1	52	52	∼90	∼90	7.1	0.65
WD SATA 10K rpm (150 GB)	HDD2	91	91	∼125	∼125	4.53	1.03
Fujitsu SAS 15K rpm (147 GB)	HDD3	179	179	∼250	∼250	12.8	1.20
Super*Talent (32 GB)	SSD1	65	50	<10.000	no info	no info	4.34
Mtron SATA 7525 (32 GB)	SSD2	130	120	12,000	130	1.44	17.6
Intel X25-M MLC (160 GB)	SSD3	250	70	35,000	3,300	0.06	2.93

Table 1: Device characteristics

the original SSD block (modified and old ones) are assigned to a new SSD block and, thus, preserve *DB page clusters*. This mechanism, called *wear leveling*, is entirely transparent to the client, i.e. the DBMS, such that all references to displaced pages, e.g., index pointers and other links, remain valid.

In recent years, write endurance concerns were often raised in the literature against the general use of SSDs in DB servers. However, we believe that this problem is solved in a satisfactory way by such wear-leveling techniques [14] automatically applied by the SSD device using an FTL.[1]

FTL implementations are optimized by the device manufacturer, hide flash-specific details from the upper layers, and aim at squeezing the best performance out of the chips. Since hardware vendors put a lot of effort in the development of FTLs, they do not want the implementation details to become public. Therefore, FTLs are black boxes with indeterministic behavior to the user. Although common practices like page-mapping are widely implemented in SSDs, the implementation varies from device to device, resulting in noticeable differences in access characteristics. A *good* FTL is crucial to the entire SSD performance [3] and newer generations dynamically accomplish maintenance tasks when idle.[2]

1.2 Comparison of Device Characteristics

SSDs are emerging as a disruptive storage technology [7] that will substantially improve both *performance* and *energy usage* of data-intensive software systems. Because neither magnetic disks nor SSDs have unique characteristics and can be described by some simple factors, we summarize the critical parameters needed in our evaluation for a spectrum of (server-quality) device types (low-end, medium, and high-end). As illustrated in Table 1, they exhibit substantial differences in all properties listed, which has to be considered, in particular, when they are used in a “heterogeneous storage device landscape”.

Compared to disks, SSDs exhibit key characteristics—especially beneficial for DB servers: very low latency, high performance of random reads (*rr*), and very low energy consumption. While the sequential read (*sr*) and the sequential write (*sw*) performance is comparable to disks, SSDs suffer from low random write (*rw*) performance. Therefore, to make the most out of this potential, SSDs can not simply replace disks as a permanent DB store. Their effective use will impact various system components, in particular, when the DB server architecture is addressed. A key to such an improvement is to make the IO-related algorithms energy-aware. In particular, the *rr*/*rw* asymmetry has to be carefully approached, e.g., by methods avoiding random writes to the extent possible or to rearrange mapping of modified DB blocks such that they can be sequentially written (*clustered writes*) to propagate multiple updates at a time.

In Table 1, sequential IO is comparable for both device types, whereas our indicative numbers confirm breakthrough behavior, in particular, for random reads and energy use. Nevertheless, SSDs are still quite expensive. A GB of SSD storage amounts to >∼3$ (but only <∼1$ for disk), but technology forecast predicts rapid market growth for it and, in turn, a dramatic decrease in the near future that it will roughly reach the disk[3] [5].

As first steps towards the proper embedding of SSDs into DB servers, we want to identify critical parameters influencing the IO behavior and the performance effects of IO-intensive DB operations. In particular, we will measure time and energy consumed by typical operation sequences to reveal the potential for improved energy efficiency of SSDs as compared to disks. The interesting question is how much DB applications can take advantage of the performance and energy potential of SSDs as sketched in Table 1.

Note, the characteristic figures in Table 1 are vendor-supplied (or, if not available, approximated (∼) by us without assuming disk cache influence), i.e., they are gained under (different) specific access models and operation environments. Hence, their actual behavior may substantially deviate from these values in specific applications.[4]

1.3 Contribution

This paper is organized as follows. Section 2 defines the term *energy efficiency* and outlines our measurement environment and the measuring principles applied. To approach the problems of measuring performance and energy use of DB servers, Section 3 introduces the types of measurements performed and access patterns used for them. As a major effort, we will check and cross-compare in Section 3.4 the figures for performance and energy use using IO-intensive access patterns typical for DB applications. Furthermore, we will analyze different generations of flash and disk devices to reveal their differences in performance and energy efficiency under DB-specific IO patterns.

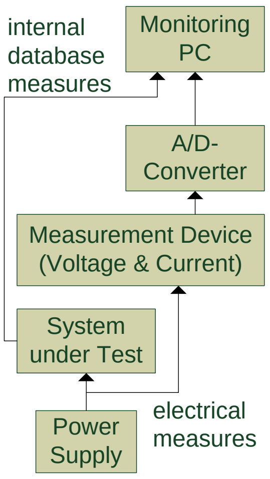
Section 3.5 reexamines a famous IO rule of thumb derived for disk-based query processing [10]. This experiment reveals that the break-even points for random read access vs

[1]A worst-case thought experiment with repetitive block overwrites indicates that the limit is reached after 51 years, http://www.storagesearch.com/ssdmyths-endurance.html.

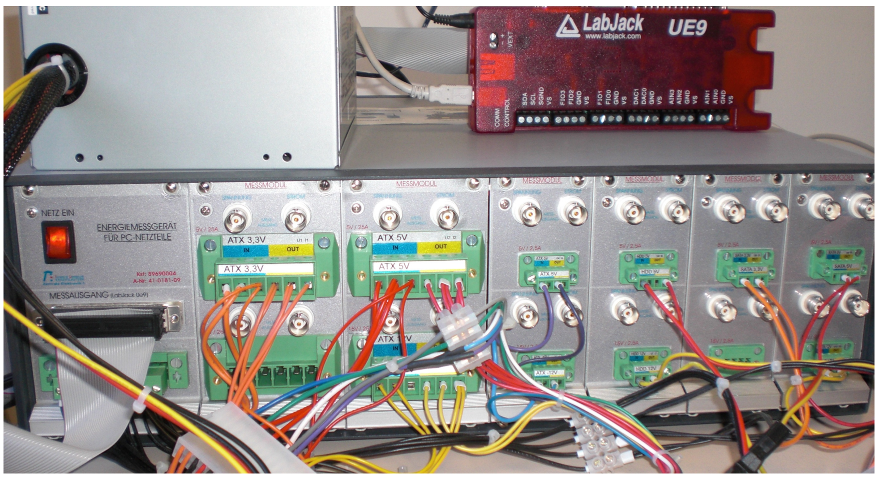
[2]http://hothardware.com/News/OCZ-and-Indilinx-Collaborate-On-New-SSD-Garbage-Collection-Scheme/

[3]We observed for a 64GB flash a reduction of ∼ 70% within 90 days, whereas a 750GB SATA only reached ∼ 7%.

[4]For example, the huge *rr* performance of SSD3 was achieved by using 25 concurrent read requests—an access model not matching typical DBMS requirements.



(a) Measurement setup



(b) Measurement device

Figure 1: Components of the measurement environment

file scan w.r.t. performance and energy efficiency are dramatically shifted to provide much more opportunities for index-based query processing. As our arguments to stimulate further discussion, Section 4 suggests for these findings database-related optimization areas where performance can be improved while energy is saved at the same time. After a discussion of the related work in Section 5, we wrap up our observations in Section 6 and point to urgent future work.

2. ANALYSIS TOOL

Given the public concern about energy waste, the focus on performance only is not sufficient for future generations of computer systems in general and DB servers in particular. Therefore, attention must be shifted from a solely performance-centric view to energy efficiency. For its evaluation, we will use the quotient of the system's work and its energy consumption:

$$EE = \frac{Work}{EnergyConsumption}$$

The system's work can be measured in units of various computational tasks like sorting operations, bytes processed or—most commonly in database environments—*transactions*. A lot of tools and benchmarks already exist to measure such performance aspects of DB servers. But the energy consumption is more difficult to analyze and requires a specific measurement setup for which we first outline the system components whose energy consumption shall be measured.

2.1 System under Test

Since we want to disclose the energy use of a commodity hardware platform, that could be used to run a DMBS, we chose a standard x86 consumer PC.

Hardware Components The hardware setup is a Mini-ITX mainboard, with a 64-bit CPU having two cores and 4 GB of main memory. An IDE and a SATA disk are connected to provide persistent storage. Since the measurement tool is designed for arbitrary ATX mainboards, the component setup can be adjusted. So far, we analyzed six drives, three traditional hard disks and three SSDs, ranging from low-end to high-end devices. The three hard disks are: a *Maxtor DiamondMax Plus 9* (7200 rpm, IDE), referred to as HDD1 and a *Western Digital VelociRaptor* (10.000 rpm, SATA), referred to as HDD2, and a *Fujitsu MBA3147RC* (15.000 rpm, SAS) disk, referred to as HDD3. The SSDs examined are a *SuperTalent FSD32GC35M* (SSD1), an *Mtron MSP-SATA7525* (SSD2), and an *Intel X25-M* (SSD3). All three are SATA devices.

Software As OS, we used the 32-bit server edition of Ubuntu Linux 9.10, Kernel version 2.6.31. To minimize spurious influences, we disabled all unneeded background tasks.

2.2 Measurement Setup

Unfortunately, ATX mainboards and power supplies do not provide integrated measuring points to record the components' energy consumption. Therefore, we set up a measurement kit for undisturbed reading voltage and current values of relevant power lines, as shown in Figure 1(a). We also added an interface to measure internal database parameters in conjunction with electrical measures. This interface acts as a probe to be injected into an arbitrary DBMS where it collects internal measures and makes them available over a Java-RMI interface. By combining both measurements, we are able to analyze the energy consumption of typical database operations. Since we did not use DB software in this paper, no further explanation of the internal database parameters will be given. Instead, the measurement of the electrical values will be explained in detail:

Custom measurement hardware, consisting of ten voltage and current meters, connects the power supply and the system's hardware components. That way, it can record the energy use on every relevant power line and the total consumption of the component in real-time.

Measuring principles To evaluate the results online, we converted the analog measurement values to digital signals using an A/D-Converter. Having the digital representation of the values measured, it gives us the opportunity to process them on a PC and combine them with other data, like performance-critical values recorded in DB server components mentioned earlier. We developed a software that reads data from different sources, e.g. the electrical data or the internal database statistics, and outputs them together on

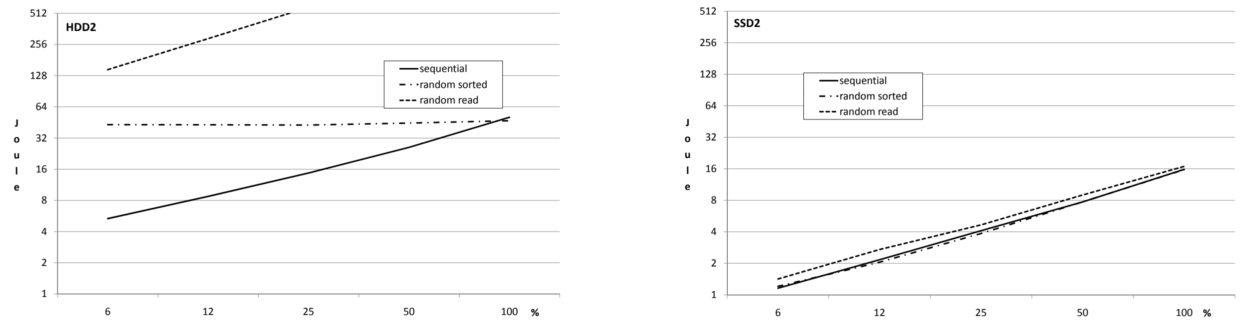


Figure 2: Energy cost of the basic read access patterns on disk and SSD

screen or in a protocol file. The software also filters and transforms the input values to merge different power lines to the device's total energy consumption. We added a remote interface for integrating the measurement device into our benchmarking suites or profiling applications.

This setup reads the energy consumption of the mainboard and the storage devices—sufficient to analyze their energy efficiency. Recording the CPU's and main memory's power separately would be highly desirable for identifying the energy use of memory. But because their power lines are very tiny and the processor is highly sensitive to voltage diffusion, integrated measurement points must be supplied by the manufacturer to enable such a power recording. Hence, we have to leave this kind of analysis for future research.

Figure 1(b) shows the front of the measurement device. The power supply that is used for the system under test is shown in the upper right, standing on top of the measurement device. On the device's front, the green inlets and outlets of the power lines are shown in the middle of the box as well as the gray data bus at the front left. The power lines are lead inside the box where the current and voltage gets measured and led back afterwards to power to the hardware under test. A more detailed explanation of the inner circuits can be found below. The data bus is used to connect the device with the A/D-Converter.

Inside the device The measurement must not tamper the voltages at the power lines, since PC components are sensitive to voltage drops. Therefore, the current measurement is done using current transformers with inductive measurement, which do not influence the current flow. At the same time, the voltage is measured using voltage dividers on a shunt circuit. Both measurements are transformed in voltages ranging from 0 to 5 Volts and forwarded over the data bus to the A-D-Converter.

Since all measurement instruments are highly precise, the total measurement error of the setup is below 2.2% of the measured value. This precision was validated by using comparative tests and should be sufficient for our needs.

3. PERFORMANCE AND ENERGY USE

The goal of our research efforts is to measure the energy efficiency of DB servers as complex and widely used software systems. Their architecture often adheres to a hierarchically layered reference model, e.g. the well-known *five-layer model* [9]. It is obvious that in such a complex software system the behavior of the upper system layers is heavily influenced by the layers it is based on. Each layer introduces further abstractions and its internal behavior has to contribute to optimal DB server performance. Although the design of optimal DB architectures still leaves questions open after more than 30 years of active research and development [11], the emerging awareness for energy efficiency raises novel design and optimization aspects.

3.1 Basic Access Patterns

To approach this new energy efficiency challenge, it makes sense to explore the layered DB server architecture in a bottom-up manner and to start at the most influential layer (for storage-related energy use), namely the file system.

At this interface, the DB server requests and propagates persistent data pages containing the user records. These pages have fixed size, typically 4 or 8 KB in case of transaction processing systems. From the DB server perspective, most file requests follow one of three basic access patterns:

- *Sequential access* is mainly triggered by so-called scans which require all pages of a file to be read.
- *Random access* of data pages occurs if only distinct pages are requested—typically selected via adequate index structures, e.g. B-trees [2].
- *Random sorted (rs) access*, also called skip-sequential access, is similar to random access. This access pattern fetches distinct pages, but is performed in ascending or descending file order. In case of disk devices, such patterns minimize the mechanical arm movements.

A random access pattern can frequently be transformed into a random sorted pattern. If an arbitrary list of page addresses is given, sorting them enables unidirectional file accesses and, in turn, substantial speed-ups.

3.2 IO-related Measurements

In the following, we evaluate these three basic access patterns towards their impact on energy efficiency at the file interface. Since the DB server is running as an (unprivileged) OS application, the OS behavior can have a significant impact on the access patterns below the file interface. Beside the file system, there exist several components designed to harmonize the speed difference between the CPU clock cycles and storage devices being six to seven orders of magnitude slower. In the Linux kernel, a page cache, prefetching techniques, and adequate IO scheduling are launched to narrow this "access gap". But with these activities, energy usage is shifted towards processor and memory. We circumvented these OS optimization measures by implementing a so-called *Direct-IO* mode which helps to identify the real IO-related costs. The impact of OS-internal optimizations, especially caching and prefetching, and their influence on the energy efficiency of the access patterns is left for future work.

All read/write experiments were performed on a ∼1GB file using a page size of 8 KB. We did not tend to a specific file allocation, because, at least for SSDs, the FTL does not enable mapping control (nor influence of the update status of the selected device). For all measurements, buffers (except device caches) were disabled.

3.3 Analyzing Energy Consumption

To illustrate the device-based energy use for all access patterns, we refer in Figure 2 as an example to HDD2 and SSD2. To facilitate direct comparison, we used the same scale factor, but needed a log scale because of largely differing Joule values. Note that the X-axis denotes the percentage of pages read in either access mode, e.g., x% of *sequential access* means that only the first x% of the file are read; a search without index support implies a full file scan and the energy consumption of the 100% case. As one can see, the SSD's energy consumption is dramatically reduced, in case of *random access* to less than a hundredth of that of the same mode on hard disk.

Note, all access patterns approximately consume the same energy on SSDs, while *rr* accesses on HDDs exceed the energy needs of *sr* and *rs* accesses by far. Although this fact is intuitive, it has great impact on the design of IO-related algorithms of DB servers: Since *rs* access is considerably faster than *rr* access, state-of-the-art DB servers often sort addresses, when a set of pages is requested, to avoid random access, e.g. by *List Prefetch*[16] implemented in most DB servers.

Using SSDs, pre-sorted page access is no longer needed, since accessing the data randomly does not downgrade performance or energy efficiency. But current DB servers do not differentiate between SSD and disk, hence they are unable to perform specific optimizations tailored to SSDs. By re-engineering existing DB algorithms towards optimal SSD utilization, we could leverage the potential of saving energy while improving performance by exchanging hard disks with SSDs.

3.4 Evaluation of Read and Write Patterns

We measured the performance and energy consumption of the three access patterns on all six storage devices. Figure 3 shows the experimental results for read access on the left-hand side and write access on the right-hand side, where all graphs show for each of the three access patterns the results derived for each device considered (from HDD1 to SSD3). In all experiments, we measured *electric power* (in Watt) and *access performance* (in *pages/second*). Derived from these measurements, the bar graphs at the bottom of Figure 3 illustrate *energy efficiency*, denoted in *pages/Joule*. All graphs contain average results gained from differing workloads (of the same pattern) which accessed varying amounts of pages.

The graphs at the top of Figure 3 show the electric power input of the devices, which is—unsurprisingly—higher for hard disks. While the power drain is approximately constant for reading and writing from/to hard disks, SSDs seem to draw more power while writing.

The graphs in the middle of Figure 3 reveal—when directly compared to SSDs—the dramatically bad random-access performance for hard disks. Using the same experiment, SSDs provided a more balanced behavior, but exhibited deteriorated performance for write accesses.

Eventually, the graphs at the bottom of Figure 3 show the energy efficiency of the devices under the evaluated patterns. SSDs outperform hard disks by far, although the newer SSDs show greatly improved efficiency.

The read benchmarks show, that hard disk performance scales with rotation speed and all access patterns behave similarly on the three disks. Thus, hard disk performance is predictable by having a look at the disks specifications. On the other hand, the SSDs do not show steady performance, e.g., SSD3 is best for sequential reads, while SSD2 is faster for random reads. Because SSDs do not contain mechanical parts that need to be positioned, identical figures for all three patterns (on a given SSD) would have been expected. The deviations of SSD3 for the three access patterns (see *readPerformance* in Figure 3) are particularly eye-catching. We cannot explain the differences for sure, but we assume the internal structures (FTL, hardware) of the flash devices need more time switching to farther addresses. Therefore, the performance of SSDs seems to be definitely less predictable than for hard disks (see Section 1.1).

After having evaluated the read behavior of the patterns on each device, it is important to check whether the results remain the same when taking into considerations write operations as well. As can be seen in Figure 3, the SSD behavior for the various access patterns exhibits substantial differences as compared to the read scenarios for the same access patterns. When the read performance of the SSD is solely considered, one can clearly see that the performance of the SSDs is independent from the employed access pattern, except SSD3. Focusing on the write performance, SSD1 and SSD2 show a behavior similar to hard disk, having comparable sequential write rates, but devastatingly bad random write behavior. The *rw* pattern reveals that the performance is even worse compared with hard disks. Surprisingly, SSD3 draws again a completely different picture: While the access pattern influenced the write performance of SSD1 and SSD2, it does not affect the performance of SSD3 where the three patterns just slightly differ compared with their respective read patterns.

Note that contrary to earlier findings [13] where SSDs are generally considered slower than hard disks for random write operations, SSD3 outpaces all hard disks. This result sounds promising, because in this way the behavior of SSDs becomes more predictable. Due to the low energy consumption, it finally delivers the best energy efficiency figures of all devices considered.

3.5 Reexamining an IO Rule of Thumb

As a first application of our measurement tool and based on the insights gained by our experiments described in the previous section, we reexamined the famous rule of thumb [10] derived for disk-based accesses: If the selectivity of a query is less than 1%–3% (depending on file and page parameters), then use index-based access (*rr*); otherwise scan the whole file (*sr*).

To this end, we compared access times and energy consumption of traditional disks to SSDs in a variety of empirical test runs in a single-user environment. All results presented here are indicative for our problem and gained from experiments using a contiguously allocated file of one

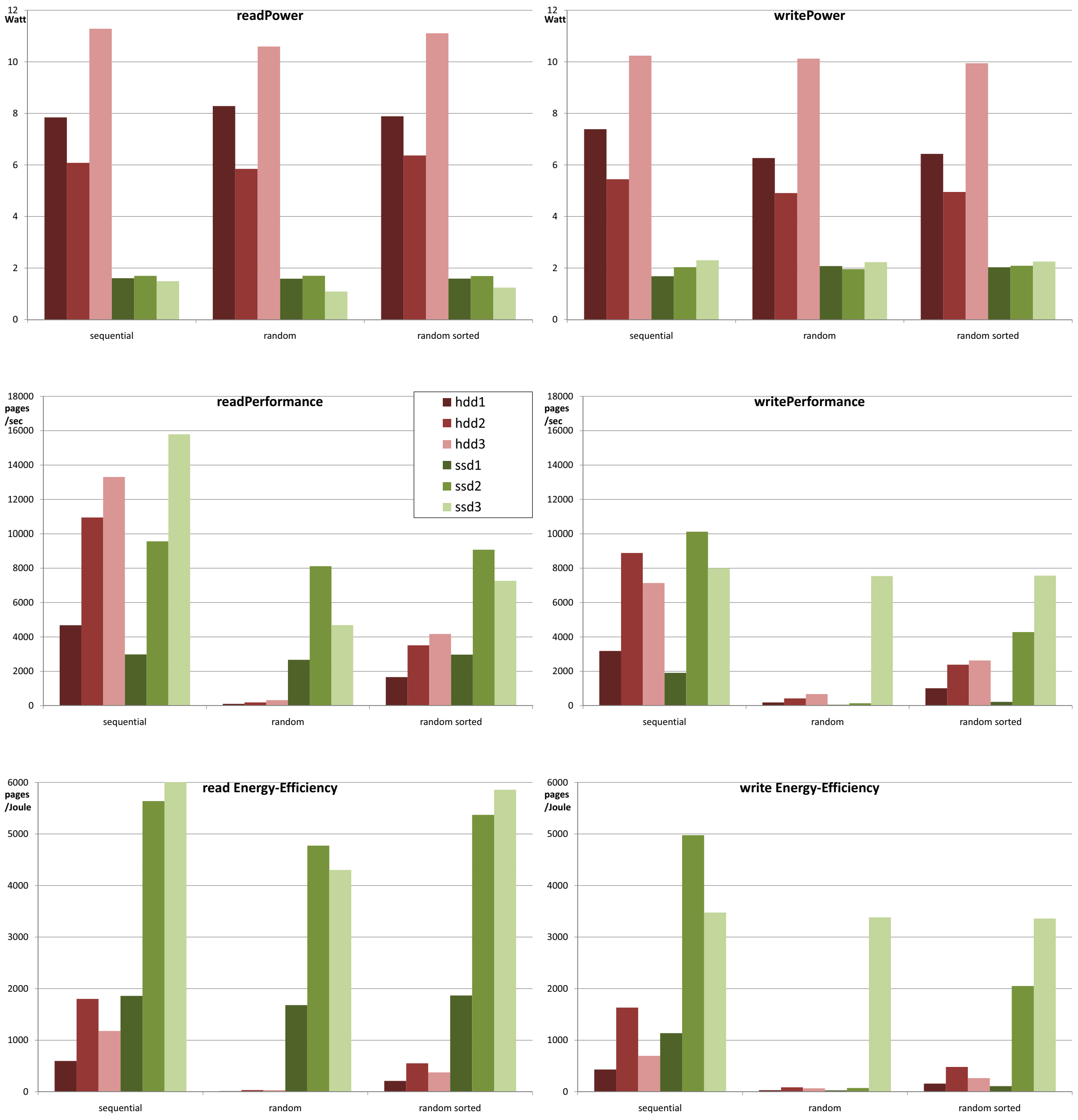


Figure 3: Comparing the differing aspects of read and write behavior for all devices considered

Break-even criterion	HDD1	HDD2	HDD3	SSD1	SSD2	SSD3
Performance	2%	2%	2 %	93%	83%	39%
Energy Efficiency	2%	2%	2 %	90%	85%	28%

Table 2: Break-even points: random read access vs scan

GB with a page size of 8 KB. These runtime conditions favor the disk-based results, because file fragmentation and multi-user interferences increase disk arm motions and, in turn, access time and energy use.

To avoid an overloaded illustration, Figure 4 records only for HDD2 and SSD2 the elapsed times for the different access patterns over the percentage of pages fetched and identifies their performance break-even points for rr/rs against sr. (Please, compare only the differing access patterns per device with each other). Because of the bad time ratio (seek + rotational delay vs data transfer) of HDD2, our experiment confirms with 2% the rule of thumb. Access pattern rs compensated this bad ratio and shifts the break-even point close to 100%. Note, however, the undisturbed unidirectional disk

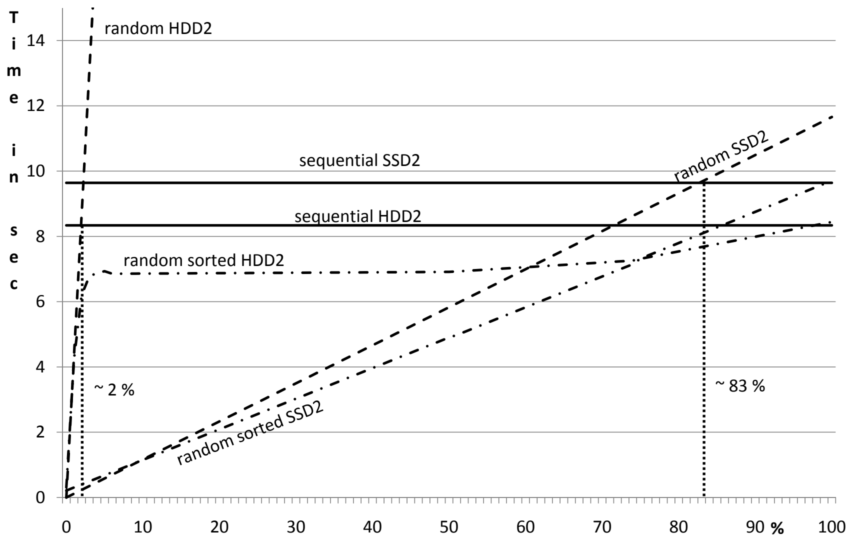


Figure 4: Identifying break-even points

arm movement is decisive for this result quality. For SSDs, *rr* and *rs* times more or less coincide because of the missing "mechanical time". The deviation in Figure 4 may be due to device-related optimizations, e.g. prefetching. Hence, without any runtime restrictions, their break-even points are shifted to >80%, which may open substantial room for IO optimizations and, in turn, for energy saving.

For all three HDD types (see Table 2), the break-even points with 2% for performance confirm the classical rule of thumb for disks, because their ratios of *rr* and *sr* (but not the absolute access times) remained comparable. The break-even points for energy efficiency are then a consequence of this behavior.

In contrast, the performance break-even points of the SSD types are clearly shifted to much higher values. While that of SSD1 is more than 90%, that of SSD3 is lower than 40%, which can be roughly explained by the changed operation speeds (see Table 1). And again, the break-even points for the energy efficiency follow the performance ratio, because SSDs have approximately the same energy consumption for all three read patterns. Therefore, the only variable in the break-even calculation is the performance.

As a summarization of our experiments, Table 2 aggregates indicative values for break-even points. These values clearly confirm that the device characteristics of HDD2 can be taken as representatives for magnetic disks. In case of SSDs, the selection of a representative is not so clear. Using SSD2, we also can approximate the behavior of SSD1. However, the characteristics of the high-end device SSD3 must be clearly distinguished from the slower ones. Nevertheless, the results of all devices considered show a strong correlation between performance (in terms of time) and energy consumption.

4. OPTIMIZATION OPPORTUNITIES

Given our measurement tool, we are now able to track the internal behavior of a DB server in correlation with the resulting energy consumption. This enables us to derive specific statements about the energy efficiency of software and hardware changes in the database world. As already shown, SSDs provide new challenges and DB servers need to be made aware of the new hardware characteristics. Making our vision more concrete, our future plans regarding these new challenges and technologies will be outlined in the following.

Optimizing the layer mapping In order to exploit the new SSD technology, it is not sufficient to simply replace traditional hard disks. Instead, design decisions in all DB server layers need to be reconsidered to make the algorithms aware of the new IO properties. An already intensively tackled problem is SSD-aware DB buffer management (related to the second layer). Of course, the read/write asymmetry lends itself to tailor-made page replacement algorithms where replacement decisions are clearly favoring write avoidance instead of preferring pages only read to stay in the buffer, i.e., clean pages can be cheaply reloaded in case of a rereference. In higher layers, index structures and path processing operators [12] have to be addressed to readjust their IO behavior anew [21]. For example, they take advantage of the extremely improved reading speed (> 10,000 IOps) and their algorithmic behavior can be optimized towards speed and energy use. Reevaluating the old "five-minute rule" [10] also should reveal characteristics for improving performance as well as giving the system the opportunity to pro-actively save energy.

Trading performance for energy efficiency Current database query optimizers try to increase the performance of the system, i.e. speed up the query execution. Their decision is based on plans which estimate the execution cost in terms of performance. Adding energy-based metrics to the query optimizer would enable the DB server to choose energy-optimized plans, which could lower the energy consumption of the system. By finding a (dynamic) trade-off between performance and energy efficiency, we can select between maximal performance and energy saving, thus creating an energy-adjustable database. [23]

Database reorganization Disk-based file performance is very sensitive to clustering, whereas SSDs support all clustering, fragmentation, and partitioning methods equally well. Hence, the urgency of reorganization runs can be figured out under the energy efficiency aspect. Established techniques like *ListPrefetch* have to be reconsidered for SSDs and may become obsolete. Furthermore, alternative storage allocation techniques like SSD-aware column-based mappings [21] have to be taken into account, because they can leverage SSDs which, in turn, result in higher performance respectively lower energy consumption.

SSD as a building block towards an energy-proportional system The idea of energy-proportional systems [1] is that the energy consumption is proportional to the system load. In a large data center environment, unused SSDs could be switched to standby mode where they consume even less energy (close to zero-energy needs in idle mode). Given a smart data partitioning, the amount of needed SSDs is proportional to the current system load, resulting in an energy-proportional storage layer. As SSDs have no moving parts, these devices can be switched to standby mode and back to active mode within one second compared to up to 30 seconds (e.g. for HDD3). Similar ideas have been introduced for hard disks [4], but, due to faster mode switches and to the significantly lower energy footprint of a single SSD, the proportionality can be controlled in a more fine-grained way. We will explore this in future work.

Optimized hybrid configurations The discriminating differences between hard disks and SSDs can be exploited by combining both device classes in a DB server. The measurement tool gives us the opportunity to identify the true energy characteristics of devices which can be used to compose an energy-optimized DB server under cost-effective aspects. By selectively placing DB objects on SSD (e.g. indexes) and hard disk (rarely referenced user data or archive data) to leverage the advantages of both device classes, we are going to improve the overall access performance while saving energy.

Energy-aware database benchmarks Some existing benchmarks already take energy consumption into account, e.g. *SPECpower_ssj2008* and *TPC-Energy*. However, the benchmark mentioned first is too coarse-grained and the latter does address peak load situation only, i.e. periods where the system is fully stressed. Benchmarks which evaluate more realistic, varying load distributions, e.g. observed in [20], do not exist. To evaluate the energy efficiency of a DB server, the behavior during the entire runtime—not only under peak-load—should be considered. Nonetheless, a more realistic benchmarking methodology, observing idle times and marginal resource utilization, is missing, although these load situations provide big energy saving potentials. As more competing approaches regarding energy efficiency are emerging, the more urgent are specific benchmarks to enable result comparisons within the world-wide research community and to determine progress in energy saving at all.

5. RELATED WORK

The optimization ideas sketched in Section 4 span a wide spectrum of DB-oriented research. Contribution [6] has described a slightly different view, where similar optimization opportunities are outlined at a conceptual level. Here, we have delivered some quantitative arguments as a basis of their realization.

So far, there are already many approaches to optimize the mapping layer, but most of them only utilizing the speed potential of SSDs. The read/write asymmetry mentioned is explicitly addressed by the CFDC algorithm (Clean First, Dirty Clustered): Clean pages are replaced first, and dirty pages are kept in the buffer to build clusters according to their SSD locations (i.e., the flash block, where they are stored), if possible, and are then propagated in larger granules. At the same time, specific measures guarantee scan-resistant replacement of buffered pages [17]. In the meantime, we have prepared a first broad study concerning energy efficiency of a variety of buffer management algorithms when different types of HDDs or SSDs are used as external storage [18].

Some approaches addressing optimizations in higher layers explicitly focus on the dramatic speed of random reads gained by the use of SSDs. Contribution [19] proposes the conversion of DBMS-internal processing strictly using sequential IO to algorithms that use a mixture of index-based random IO and sequential IO (only if appropriate) to process less data in less time. Hence, scans, joins, and sorting are reconsidered when supported by SSDs [5]. Furthermore, several approaches focus on indexing where the data is left on magnetic disks and the only the indexes are kept on SSDs or the index mechanism is aware of the specific SSD behavior [15].

Trading performance for energy efficiency is currently discussed in controversial way. An empirical study [15] claims that there is some (limited) potential of power conservation, if the query optimizer takes the power cost of query execution plans into consideration. Hence, such an approach will be able to reduce the power consumption of database servers or to explicitly control the tradeoffs between power usage and system performance. In this way, the system could offer to the user cheaper, but slower or faster and, therefore, more expensive query execution plans. On the other hand, a broad and detailed study appearing just recently strictly advocates that there is no energy-performance tradeoff in centralized server systems: *The most energy-efficient configuration is typically the highest performing one* [22]. A strong reason for this result is said to be the missing energy-proportional behavior of current computer systems. Hence, there is plenty of demand to have a deeper look to this controversy and also to include distributed databases into this consideration.

According to our knowledge, the remaining issues raised as optimization opportunities were so far neither approached by conceptual work nor by experimental studies.

6. CONCLUSIONS

In this paper, we have outlined our measurement environment primarily used to identify the energy needs of external storage devices. For IO-intensive DB applications, we have explored the performance behavior and the related energy consumption of a wide spectrum of HDDs and SSDs under typical access patterns.

We have shown that the SSD technology provides a new level of performance and of energy efficiency, but it will not pay off by simply replacing disks with SSDs. Instead their new characteristics need to be exploited by the DB server. Our work shows a technique to identify the characteristics of SSDs and we outlined some important areas we will be touching in the future to make DB servers more energy efficient. Fortunately, we are able to use XTC [12], our native XML DB server, to respond with energy-aware algorithms and, at the same time, to immediately measure the gained success (in Joules).

The rule of thumb considered indicated an important area of optimization in DB servers. In form of heuristics, several other rules [10] are also dominating internal DB server processing. Therefore, they should be checked as well to identify algorithm adaptations needed to match SSD properties, especially because SSDs show wide performance diversity, not predictable by hardware characteristics. However, the more competing approaches are emerging, the more urgent are specific energy efficiency benchmarks to enable world-wide result comparisons and to determine progress in energy saving at all.

7. REFERENCES

[1] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[2] R. Bayer and E. M. McCreight. Organization and Maintenance of Large Ordered Indices. *Acta Informatica*, 1:173–189, 1972.

[3] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song. System software for flash memory: A survey. In *Proceedings of Int. IEEE/IFIP Conference on Embedded and Ubiquitous Computing (EUC-06)*, pages 394–404, 2006.

[4] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Proceedings of ACM/IEEE Conference on Supercomputing*, pages 1–11, 2002.

[5] G. Graefe. The Five-minute Rule: 20 Years Later and How Flash Memory Changes the Rules. In *Proceedings of ACM SIGMOD Workshop "Data Management on New Hardware (DaMoN)"*, Beijing, China, 2007.

[6] G. Graefe. Database Servers Tailored to Improve Energy Efficiency. In *Proceedings of Int. EDBT Workshop on "Software Engineering for Tailor-made Data Management"*, pages 24–28, Nantes, France, 2008.

[7] J. Gray. Tape is Dead, Disk is Tape, Flash is Disk, RAM Locality is King. `http://research.microsoft.com/en-us/um/people/gray/talks/Flash_Is_Good.ppt`, Dec. 2006. Powerpoint Talk, Microsoft.

[8] J. Gray and B. Fitzgerald. Flash Disk Opportunity for Server Applications. *Queue*, 6(4):18–23, 2008.

[9] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[10] J. Gray and P. J. Shenoy. Rules of Thumb in Data Engineering. In *Proceedings of Int. Conf. on Data Engineering*, pages 3–12, San Diego, Calif., 2000.

[11] T. Härder. DBMS Architecture - Still an Open Problem (Keynote). In *Proceedings of German National Conference "Databases in Business, Technology, and Web"*, pages 2–28, Karlsruhe, West Germany, 2005. LNI-P65, GI.

[12] M. P. Haustein and T. Härder. An Efficient Infrastructure for Native Transactional XML Processing. *Data & Knowledge Engineering*, 61(3):500–523, 2007.

[13] S.-W. Lee and B. Moon. Design of flash-based dbms: an in-page logging approach. In *Proceedings of ACM SIGMOD Conference*, pages 55–66, New York, NY, USA, 2007. ACM.

[14] A. Leventhal. Flash storage today. *Queue*, 6(4):24–30, 2008.

[15] Y. Li, B. He, Q. Luo, and K. Yi. Tree Indexing on Flash Disks. In *Proceedings of Int. Conference on Data Engineering*, pages 1303–1306, Shanghai, China, 2009.

[16] P. O'Neil and E. O'Neil. *Database-Principles, Programming and Performance*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[17] Y. Ou, T. Härder, and P. Jin. CFDC - A Flash-aware Replacement Policy for Database Buffer Management. In *Proceedings of ACM SIGMOD Workshop "Data Management on New Hardware (DaMoN)"*, Providence, RI, 2009.

[18] Y. Ou, T. Härder, and D. Schall. Performance and Power Evaluation of Flash-Aware Buffer Algorithms. In *submitted 2010*.

[19] M. A. Shah, S. Harizopoulos, J. L. Wiener, and G. Graefe. Fast scans and joins using flash drives. In *Proceedings of DaMoN '08*, pages 17–24, 2008.

[20] A. Z. Spector. Distributed Computing at Multi-dimensional Scale (Keynote). In *Proceedings of Int. Middleware Conference*, 2008.

[21] D. Tsirogiannis, S. Harizopoulos, J. L. Shah, Mehul A.and Wiener, and G. Goetz. Query Processing Techniques for Solid State Drives. In *Proceedings of ACM SIGMOD Conference*, pages 59–72, Providence, RI, 2009.

[22] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the Energy Efficiency of a Database Server. In *Proceedings of ACM SIGMOD Conference*, 2010.

[23] Z. Xu, Y. Tu, and X. Wang. Exploring Power-Performance Tradeoffs in Database Systems. In *Proceedings of Int. Conference on Data Engineering*, 2010.