Wear-Aware Algorithms for PCM-Based Database Buffer Pools

Yi Ou[1], Lei Chen[2], Jianliang Xu[2], and Theo Härder[1]

[1] University of Kaiserslautern
{ou,haerder}@cs.uni-kl.de
[2] Hong Kong Baptist University
{lchen,xujl}@comp.hkbu.edu.hk

Abstract. PCM can be used to overcome the capacity limit and energy issues of conventional DRAM-based main memory. This paper explores how the database buffer manager can deal with the write endurance problem, which is unique to PCM-based buffer pools and not considered by conventional buffer algorithms. We introduce a range of novel buffer algorithms addressing this problem, called wear-aware buffer algorithms, and study their behavior using trace-driven simulations.

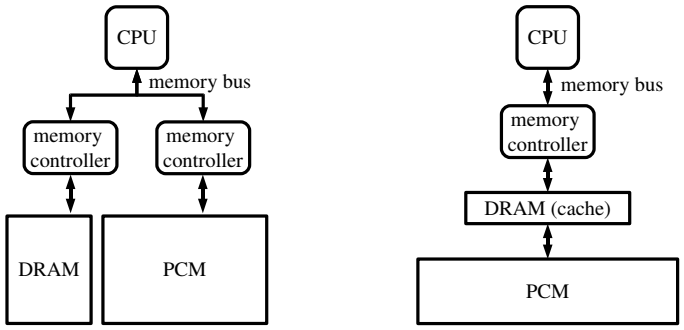
1 Introduction

Phase Change Memory (PCM) is a promising next-generation memory technology with a range of interesting properties: it is *non-volatile*, *bit alterable*, and *byte addressable*. Instead of being used as an external storage solution (like flash memory), PCM is more likely to be used in the main memory system, for two major reasons. First, similar to DRAM, bytes on PCM are *directly addressable* by the processor. Second, the *read latency* of PCM is close to that of DRAM. PCM has the potential to greatly impact core database technologies in the near future.

Similar to flash memory, PCM can endure only a limited number of writes (i. e., *limited write endurance*) and writes have a higher latency than read accesses (i. e., *read-write asymmetry)*. However, PCM can endure about 10^7–10^8 writes per cell [12] (even up to 10^{12} by projection [7]), whereas flash memory can only endure about 10^5–10^6 erase cycles per block [5]. Furthermore, PCM allows in-place update and does not have the erase-before-write constraint of flash memory, which further negatively and significantly impacts performance and lifespan of flash devices [11]. In terms of access latency, PCM is two to three orders of magnitude faster than flash memory.

Compared with DRAM, PCM offers *higher density* and, therefore, potentially much lower cost per gigabyte. PCM is also *more energy-efficient* than DRAM in idle mode[3]. However, PCM suffers from *two critical problems*: higher

[3] DRAM consumes, independent of its utilization, the lion's share of energy for a typical computer system and, with growing memory capacities, this situation gets even worse.



(a) Variant 1 **(b)** Variant 2

Fig. 1: Architectural variants of a hybrid main memory system based on DRAM and PCM, based on [2]

write latency and limited write endurance. Especially for data-intensive DBMS applications, it is therefore reasonable to consider a *hybrid main memory system* which consists of both PCM and (a relatively small amount of) DRAM, because such a design can overcome the capacity limit and energy issues of a conventional DRAM-based main memory without significant performance degradation.

1.1 Architectural consideration

There are two architectural variants of such a hybrid main memory system, shown in Figure 1. The first variant places PCM directly on the memory bus, side-by-side with DRAM [3]. The second variant places PCM below DRAM, as another layer in the memory hierarchy [12]. Their difference is that the first variant gives software explicit control over both types of memory (volatile and non-volatile), whereas the second variant manages DRAM as a hardware cache transparent to software developers [2].

The first architectural variant is more attractive for DBMS designers, because it allows the database software to take advantage of PCM's byte addressability and non-volatility. Our study follows such an architectural design and studies the management of a PCM-based database buffer pool, where the high-traffic, dynamic buffer management data structures, e. g., for indexing the buffer pool and for supporting efficient page replacement, are maintained in DRAM and the buffer pages are stored in PCM. By combining a relatively small amount of DRAM[4] with a large PCM buffer pool in such a way, the higher (write) latency can be partially hidden from the processor.

Although PCM's write endurance as compared to flash memory is improved by about two orders of magnitude, the write traffic to a buffer pool is expected to be significantly higher than to a secondary storage (e. g., based on flash memory).

[4] The DRAM can even accommodate a small number of hottest pages, depending on its available capacity.

Furthermore, page replacements also generate a substantial amount of write traffic to PCM. Therefore, special care has to be taken to prevent high-traffic writes (back from the processor's cache hierarchy) to the same PCM locations in the buffer pool. Minimizing such write traffic and distributing it uniformly across the PCM area may extend its lifetime long enough for practical DB use.

1.2 Goals

Conventional buffer algorithms are designed for DRAM-based buffer pools, which do not have the write endurance problem. Our study has three important goals:

1. Examine whether this problem can be effectively addressed by the database buffer manager.
2. Design buffer algorithms that address the endurance problem using wear-leveling techniques.
3. Study the behavior of such wear-aware buffer algorithms.

Our goals imply that, in addition to the hit ratio as the classical buffer pool metric, we shall be able to quantify (count or measure) the wear status of the PCM where the buffer pages are accommodated. Although PCM is managed by the buffer manager at a page level, other components of the DBMS can write to a page at a much smaller granule. Such writes can cause some parts of a page (more accurately, of the underlying PCM partition) to be worn out sooner than the other parts, unless the writes are uniformly distributed inside a page, which is a rare case. Therefore, studying the wear status at the page level, e. g., number of writes endured by each page, is not sufficient.

2 Wear-aware buffer management

In this work, the smallest unit for which the wear is quantified is referred to as the *wear unit*. A candidate for the wear unit size is the lowest-level cache line size (e. g., 64 B), because that is the data transfer unit between CPU and the memory controller. For a page size of 8 KB and a wear-unit size of 64 B, we would have 128 wear units per page.

To stay with the conventional terminology when discussing DB buffer management, we denote the buffer replacement units as pages. For brevity, the term *page* either refers to a logical page or to a buffer page. A buffer page is a PCM partition having the size of a logical page, i. e., a buffer pool of B pages consists of B such partitions. Where it is unclear from context, we use the explicit terms *logical page* and *buffer page* to avoid ambiguity.

Considering the distinguished properties of PCM, some hardware optimizations have already been proposed, e. g., data comparison writes [14] or partial writes [7], to reduce the number of bits written to PCM. These hardware optimizations would be integrated into the conventional memory controller or even implemented in a dedicated memory controller for PCM. In both cases, the

unit of data transfer between memory controller and PCM can be as small as a word (4 B) [7]. Due to these hardware optimizations, software optimizations at a granule smaller than the cache line size, e. g., those of [2], can also significantly improve wear leveling and performance. Therefore, we choose to examine the wear of PCM at the word level, i. e., the wear unit size is 4 B in our study. The wear unit size is a parameter for some of the algorithms that we will discuss. However, the logic of those algorithms are not specific to the parameter value.

2.1 Problem analysis

For a wear unit u, we denote its *wear count* as $w(u)$, which is the number of times u was written. A wear unit u is *worn out* (i. e., it *fails*), if $w(u) = L$, where L is the *(wear) endurance limit*. Furthermore, we define *page wear* as the total wear count of all wear units of a buffer page and *total wear* as the total page wear of all buffer pages.

Assuming the wear count follows the normal distribution, we have $w(u) = \mu + \sigma \cdot Z$, where μ is the expected value of $w(u)$, σ the standard deviation, and Z the standard normal random variable. After a large number of writes to the PCM, the probability that u is not worn out is:

$$P[w(u) < L] = P[\frac{L - \mu}{\sigma} > Z] \tag{1}$$

If a PCM device with M wear units is considered worn out (i. e., the device fails) as long as one of its wear units fails, the probability of no device failure is (assuming that the wear units fail independently):

$$P[\text{no device failure}] = \left(P[\frac{L - \mu}{\sigma} > Z] \right)^M \tag{2}$$

There are two cases where the wear count increases. In both cases, we say that the corresponding buffer page is *modified*:

Page replacement When a buffer fault occurs, the buffer manager has to evict a victim page from the buffer pool to make room for the missing logical page, which will then be written to the buffer page where the victim resided. This means the wear count for all the wear units of that buffer page will be *potentially* increased by one.

Page update Update operations on the buffer page (insert, delete, and update of its records). Updating a record will only increase the wear count for the related wear units (instead of for the entire buffer page).

These two cases correspond to the two sources of write skew to the PCM used for a buffer pool: *inter-page* skew and *intra-page* skew. Although a buffer algorithm can only directly influence inter-page skew through its page replacement decisions, its page replacement strategy shall consider the page wear status for an effective wear leveling. Note, the page wear status is co-influenced by page replacement and page update.

2.2 Replacement strategies

Formula (2) implies that to extend the device lifespan, two options are possible. The first one is to reduce the number of writes to PCM. This impacts μ. This option is taken by the LWD algorithm introduced in the following. The second one, taken by the remaining algorithms to be introduced, is to distribute the writes uniformly across the wear units. This reduces σ.

LWD algorithm The LWD (least wear-unit difference) algorithm is our first attempt to address the write endurance problem. On a buffer fault, the algorithm compares the page p waiting to enter the buffer pool with each page q in the buffer pool and chooses the one that has the smallest *wear-unit difference (WD)* to p as the victim. If multiple pages have the smallest WD to p, the least recently used (LRU) one among them is selected as the victim.

The *wear-unit difference* of a page q to page p, denoted as $d(p, q)$, is computed by comparing each corresponding pair of wear units (i. e., the two wear units at the same offset in both pages) and count the number of pairs that differ. In other words, $d(p, q)$ describes the physical similarity between p and q at the granule of a wear unit.

LWD assumes that to replace q by p in the buffer pool, instead of writing an entire page to the PCM, only the wear units that differ in both pages need to be written. This kind of optimization can be easily implemented in the buffer manager and it is also offered by the afore-mentioned hardware approach (data comparison write [14]). Therefore, LWD can potentially reduce the *total wear*.

However, LWD has a few problems. First, although the algorithm considers the WD, it does not consider the current *wear status*, e. g., how often is a buffer page already written or how are the writes distributed on its wear units. Consequently, a nearly worn-out page can still be selected as the victim (and thus to be written again) as long as it is the most similar one compared with the page entering the buffer pool. Therefore, it could not effectively achieve wear leveling. Second, the probability that two pages are highly similar is very small. Considering a wear unit size of 4 B, the probability that two wear units are identical is $1/2^{32}$, if the binary value of a wear unit follows a uniform distribution. Therefore, it can not substantially reduce the total wear.

LPW algorithm The LPW (least page wear) algorithm addresses the first problem of LWD. Instead of considering the physical similarity, LPW always selects the page having the smallest *page wear* as victim. Page wear can only *approximately* represent the wear status of a page, because it does not capture the distribution of the wear count inside the page. This means, even if a few wear units of a page have high wear counts (e. g., due to skewed updates), the page can still can have a low page wear relative to other pages and, therefore, be (repeatedly) selected as victim. However, this is a trade-off for simplicity, because the buffer manager only has to maintain a counter for each buffer page. For a page replacement, the counter is incremented by the number of wear units

the page contains. For a page update, the counter is incremented by the number of wear units that are updated.

Heuristic algorithms The LFM (least frequently modified) algorithm further simplifies the LPW approach by using page modification (i. e., page replacement or page update) *frequency* as the heuristic for the page wear status. As the name suggests, LFM selects the least frequently modified page as the victim, similar to the classical LFU (least frequently used) algorithm. However, LFU typically maintains, for each logical page currently in the buffer pool, an access frequency counter, which is reset at each page replacement. In contrast, the modification-frequency counter in LFM is maintained for each buffer page and the value of the counter survives page replacements.

Similarly, the LRM (least recently modified) algorithm resembles the classical LRU (least recently used) algorithm, but uses the modification *recency* as victim selection criterion. Similar to LRU, LRM can be implemented using a linked list of buffer page pointers. Page update or page replacement moves the modified page pointer to the MRM (most recently modified) position. On page hits, in contrast to LRU, LRM does not change the page (pointer) position in the list.

2.3 Complexity

The LWD algorithm has the highest time complexity, which is $O(B \cdot P)$ in our current implementation, where B is the buffer pool size and P the page size. Both LPW and LFM have a complexity of $O(B)$. The LRM algorithm has a complexity of $O(1)$.

In terms of space overhead, LRM is similar to LRU: the only overhead is introduced by the linked list structure. The LWD algorithm does not introduce any space overhead, because it compares the content of the wear units on the fly. Both LPW and LFM maintain a counter (of 8 B) for each buffer page. But this space overhead is ignorable, e. g., it is 0.1% for a page of 8 KB. Note, all these data structures and management information are maintained in DRAM, based on the architectural assumption given in Section 1.1. If, for practical considerations, these small amount of meta data need to survive a DB server restart or crash, it is sufficient to propagate them to a persistent storage in appropriate intervals, e. g., hours or even days are acceptable.

3 Experiments

We used trace-driven simulations to evaluate the afore-mentioned algorithms. As explained in Section 1.2, we are primarily interested in two aspects of the algorithms: wear-leveling effectiveness and hit ratio. To study the wear-leveling effectiveness, we have to use record-oriented traces, instead of the page-oriented workloads (i. e., page reference strings) typically used by the studies on conventional buffer management [4].

3.1 Workload

Most experiments reported in this section used a synthetic trace which simulates a typical workload for a database buffer pool. The trace is generated as follows. Each tuple of the trace is a record request, which consists of a page identifier $\in [0, 131072)$, a record identifier $\in [0, 64)$, and a type identifier $\in \{\mathrm{R}, \mathrm{U}\}$. The type identifier describes the operation to be performed on the record: read (R) or update (U). For a U record request, the tuple additionally contains a randomly generated record of 128 B. A trace contains one million such record requests. The workload follows the 80–20 rule: both the page identifier and the record identifier follow a 80–20 self-similar distribution within their respective ranges to simulate skewed accesses (particularly skewed updates). Moreover, the update ratio is 20%, i. e., 80% of the requests are R requests and the remaining 20% are U requests. Our experiments used a typical database page size of 8 KB. The page layout follows a simplified N-ary storage model (NSM) [1]: each page consists of N equi-length records ($N = 64$).

3.2 Methodology

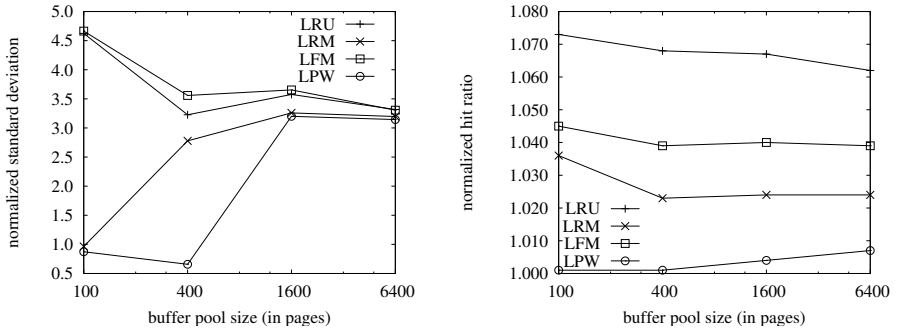
We implemented a database storage manager supporting all the algorithms under examination. The traces are processed by the storage manager as follows. Prior to each trace execution, the database file (1 GB) is initialized with 131072 pages and each page contains 64 randomly generated records. An R record request is processed by getting the corresponding page via the buffer pool and reading the record. A U record request additionally overwrites the corresponding record in place using the generated modification pattern (sequence of bytes) contained in the request.

In addition, we implemented a PCM simulator, which maintains a wear counter for each wear unit of the buffer pool. The wear counters are all reset prior to each experiment. For each buffer page modification during the trace execution, the simulator increments the wear counters of the affected wear units. The wear-leveling effectiveness of the algorithms is expressed by the *standard deviation* σ of the wear count after each trace execution. The *hit ratio* h achieved by the algorithms are also compared. For a simulated PCM of n wear units $\{u_i$, for $0 <= i < n\}$, the wear count standard deviation σ is computed as:

$$\sigma = \sqrt{\frac{\sum_{i=0}^{n-1} (w(u_i) - \overline{w})^2}{n-1}}$$

where $\overline{w}$ is the mean of $w(u_i)$.

We included LPW, LFM, and LRM in the comparison. Additionally, LRU and RND (random) are used as baselines for hit ratio (LRU) and wear-leveling effectiveness (RND). We did not include LWD in the comparison due to its problems discussed in Section 2.2, which are confirmed by our experiments (omitted due to space limitations). To facilitate visual comparison, we present the *normalized* performance figures, i. e., their ratios *relative* to the corresponding figures of RND.



(a) Normalized σ **(b)** Normalized h

Fig. 2: Normalized σ and h, buffer pool size scaled from 100 to 6400 pages, typical workload

3.3 Typical workload

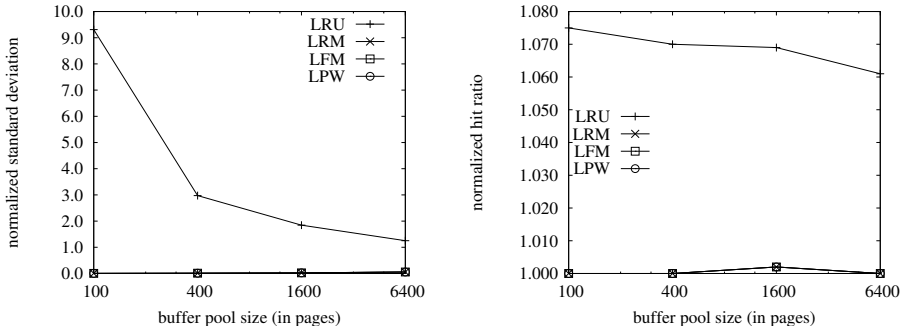
Figure 2 compares the σ (wear count standard deviation) and h (hit ratio) of our algorithms (relative to RND) for the traces described above. Figure 2a confirms that buffer algorithms can have a great impact on wear leveling. For example, for a buffer pool of 100 pages, LRM improves the σ by nearly a factor of five compared with LRU.

To our surprise, the wear-leveling effectiveness of LFM is even worse than LRU for all buffer pool sizes. Our explanation is that the page modification frequency used by LFM is not a good approximation for the page wear status, because it can not distinguish between page replacement and page update. In contrast, this difference is captured by the page wear metric used by LPW, which, as a consequence, delivered the best σ for all buffer pool sizes. For a buffer pool of 400 pages, LPW reduced the σ by 24.2% even compared with RND. The performance of LPW reveals that, although a buffer algorithm can primarily influence only the inter-page skew, considering the intra-page skew in the victim selection can help to improve wear-leveling effectiveness.

The hit ratios of LFM, LRM, and LPW are lower than that of LRU. LPW is even close to RND in terms of h. However, this is expected, because page reference statistics are not considered by LPW at all, while LFM (LRM) only partially considers the frequency (recency) of page updates in their victim selection decision. This implies that, if the workload is read-only, i. e., there is only page replacement and no page update, LFM and LRM would have no advantage over LPW in terms of h.

3.4 Read-only workload

This is confirmed by our experiment using a read-only trace, which shares all the parameters with the afore-mentioned trace except for the update ratio, i. e., the



(a) Normalized σ **(b)** Normalized h

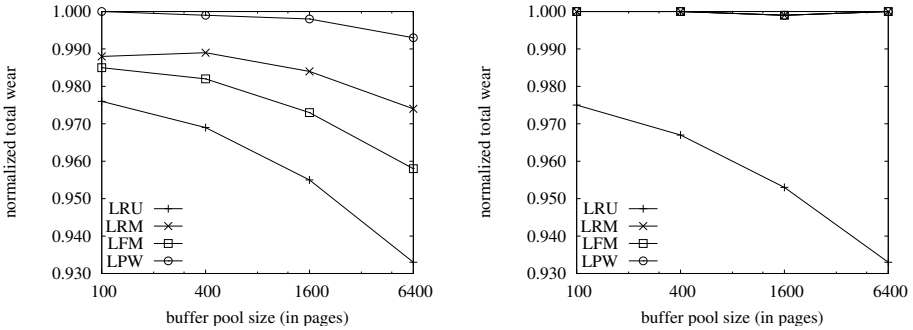
Fig. 3: Normalized σ and h, buffer pool size scaled from 100 to 6400 pages, read-only workload

read-only trace contains only R requests, which only trigger page replacements. The results of this experiment are shown in Figure 3.

Under the read-only workload, all algorithms compared—LRM, LFM, and LPW—achieved nearly a perfect wear leveling (Figure 3a), because there is no intra-page write skew (i. e., no record-level writes), but only a (inter-page) reference skew. In other words, the write is uniform within a page, but non-uniform among pages. The latter, inter-page write skew, can be effectively handled by our algorithms. This implies that if the write skew inside a page is less significant than the page reference skew, our algorithms are more effective in wear leveling. After all, the buffer algorithms have no direct influence on the update locations inside a page. This further implies that our algorithms would be even more effective in terms of wear leveling for a smaller page size. The price to pay for a nearly perfect wear leveling is a very low hit ratio which is close to that of RND, as shown in Figure 3b.

3.5 Total wear

Figure 4 reports the total wear for the experiments corresponding to Figure 2 and Figure 3. The curves in Figure 4b look (reversely) similar to the h curves in Figure 3b. The same relation exists between Figure 4a and Figure 2b. This results from the fact that the total wear is dominated by the number of page faults. Each page fault requires a page replacement, which has a much greater impact on the total wear than a page update: the former overwrites an entire buffer page, whereas the latter only updates a record. Therefore, the total wear has a strong correlation with h.



(a) Normalized total wear for the typical workload

(b) Normalized total wear for the read-only workload

Fig. 4: Normalized total wear, buffer pool size scaled from 100 to 6400 pages

4 Related work

Qureshi et al. analyzed the write traffic to pages for two database applications and found that for both applications there is significant non-uniformity in which lines (i. e., page partitions of cache line size) in the page are written back [13]. Their analysis confirmed the significance of the intra-page skew. As solution, the authors proposed a fine-grained wear-leveling technique, which randomly shuffles the lines when writing a page to PCM and restores the page layout when the page is written back to disk. Our approach to wear leveling is complementary to theirs, because our software approach manipulates the write traffic at a page level, whereas their hardware approach does this at the line level.

There have been some pioneer works on the use of PCM in database systems. For example, Chen et al. advocate that database algorithms should be adapted to PCM technology to improve performance, energy efficiency, and PCM's write endurance. For this purpose, they presented PCM-friendly algorithms for two core database techniques: B^+-tree index and hash joins [2]. Gao et al. presented a novel logging scheme that exploits the non-volatility and bit alterability of PCM for efficient transaction logging in disk-based databases [6]. However, the issues of using PCM for database buffer pool, i. e., the focus of our work, are not covered by these works.

Ou et al. identified the cold-page migration (CPM) problem related to the indirect use of flash memory for mid-tier buffer pool and proposed two effective solutions [11]. Our study has one thing in common with theirs: one of our goals is to address the write endurance problem and extend the device lifespan. However, the CPM problem studied by them is specific to flash memory, because it is rooted in the flash memory erase-before-write limitation, which is not present for PCM. Furthermore, wear-leveling techniques are not the focus of their study.

5 Conclusion and future work

In this work, we studied the write endurance problem of PCM-based database buffer pools. To attack this problem, we identified and classified the sources of write skew in such an environment and introduced a range of novel buffer algorithms, which examine page content or consider page wear status for their replacement decisions, to minimize and uniformly distribute the write traffic.

Using these algorithms, the database buffer manager can effectively address the endurance problem, as confirmed by our experiments. The experiments reveal that, although a buffer algorithm can primarily influence only the inter-page skew, precise page-wear status information can be used to improve its wear-leveling effectiveness.

Among the algorithms compared, LRM and LPW can effectively achieve wear leveling (e. g., up to factor five in one of the experiments or even a nearly perfect wear leveling for read-only workloads). However, they had a lower hit ratio compared to the conventional buffer algorithms represented by LRU. This suggests that wear leveling and hit ratio are two conflicting goals that must be considered by the buffer algorithm in a well-balanced fashion. Nevertheless, for a workload with lower page-level locality, e. g., 70–30 (or even 60–40) instead of 80–20, their penalty in terms of hit ratio would be smaller. For a hybrid main memory system, where the hottest pages are retained in the smaller DRAM, we can assume that the locality of page reference to the PCM would be lower than the 80–20 one.

Under the read-only workload, the wear-aware algorithms achieved the best wear-leveling effect but the lowest hit ratios. This is an issue requiring further algorithmic improvements. A possible approach is to dynamically adjust the behavior of the buffer manager based on workload statistics, similar to the approaches of [8–10].

6 Acknowledgement

Yi Ou's work is partially supported by the German Research Foundation and the Carl Zeiss Foundation. Jianliang Xu's work is partially supported by Research Grants Council (RGC) of Hong Kong under the grant no. G_HK018/11. The authors are grateful to German Academic Exchange Service (DAAD) for supporting their cooperation. They are also grateful to anonymous referees for valuable comments.

References

1. Ailamaki, A., DeWitt, D.J., Hill, M.D.: Data page layouts for relational databases on deep memory hierarchies. The VLDB Journal 11, 198–215 (2002), http://dx.doi.org/10.1007/s00778-002-0074-9
2. Chen, S., Gibbons, P.B., Nath, S.: Rethinking database algorithms for phase change memory. In: CIDR. pp. 21–31. www.cidrdb.org (2011)

3. Condit, J., Nightingale, E.B., Frost, C., Ipek, E., Lee, B.C., Burger, D., Coetzee, D.: Better I/O through byte-addressable, persistent memory. In: SOSP. pp. 133–146 (2009)
4. Effelsberg, W., Härder, T.: Principles of database buffer management. ACM TODS 9(4), 560–595 (12 1984)
5. Gal, E., Toledo, S.: Algorithms and data structures for flash memories. ACM Computing Surveys 37(2), 138–163 (2005)
6. Gao, S., Xu, J., He, B., Choi, B., Hu, H.: PCMLogging: reducing transaction logging overhead with PCM. In: CIKM. pp. 2401–2404 (2011)
7. Lee, B.C., Ipek, E., Mutlu, O., Burger, D.: Architecting phase change memory as a scalable DRAM alternative. In: ISCA. pp. 2–13 (2009)
8. Megiddo, N., Modha, D.S.: ARC: a self-tuning, low overhead replacement cache. In: USENIX FAST'03. USENIX (2003)
9. Ou, Y., Härder, T.: Clean first or dirty first? a cost-aware self-adaptive buffer replacement policy. In: IDEAS'10. Montreal, QC, Canada (2010)
10. Ou, Y., Jin, P., Härder, T.: Flash-aware buffer management for database systems. Int. Journal of Knowledge-Based Organizations 3(4), 22–39 (2014)
11. Ou, Y., Xu, J., Härder, T.: Towards an efficient flash-based mid-tier cache. In: DEXA. pp. 55–70 (2012)
12. Qureshi, M.K., Karidis, J., Franceschini, M., Srinivasan, V., Lastras, L., Abali, B.: Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In: MICRO. pp. 14–23. IEEE (2009)
13. Qureshi, M.K., Srinivasan, V., Rivers, J.A.: Scalable high performance main memory system using phase-change memory technology. In: ISCA. pp. 24–33 (2009)
14. Yang, B.D., Lee, J.E., Kim, J.S., Cho, J., Lee, S.Y., Yu, B.G.: A low power phase-change random access memory using a data-comparison write scheme. In: ISCAS. pp. 3014–3017 (2007)