Performance and Power Evaluation of Flash-Aware Buffer Algorithms

Yi Ou, Theo Härder, and Daniel Schall

University of Kaiserslautern, Germany
{ou,haerder,schall}@cs.uni-kl.de

Abstract. With flash disks being an important alternative to conventional magnetic disks, various design aspects of DBMSs, whose I/O behavior is performance-critical, and especially their I/O architecture should be reconsidered. Taking the distinguished characteristics of flash disks into account, several flash-aware buffer algorithms have been proposed with focus on flash-specific performance optimizations. We introduce several basic principles of flash-aware buffer management and evaluate performance and energy consumption of related algorithms in a DBMS environment using both flash disks and magnetic disks. Our experiments reveal the importance of those principles and the potential of flash disks both in performance improvement and in energy saving.

1 Introduction

In recent years, green computing gained a lot of attention and visibility by public discussion concerning energy waste and, in turn, global warming. As a consequence, thermal management is seriously addressed in IT to reduce power usage, heat transmission, and, in turn, cooling needs. Here, we consider the role DBMSs can play for green IT.

So far, flash disks were primarily considered ideal for storing permanent data in embedded devices such as personal digital assistants (PDAs) or digital cameras, because they have no mechanical parts and are energy efficient, small, light-weight, noiseless, and shock resistant. Furthermore, they provide the great advantage of zero-energy needs in idle or turned-off modes. Because of breakthroughs in bandwidth (IOps), energy saving, reliability, and volumetric capacity [1], flash disks will also play an increasingly important role in DB servers. However, to optimize their performance and, at the same time, energy efficiency, especially their I/O architecture must be reconsidered.

The most important building blocks of flash disks are flash memory and flash translation layer (FTL). Logical block addresses are mapped by the FTL to varying locations on the physical medium. This mapping is required due to the intrinsic limitations of flash memory [2]. The implementation of FTL is device-related and supplied by the disk manufacturer. Several efforts were recently made to systematically benchmark the performance of flash disks [1,3,4]. The most important conclusions of these benchmarks are:

P. García Bringas et al. (Eds.): DEXA 2010, Part I, LNCS 6261, pp. 183–197, 2010.

- For sequential read-or-write workloads, flash disks often achieve a performance comparable to high-end magnetic disks.
- For random workloads, the performance asymmetry of flash disks and their difference to magnetic disks is significant: random reads are typically two orders of magnitude faster than those on magnetic disks, while random writes on flash disks are often even slower than those on magnetic disks.

As an example, one of our middle-class flash disk achieves 12,000 IOPS for random reads and only 130 IOPS for random writes of 4 KB blocks, while high-end magnetic disks typically have more than 200 IOPS for random I/O [1]. Interestingly, due to the employment of device caches and other optimizations in the FTL, page-level writes with strong spatial locality can be served by flash disks more efficiently than write requests without locality. In particular, many benchmarks show that flash disks can handle random writes with larger *request size*s more efficiently. For example, the bandwidth of random writes using units of 128 KB can be more than an order of magnitude higher than writing at units of 8 KB. In fact, a write request of, say 128 KB, is internally mapped to 64 *sequential* writes of 2-KB flash pages inside a flash block. Note that sequential access is an extreme case of high spatial locality.

1.1 Basic Principles

In DBMS buffer management, the fact "whether a page is read only or modified" is an important criterion for the replacement decision [5]. To guarantee data consistency, a modified buffer page has to be written back to disk (called *physical write* or *page flush*), before the memory area can be reused. Hence, if the replacement victim is dirty, the process or thread requesting an empty buffer frame must wait until page flush completion—potentially a performance bottleneck.

This criterion is now much more important in our context, because, for flash disks, the average cost of a page write (including block erasure) may be two orders of magnitude higher than that of a page read. At a point in time running a given workload, if a clean page p is to be re-read n times and a dirty page q to be re-modified m times ($n \sim m$), the buffer manager should replace p in favor of q, because the benefit of serving m write requests directly from the buffer is much higher than the cost of n repeated flash reads.

Yet the total cost of page flushing is not linear to the number of page flushes. As introduced above, write patterns strongly impact the efficiency of flash writing; hence, they have to be addressed as well. An intuitive idea is to increase the DB *page size* so that we can write more efficiently. In most systems, the page size is the unit of data transfer between the buffer layer and the file system (or the raw device directly). It would be an attractive solution if the overall performance could be improved this way, because only a simple adjustment of a single parameter would be required. However, a naive increase of the page size generally leads to more unnecessary I/O (using the same buffer size), especially for OLTP workloads, where random accesses dominate. Furthermore, in multi-user environments, large page sizes favor thread contentions. Hence, a more sophisticated solution is needed.

Even with flash disks, maintaining a high hit ratio—the primary goal of conventional buffer algorithms—is still important, because the bandwidth of main memory is at least an order of magnitude higher than the interface bandwidth of storage devices. Based on the flash disk characteristics and our discussion so far, we summarize the basic principles of flash-aware buffer management as follows:

P1 Minimize the number of physical writes.
P2 Address write patterns to improve the write efficiency.
P3 Keep a relatively high hit ratio.

1.2 Contributions

Previous research on flash-aware buffer management focused on flash-specific performance optimizations. However, an important question remains open: how well do the flash-aware algorithms perform on conventional magnetic disks? This question is important because, due to the lower $/GB cost, magnetic disks will certainly remain dominant in the near future, therefore enterprises have to deal with the situations where both kinds of devices co-exist. Another interesting aspect ignored in all previous works is the energy consumption of related algorithms, which calls more and more attention in server environments and is often critical in environments where flash devices are deployed in the first place, e.g., mobile data management. The major contributions of this paper are:

- We accomplish an extensive performance study of related algorithms in a DBMS-based environment using a variety of flash disks and magnetic disks. This device-sensitivity study is missing in all previous works, where evaluation was often performed on a single simulated flash device. Furthermore, the performance study emphasizes the basic principles introduced in Sect. 1.
- We examine the energy consumption of the system running these algorithms using a tailor-made energy measurement device. Our experiments reveal a strong correlation between system performance and energy consumption and demonstrated the great energy-saving potential of flash disks.

The remainder of the paper is organized as follows. Sect. 2 sketches related work. Sect. 3 discusses one of the flash-aware buffer algorithms in detail. Our experiments are presented in Sect. 4. The concluding remarks are given in Sect. 5.

2 Related Work

LRU and CLOCK [6] are among the most widely-used replacement policies. The latter is functionally identical to Second Chance [7]: both of them often achieve hit ratios close to those of LRU.

CFLRU [8] is a flash-aware replacement policy for operating systems based on LRU. It addresses the asymmetry of flash I/O by allowing dirty pages to stay in the buffer longer than clean pages. The LRU list of CFLRU is divided into two regions: the *working region* at the MRU (most recently used) end of the list, and the *clean-first region* at the LRU end, where clean pages are always selected as victims over dirty pages. In this way, the buffer area for dirty pages is effectively increased—thus, the number of flash writes can be reduced.

LRU-WSR [9] is a flash-aware algorithm based on LRU and Second Chance, using only a single list as auxiliary data structure. The idea is to evict clean and cold-dirty pages and keep the hot-dirty pages in buffer as long as possible.

REF [10] is also a flash-aware replacement policy based on LRU. It maintains an LRU list and has a *victim window* at the MRU end of the list, similar to the clean-first region of CFLRU. Victim pages are only selected from the *victim blocks*, which are blocks with the largest numbers of pages in the victim window.

CFLRU and LRU-WSR do not address the problem of write patterns, while REF does not distinguish between the clean and dirty states of pages. To the best of our knowledge, CFDC [11] is the only flash-aware algorithm that applied all the three basic principles P1 to P3 introduced in Sect. 1. The study of the energy behavior of related buffer management algorithms distinguishes our work from that of [4], where the impact of the device type (flash disk or magnetic disk) on the energy efficiency of a complete database system is examined and discussed.

3 The CFDC Algorithm

For comprehension, we repeat the essential properties of the CFDC (Clean-First Dirty-Clustered) algorithm.

3.1 The Two-Region Scheme

CFDC manages the buffer in two regions: the *working region* W for keeping *hot* pages that are frequently and recently revisited, and the *priority region* P responsible for optimizing replacement costs by assigning varying priorities to page clusters. A parameter λ, called *priority window*, determines the size ratio of P relative to the total buffer. Therefore, if the buffer has B pages, then P contains λ pages and the remaining $(1-\lambda)\cdot B$ pages are managed in W. Note, W does not have to be bound to a specific replacement policy. Various conventional replacement policies can be used to maintain high hit ratios in W and, therefore, prevent hot pages from entering P.

The parameter λ of CFDC is similar to the parameter *window size* (w) of CFLRU. The algorithm REF has a similar configurable *victim window* as well. For simplicity, we refer to them uniformly with the name “window size”. If a page in P is hit, the page is moved (promoted) to W. If the page hit is in W, the base algorithm of W should adjust its data and structures accordingly. For example, if LRU is the base algorithm, it should move the page that was hit to the MRU end of its list structure. In case of a buffer fault, the victim is always first selected from P. Only when all pages in P are fixed[1], we select the victim from W. Considering recency, the newly fetched page is first promoted to W.

[1] The classical pin-use-unpin (or fix-use-unfix) protocol [12] is used for page requests.

3.2 Priority Region

Priority region P maintains three structures: an LRU list of clean *pages*, a priority queue of *clusters* where dirty pages are accommodated, and a hash table with cluster numbers as keys for efficient cluster lookup. The cluster number is derived by dividing page numbers by a constant *cluster size.*

In our context, *spatial locality* refers to the property of contiguously accessed DB pages being physically stored close to each other. A *cluster* is a set of pages located in proximity, i.e., whose page numbers are close to each other. Though page numbers are logical addresses, because of the space allocation in most DBMSs and file systems, the pages in the same cluster have a high probability of being physically neighbored, too. The size of a cluster should correspond, but does not have to be strictly equal to the size of a flash block, thus information about exact flash block boundaries are not required.

CDFC's principles of victim selection are:

- Inside P, clean pages are always selected over dirty pages. If there is no clean page available, a *victim cluster* having the lowest priority is selected from the priority queue.
- The oldest page in the victim cluster will be evicted first, if it is not re-referenced there. Otherwise, it would have been already promoted to W.
- The priority of the victim cluster is set to minimum and will not be updated anymore, so that the next victim pages will still be evicted from this cluster, resulting in strong spatial locality of page evictions.

For a cluster c with n (in-memory) pages, its priority $P(c)$ is computed according to Formula 1:

$$P(c) = \frac{\sum_{i=1}^{n-1} |p_i - p_{i-1}|}{n^2 \times (globaltime - timestamp(c))} \tag{1}$$

where $p_0, ..., p_{n-1}$ are the page numbers ordered by their time of entering the cluster. The algorithm tends to assign large clusters a lower priority for two reasons: 1. Flash disks are efficient in writing such clustered pages. 2. The pages in a large cluster have a higher probability of being sequentially accessed.

The sum in the dividend in Formula 1, called *inter-page distance* (IPD), is used to distinguish between randomly accessed clusters and sequentially accessed clusters (clusters with only one page are set to 1). We prefer to keep a randomly accessed cluster in the buffer for a longer time than a sequentially accessed cluster. For example, a cluster with pages $\{0, 1, 2, 3\}$ has an IPD of 3, while a cluster with pages $\{7, 5, 4, 6\}$ has an IPD of 5.

The purpose of the time component in Formula 1 is to prevent randomly, but rarely accessed small clusters from staying in the buffer forever. The cluster timestamp *timestamp*(c) is the value of *globaltime* at the time of its creation. Each time a dirty page is inserted into the priority queue ($min(W)$ is dirty), *globaltime* is incremented by 1.

If a page in P is hit, it will be promoted to W. A newly fetched page in P will also be promoted to W. In both cases, page $min(W)$ is determined by W's victim selection policy and demoted to P. It does not have to be unfixed, because it is just moved inside the buffer. If $min(W)$ is clean, it is simply appended to the LRU list of clean pages. If it is dirty, the *globaltime* is incremented and we derive its cluster number and perform a hash lookup using this cluster number. If the cluster does not exist, a new cluster containing this page is created with the current *globaltime* and inserted to the priority queue. Furthermore, it is registered in the hash table. Otherwise, the page is added to the existing cluster tail and the cluster position in the priority queue is adjusted.

After demoting $min(W)$, the page to be promoted, say p, will be removed from P and inserted to W. If p is dirty and its containing cluster c is not a victim cluster, we know that p is promoted due to a buffer hit. We update the cluster IPD including the timestamp. This will generally increase the cluster priority according to Formula 1 and cause c to stay in the buffer for a longer time. This effect is desirable, because the remaining pages in the cluster will probably be revisited soon due to locality. In contrast, the cluster timestamp is not updated, when pages are added to a cluster, because they are demoted from W.

The time complexity of CFDC depends on the complexity of the base algorithm in W and the complexity of the priority queue. The latter is $O(\log m)$, where m is the number of clusters. This should be acceptable due to $m \ll \lambda \cdot B$, where $\lambda \cdot B$ is the number of pages in P. Furthermore, with the priority queue and its priority function, both temporal and spatial locality of the dirty pages are taken into account, thus potentially high hit ratios and improved runtime performance can be expected.

The CFDC approach implies the NoForce and Steal strategies for the logging&recovery component [13], which, however, is the standard solution in most DBMSs. In practice, page flushes are normally not coupled with the victim replacement process—most of them are performed by background threads. Obviously, these threads can benefit from CFDC's dirty queue, where the dirty pages are already collected and ordered. The two-region scheme makes it easy to integrate CFDC with conventional replacement policies in existing systems.

4 Experiments

4.1 Hardware Environment

The system under test (SUT) has an Intel Core2 Duo processor and 2 GB of main memory. Both the OS (Ubuntu Linux with kernel version 2.6.31) and the DB engine are installed on an IDE magnetic disk (system disk). The test data (as a DB file) resides on a separate magnetic/flash disk (data disk). The data disks, listed in Tab. 1, are connected to the system one at a time.

A tailor-made power measurement device, consisting of ten voltage and current meters, connects the power supply and the system's hardware components, as depicted in Fig. 1. The device does not tamper the voltages at the power lines, because the hardware components are sensitive to voltage variations. Instead, it

Table 1. Disk drives used in the test

name	device type	idle (W)	peak (W)	interface
HDD1	WD WD800AAJS 7200 RPM	5.3	6.3	SATA
HDD2	WD WD1500HLFS 10000 RPM	4.5	5.7	SATA
HDD3	Fujitsu MBA3147RC 15000 RPM	8.4	10.0	SAS
SSD1	SuperTalent FSD32GC35M	1.3	2.1	SATA
SSD2	MTRON MSP-SATA-7525-032	1.2	2.0	SATA
SSD3	Intel SSDSA2MH160G1GN	0.1	1.2	SATA

measures the current using current transformers with inductive measurement, and the voltage using voltage dividers on a shunt circuit. Both measurements are forwarded over a data bus to the A/D-Converter, which allows the signals being processed by a monitoring PC in real-time.

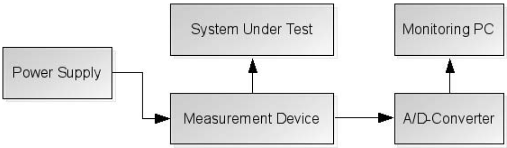


Fig. 1. Power measurement setup

Using this setup, we are able to precisely measure the energy consumption of the SUT's major parts of interest: the data disk (denoted as SATA, although HDD3 is measured over the SAS power lines), the system disk (denoted as IDE), and the remaining components on the mainboard (denoted as ATX) including CPU and RAM. For a time period T, the average power $\bar{P}$ is given by Formula 2:

$$\bar{P} = \frac{1}{T}\int_0^T (v(t) \cdot i(t))dt \tag{2}$$

where $v(t)$ and $i(t)$ are the voltage and current as functions of time, and $\int_0^T (v(t) \cdot i(t))dt$ is the work, which is equal to the energy consumption E.

Tab. 2 lists the power profile of the major components of SUT. The idle column refers to the power values when the components are idle (0% utilization) but ready for serving requests, i.e., not in a service-unavailable state such as stand-by or hibernate, while the peak column refers to the power values when the components are under 100% utilization. An interesting observation can be made from this profile: ignoring the data disk, the idle power of the system is 57% of the peak power (25.9 W / 45.7 W). In other words, a lion's share of the power is consumed only to keep the system in a ready-to-service state. Note this "idle share" is even larger in practice, because the peak power can only

Table 2. Power profile of SUT

power line	components	idle (W)	peak (W)
SATA/SAS	data disk	(see	Tab. 1)
IDE	system disk (Maxtor 6Y080P0)	6.3	12.2
ATX	CPU, RAM, and mainboard chips	19.6	33.5

be reached in some extreme conditions, where all the hardware components are fully-stressed at the same time. With the data disk, this observation still holds, because similar ratios exist between the idle and peak power of HDDs, while SSDs had too low power values to have a large impact on the idle/peak power ratio of the overall system. Furthermore, this observation is not specific to the test machine discussed here, because most state-of-the-art servers and desktop computers have similar idle/peak power ratios.

4.2 Software Settings

In all experiments, we use a native XML DBMS designed according to the classical *five-layer reference architecture*. For clarity and simplicity, we only focus on its bottom-most two layers, i.e., the *file manager* supporting page-oriented access to the data files, and the *buffer manager* serving page requests. Although designed for XML data management, the processing behavior of these layers is very close to that of a relational DBMS.

We deactivated the file-system prefetching and used *direct I/O* to access the DB file, so that the influences of file system and OS were minimized. The logging&recovery component is deactivated so that no extra I/Os for logging will influence our measurements. Our experiments are driven by two real-life OLTP page reference traces and only the two layers introduced were involved in the processing, thus the workload is I/O intensive. All experiments started with a *cold* DB buffer. Except for the native code responsible for direct I/O, the DB engine and the algorithms are completely implemented in Java. CFDC and competitor algorithms are fully integrated into the XML DBMS and work with other components of the DB engine.

We cross-compared five buffer algorithms, including the flash-aware algorithms CFLRU, LRU-WSR, REF, CFDC and the classical algorithms represented by LRU. The *block size* parameter of REF, which should correspond to the size of a flash block, was set to 16 pages (DB page size = 8 KB, flash block size = 128 KB). To be comparable, the *cluster size* of CFDC was set to 16 as well. The $|VB|$ parameter of REF (the number of allowed victim blocks) was set to 4, based on the empirical studies of its authors. Furthermore, we used an improved version of CFLRU which is is much more efficient at runtime yet functionally identical to the original algorithm.

4.3 Measuring Spatial Locality

Both CFDC and REF improves the spatial locality of page flushes. We define the metric *cluster-switch count (CSC)* to quantify the spatial locality of page flushes. Let $S := \{q_0, q_1, \ldots, q_{m-1}\}$ be the sequence of page flushes, the metric $CSC(S)$ reflects the spatial locality of S:

$$CSC(S) = \sum_{i=0}^{m-1} \begin{cases} 0, \text{if } q_{i-1} \text{ exists and in the same cluster as } q_i \\ 1, \text{otherwise} \end{cases} \tag{3}$$

The *clustered writes* of CFDC are write patterns with high spatial locality and thus minimized cluster-switch counts. Sequential writes are a special case of clustered writes, where pages are updated in a forward or reverse order according to their locations on the storage device. If $d(S)$ is the set of distinct clusters addressed by S and S is a sequential access pattern, we have $CSC(S) = |d(S)|$. In the context of magnetic disks, if we set the cluster size equal to the track size, then $CSC(S)$ approximates the number of disk seeks necessary to serve S.

Compared to CFDC, the sequence of dirty pages evicted by the algorithm REF generally has a much higher CSC, because it selects victim pages from a *set* of victim blocks and the victim blocks can be addressed in any order. However, this kind of write requests can also be efficiently handled by flash disks, if the parameter VB is properly set. Because the sequence of dirty pages evicted can be viewed as multiple sequences of clustered writes that are interleaved with one another, we call the approach of REF *semi-clustered writes.*

Let $R := \{p_0, p_1, \ldots, p_{n-1}\}$ be the sequence of page requests fed to the buffer manager, we further define the metric *cluster-switch factor* (CSF) as:

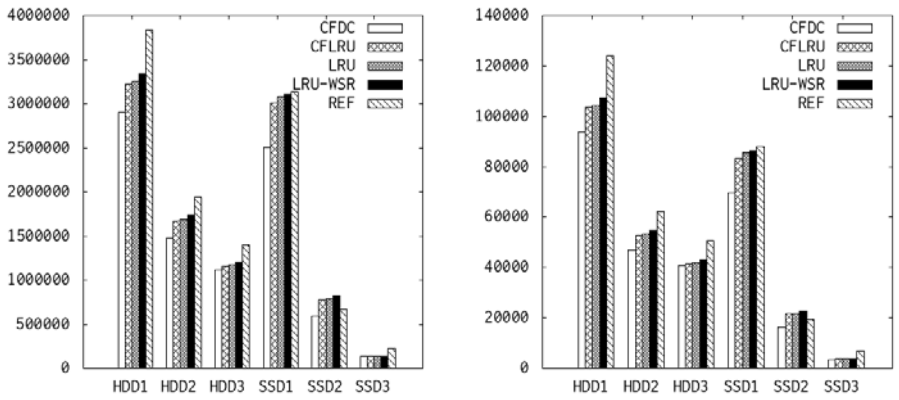
$$CSF(R, S) = CSC(S)/CSC(R) \tag{4}$$

CSF reflects the efficiency to perform clustering for the given input R. To compare identical input sequences, it is sufficient to consider the CSC metric alone.

4.4 The TPC-C Trace

The first OLTP trace was obtained using the PostgreSQL DBMS. Our code integrated into its buffer manager recorded the buffer reference string of a 20-minutes TPC-C workload with a scaling factor of 50 warehouses.

We ran the trace for each of the five algorithms and repeat this on each of the devices listed in Tab. 1. The recorded execution times and energy consumptions are shown in Fig. 2. Since our workload is I/O-intensive, the device performance has a strong impact on the overall system performance, e.g., SSD3 reduced the average runtime (average over the algorithms) by a factor of 21 compared with HDD1 and by a factor of 19 compared with SSD1, while the corresponding energy-saving factors are 25 and 19 respectively. The CFDC algorithm had the best performance on all of the devices, with a maximum performance gain of 22% over CFLRU on SSD1. Interestingly, even on the magnetic disks, CFDC



(a) execution times (ms)

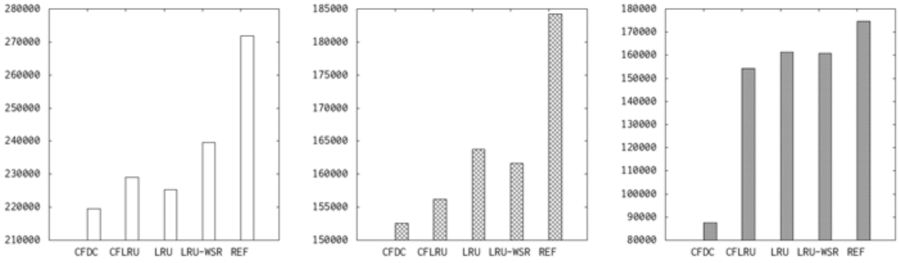
(b) energy consumption (J)

Fig. 2. Performance and energy consumption running the TPC-C trace

and CFLRU had a better performance than LRU, which, in turn, outperformed LRU-WSR even on the flash disks. In most configurations, REF had the longest execution times due to its lower hit ratio and higher number of page flushes, with the exception on SSD2, where its semi-clustered writes seems to be best accommodated by that specific device.

Similar to magnetic disks, it is common for flash disks to be equipped with a device cache. Very often, it can not be deactivated or re-sized by the user or the OS, and its size is often undocumented. Obviously, the DB buffer in our experiment should be larger than the device caches. Otherwise, the effect of the DB buffer would be hidden by them. On the other hand, if the DB buffer is too large, the difference between our algorithms would be hidden as well, since with a large-enough buffer, there would be hardly any I/O. Based on these considerations, we used a buffer size of 8000 pages (64 MB) for this experiment, because the largest known device cache size is 16 MB of HDD3. The difference between the execution times of the algorithms becomes smaller on SSD3 (see Fig. 2) due to two reasons: 1. The I/O cost on SSD3 is much smaller than on other devices, yielding the buffer layer optimization less significant; 2. This device has supposedly the largest device cache, since it is the newest product among the devices tested.

Fig. 3 shows the performance metrics that were constant across all the devices (device-independent): number of buffer faults (physical reads), number of page flushes (physical writes), and cluster-switch count (CSC). As shown in Fig. 3a and Fig. 3b, the clean-first algorithms (CFLRU and LRU-WSR) traded some of their hit ratios for a reduced page flush count, while REF suffered both from a higher number of buffer faults and a higher number of page flushes. Among all algorithms compared, the page flushes generated by CFDC had the highest spatial locality (Fig. 3c).



(a) buffer faults

(b) page flushes

(c) CSC

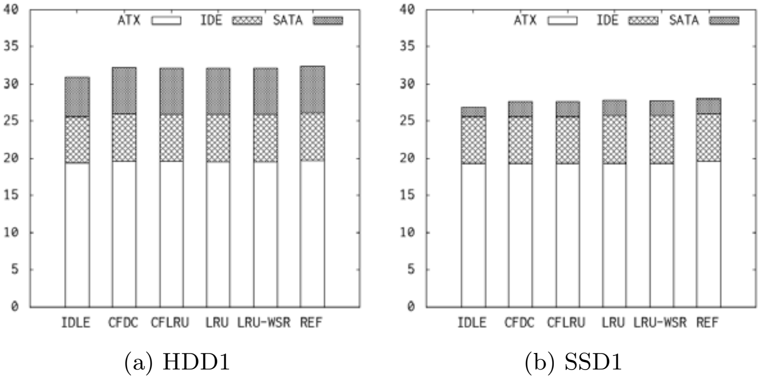
Fig. 3. Device-independent performance metrics

Comparing Fig. 2a with Fig. 2b, we can see a strong correlation between the execution times and the energy consumption. In particular, the best-performing algorithm was also the most energy-saving algorithm. For example, CFDC reduced energy consumption by 32% on HDD1 and by 71% on SSD3 compared with REF. This effect is further explained by Fig. 4, which contains a breakdown of the average working power of major hardware components of interest, compared with their idle power values. The figures shown for the configurations HDD1 and SSD1 are indicative[2]. Ideally, the power consumption of a component (and the system) should be determined by its utilization. But for both configurations, there is no significant power variation when the system state changes from idle to working. Furthermore, no clear difference can be observed between the various algorithms, although they have different complexities and, in fact, also generate different I/O patterns. This is due to the fact that, independent of the workload, the processor and the other units of the mainboard consume most of the power (the ATX part in the figure) and these components are not *energy-proportional*, i.e., their power is not proportional to the system utilization caused by the workload. However, due to the missing energy-proportional behavior of most system components, the elapsed time T of the workload (algorithm) almost completely determines its energy consumption E (note, $E = \bar{P} \cdot T$).

4.5 The Bank Trace

The second trace used here is a one-hour page reference trace of the production OLTP system of a Bank. It was also used in the experiments of [14] and [15]. This trace contains 607,390 references to 8-KB pages in a DB having a size of 22 GB, addressing 51,870 distinct page numbers. About 23% of the references update the page requested, i.e., the workload is read-intensive. Even for the update references, the pages must be first present in the buffer, thus more reads are required. Moreover, this trace exhibits an extremely high access skew, e.g., 40% of the references access only 3% of the DB pages that were accessed in

[2] They are similar for the other device types (IDE and ATX remain constant). For this reason, we have omitted the other configurations due to space limitations.

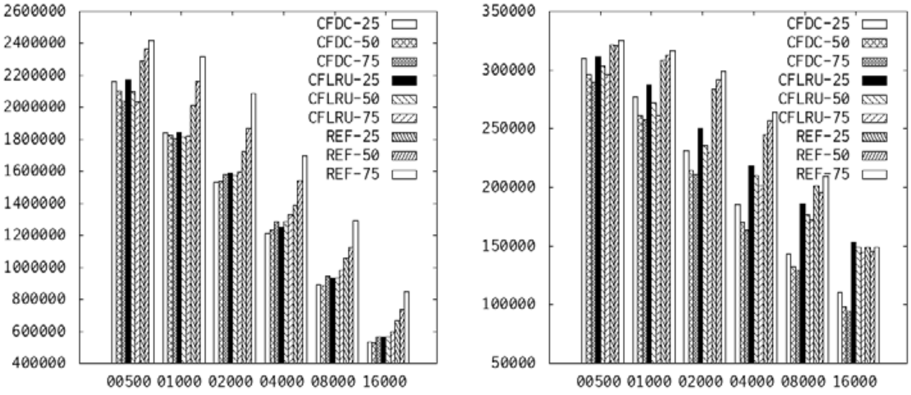


(a) HDD1

(b) SSD1

Fig. 4. Break-down of average power (W)

the trace [14]. We use this trace to examine the impact of buffer size on the performance and energy consumption of our algorithms. Hence, the buffer size varied from 500 to 16000 pages (factor 32).

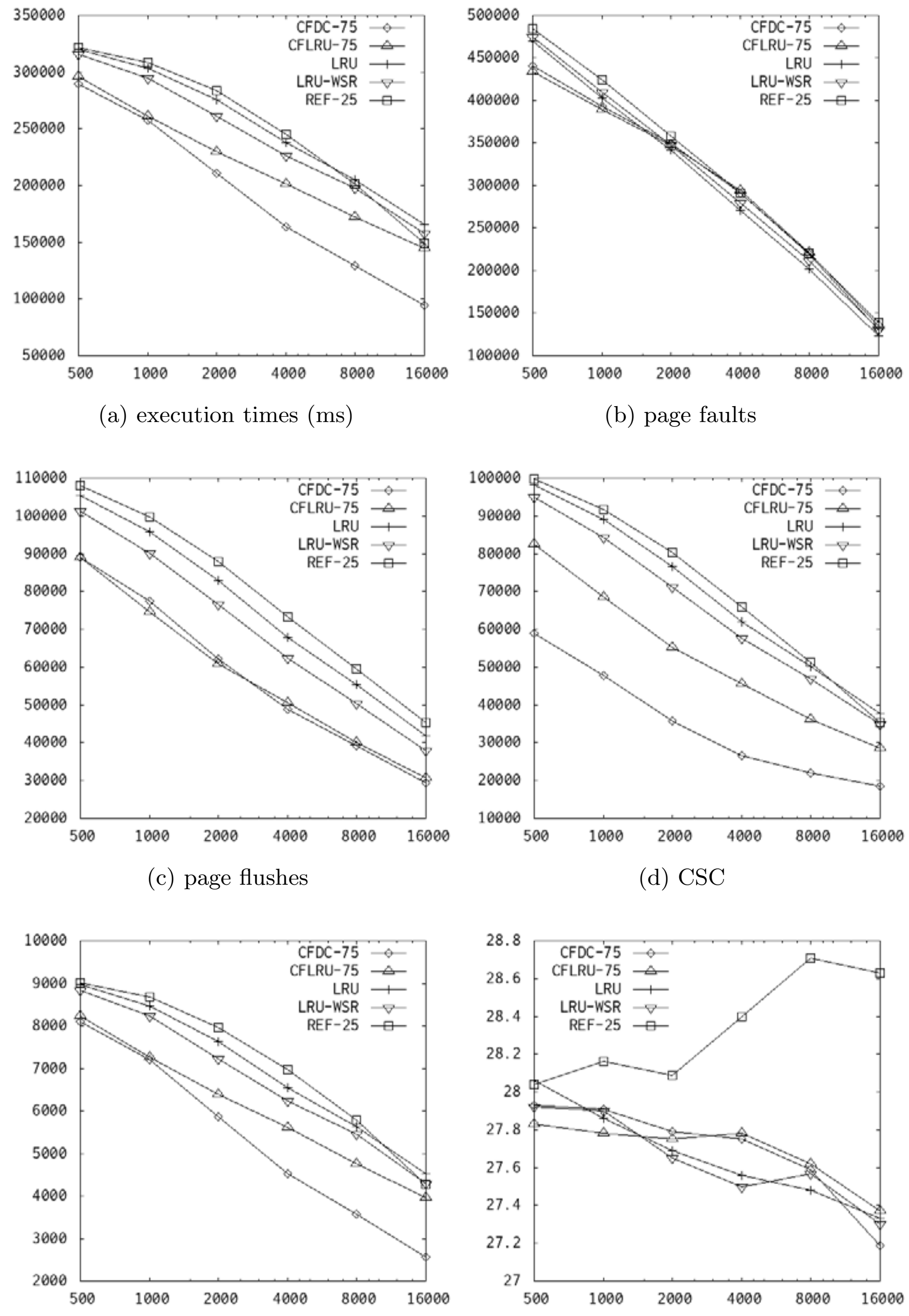


(a) execution times (ms) on HDD2

(b) execution times (ms) on SSD2

Fig. 5. Impact of window size

In the experiments discussed in Sect. 4.4, the parameter "window size" was *not* tuned—it was set to 0.5 for all related algorithms. To examine its impact, for each of the algorithms CFDC, CFLRU, and REF, we always ran the trace three times with the window size parameter set to 0.25, 0.50, and 0.75 respectively, denoted as REF-25, REF-50, REF-75, etc. As an indicative example, the execution times measured on the middle-class devices HDD2 and SSD2 are shown in Fig. 5. We found that the optimal window size depends on both the buffer size and the device characteristics, for all the three related algorithms.



(a) execution times (ms)

(b) page faults

(c) page flushes

(d) CSC

(e) energy (J)

(f) power (W)

Fig. 6. Performance and power running the bank trace on SSD2

Due to the space limitation, we only discuss the performance and power figures measured on SSD2 in more detail. For the same reason, we choose one best-performing window-size configuration for each of the related algorithms and compare them with LRU and LRU-WSR in Fig. 6. For CFDC and CFLRU it was 0.75, and for REF it was 0.25. The performance figures shown in Fig. 6a are explained by the device-independent metrics shown in Fig. 6b to Fig. 6d. For example, CFDC and CFLRU had comparable numbers of buffer faults and page flushes, but the latter suffered from a lower spatial locality of page flushes (Fig. 6d). The access skew is reflected in Fig. 6b: the difference in number of page faults becomes insignificant beyond 4000 pages.

The linear complexity of REF resulted in a higher CPU load compared with other algorithms having constant complexity. This is captured by Fig. 6f, where the working power of the system is illustrated. The power value of REF goes up with an increasing buffer size, while the power values caused by the other algorithms slightly decrease, because the larger the buffer sizes the more physical I/Os were saved. Note, handling a logical I/O from the buffer is more energy-efficient than doing a physical I/O, because fewer CPU cycles are required and device operations are not involved.

The algorithms' complexity did have an impact on the power of the system, but the time factor had a stronger impact on the overall system energy consumption. For example, enlarging the buffer from 500 to 16000 pages augmented the power value of REF by 2.1% (from 28.04 W to 28.63 W), while the corresponding execution time (Fig. 6a) decreased by 46.3%. In fact, the effect of the increased CPU load was hidden by the "idle share" of the working power (27.08 W, not shown in the figure). Therefore, the energy consumption curves in Fig. 6e largely mirror the performance curves of Fig. 6a. In particular, at a buffer size of 16000 pages, the relative performance gain of CFDC over CFLRU is 54%, while the corresponding energy saving is 55%.

5 Conclusions

Our experiments reveal a great performance potential of flash disks. The use of flash-aware buffer algorithms can further significantly improve system performance on these devices. The CFDC algorithm clearly outperformed the other algorithms in most settings. According to our device sensitivity study, its flash-specific optimizations do not exclude its application in systems based on magnetic disks.

The performance gain can be translated into a significant energy-saving potential due to the strong correlation between performance and system energy consumption. Furthermore, faster I/O reduces the overall system runtime and leaves more opportunities for the system to go into deeper energy-saving states, e.g., stand-by or even off-line.

As an important future task of hardware designers and device manufacturers, all system components other than flash disks should be developed towards stronger energy-proportional behavior. Then, the speed or runtime reduction gained by flash use for I/O-intensive applications could be directly translated into further substantial energy saving. As a consequence, energy efficiency due to flash disk use would be greatly enhanced as compared to magnetic disks.

References

1. Gray, J., Fitzgerald, B.: Flash disk opportunity for server applications. ACM Queue 6(4), 18–23 (2008)
2. Woodhouse, D.: JFFS: the journalling flash file system. In: The Ottawa Linux Symp. (2001)
3. Bouganim, L., et al.: uFLIP: Understanding flash IO patterns. In: CIDR (2009)
4. Härder, T., et al.: Towards flash disk use in databases—keeping performance while saving energy? In: BTW. LNI, vol. 144, pp. 167–186 (2009)
5. Effelsberg, W., Härder, T.: Principles of database buffer management. ACM TODS 9(4), 560–595 (1984)
6. Corbato, F.J.: A paging experiment with the multics system. In: In Honor of Philip M. Morse, p. 217. MIT Press, Cambridge (1969)
7. Tanenbaum, A.S.: Operating Systems, Design and Impl. Prentice-Hall, Englewood Cliffs (1987)
8. Park, S., et al.: CFLRU: a replacement algorithm for flash memory. In: CASES, pp. 234–241 (2006)
9. Jung, H., et al.: LRU-WSR: integration of LRU and writes sequence reordering for flash memory. Trans. on Cons. Electr. 54(3), 1215–1223 (2008)
10. Seo, D., Shin, D.: Recently-evicted-first buffer replacement policy for flash storage devices. Trans. on Cons. Electr. 54(3), 1228–1235 (2008)
11. Ou, Y., et al.: CFDC: a flash-aware replacement policy for database buffer management. In: DaMoN, Providence, RI, pp. 15–20. ACM Press, New York (2009)
12. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco (1993)
13. Härder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Computing Surveys 15(4), 287–317 (1983)
14. O'Neil, E.J., et al.: The LRU-K page replacement algorithm for database disk buffering. In: SIGMOD, pp. 297–306 (1993)
15. Megiddo, N., Modha, D.S.: ARC: A self-tuning, low overhead replacement cache. In: FAST, USENIX (2003)