SCM-BP: An Intelligent Buffer Management Mechanism for Database in Storage Class Memory

Júlio A. Tavares, José de Aguiar Moraes Filho, Angelo Brayner, Eduardo Lustosa

University of Fortaleza - UNIFOR, Brazil
{julio,jaguiar,brayner,eduardolustosa}@unifor.br

Abstract. A set of new storage media, called Storage Class Memory (SCM), has emerged as a quite promising solution to decrease the difference between HDD data access time and the time that processors can consume data. Four main characteristics may be highlighted in SCM: *(i)* non-volatility; *(ii)* low access time; *(iii)* high rates of IOPS, and *(iv)* read/write execution time asymmetry. The former three have a direct benefit for database systems. Notwithstanding, the latter one poses challenges for database systems. In fact, read-write speed ratio in SCMs can reach an 1-to-300 factor. For databases stored in HDDs, disk access is a critical factor for database performance. The buffer manager is in charge of reducing the amount of disk access, maintaining the most used database pages in main-memory buffer. Most buffer-replacement policies have been proposed aiming at only avoiding disk access. Nonetheless, flushing too many changed pages from buffer to disk may reduce IOPS rates of SCMs. In this article, we propose the SCM-BP, an intelligent buffer replacement policy targeted to SCM devices. SCM-BP is able to autonomously adapt its behaviour and to choose the most efficient moment to do this. We provide an empirical study comparing SCM-BP to the widely used LRU policy. The results show the positive impact of our proposal to database system performance w.r.t buffer management when SCM media is used.

Categories and Subject Descriptors: H.2 [**Database Management**]: Miscellaneous; H.3 [**Information Storage and Retrieval**]: Miscellaneous

Keywords: buffer management, storage class memory, flash memory, performance

1. INTRODUCTION

It is well know by now that SCM solutions outperform hard-disk drives (HDDs). While the speed of processors has raised exponentially, the number of inputs/outputs per second (IOPS) afforded by hard-disk drivers (HDDs) has only increased marginally. In other words, there is a significant difference between HDD response time and the time that processors can consume data. Another critical drawback presented by HDDs and overcome by SCMs are the high rates of energy consumption.

SCM, a term coined at the IBM research laboratories, encompasses a set of promising storage technologies to improve data access speed in non-volatile storage drives and energy efficiency. For example, NOR and NAND flash memories and phase-changed memories (PCM) are considered to be SCM. As SCM products, we may cite pen drives, memory cards used in ultrabooks and digital cameras, internal memory of sensor units and solid state drives (SSD).

It is a fact that SCM drives currently have a storage capacity inferior to their HDD counterparts. While one has petabytes in HDDs, there are only hundreds of gigabytes in SCM. However, in the current pace of SCM technology development, we can expect that in a couple of years SCM storage capacity will be similar to that provided by HDDs. Additionally, there exists the alternative of exploiting the use of storage array of SCM drives. Therefore, storing databases in SCMs is also a fact in modern enterprises.

Work partially supported by FUNCAP grant PRN-0040-00037.01.00/10.

SCM memories share features from which the most evident one is the absence of mechanical parts in their assembly, only semi-conductors (chips) are used in. Due to such a feature, SCM presents the following characteristics:

(1) Low rates of energy footprint. This is because, to perform read/write operations, there are only logical gates (circuitry) involved;
(2) Low random access time. SCM provides random access time up to 1000 times faster than HDDs;
(3) High random IOPS rates. Since SCMs have no mechanical moving parts, there is no mechanical seek time or latency to overcome. In fact, SCMs provide IOPS over 100 times faster than 15K RPM HDDs;
(4) High reliability rates. Enterprise class SCMs may present 2 million hours MTBF (mean time between failure) versus most 15K RPM HDDs, which present up to 1.6 million hours MTBF;
(5) Asymmetry of read and write operation execution time. As a consequence of the technology used in SCMs, a read operation is up to 300 times faster than a write operation.

From a database technology perspective, characteristics 1 to 4 are directly beneficial to existing database systems. On the other hand, read/write asymmetry may pose challenges to database technology. Several database components may be impacted, e.g., buffer manager, query processor and recovery components. [Harizopoulos et al. 2008] has shown that the buffer manager is responsible for more than 1/3 of the number of instructions executed for a DBMS and for almost 30% of CPU cycles used by a DBMS. Therefore, buffer management is critical for database performance and it is very sensitive to the underlying storage media.

The key goal of the buffer manager is to reduce the amount of disk access. For that, the buffer manager has to maintain the most used database pages in main-memory buffer. Thus, whenever the database system requests access to a page P (for example, to process a query), the buffer manager checks whether or not P is already in the buffer pool. If P is in the buffer pool, it is made available without a disk access. In case P is not in the buffer, the buffer manager requests the disk access and loads P into the buffer (swap-in operation). Clearly, the number of pages in the buffer pool is much smaller than the number of pages in a database. Consequently, during a swap-in operation the buffer pool is already full of pages. Whenever this occurs, the buffer manager has to select a given page P' to be removed from the buffer. If P' has not been updated, then it may be erased from the main memory. However, if P' has been changed, it has to be written back to its physical address in disk (swap-out operation). The metric for measuring buffer manager efficiency is the hit ratio, which represents the probability of finding a requested page in the buffer pool (see Figure 1 for a schematic view of a buffer manager).

The most critical operation executed by the buffer manager is to select a page to leave the buffer when a new page has to be loaded into the buffer pool and the maximal capacity of the buffer pool has been achieved. For that reason, the buffer manager implements a buffer-replacement strategy (e.g., LRU, MRU and FIFO). Most buffer-replacement strategy have been proposed to avoid disk access. Thus, such policies may throw out (from the buffer) pages, which have only been read or written pages. However, removing written pages implies write operations on the non-volatile media. On the other hand, access to the media usually is not a bottleneck for solid-state memory technology, but write operations may jeopardize the benefits of using SCMs.

Therefore, we propose a new buffer management policy called SCM-BP, which explores SCM asymmetry to yield a better database performance. SCM-BP tries to keep in database buffer (main memory) write-intensive pages, postponing the moment to write them back to the SCM media. By doing so, SCM-BP reduces the number of write operations into SCM media, which increases SCM performance and lifetime. In order to achieve its goal, SCM-BP maintains two regions in the buffer: read and write region. The area allocated to each region is autonomously and automatically adjusted by SCM-BP. We have evaluated SCM-BP empirically against the well-known LRU policy using different workloads.

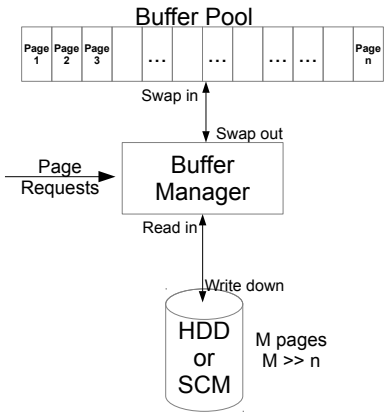


Fig. 1: Sketch of a Buffer Manager.

This article is structured as follow. Section 2 describes SCM memories technological characteristics. In section 3 some related works are studied. Section 4 details the SCM-BP, its features and algorithms. In Section 5, we show the empirical evaluation results. Section 6 concludes this article.

2. SCM CHARACTERISTICS

In this section, we describe and analyse the characteristics of the SCM media. There are two main types of SCM: Flash memory and PCM (phase change memory) memory.

PCM memory exploits the unique behaviour of the chalcogenide glass in which the heat produced by the passage of an electric current switches this material between two states: crystalline (with low resistance state, cool and representing bit 1) and amorphous (with high resistance state, hot and representing bit 0). The chalcogenide is the same material used in CD-RW and DVD-RW. Although PCM has not yet reached the commercialization stage for consumer electronic devices, it has a quite good potential to overcome the drawbacks of Flash memory. As recent news, we may cite the two most auspicious late announces for PCM development. In April 2010[1], Numonyx announced the Omneo line of 128-Mbit NOR-compatible phase-change memories and Samsung announced shipment of its 512 Mb phase-change RAM (PRAM) in a multi-chip package (MCP) for use in mobile handsets by Fall 2010. In June 2011[2], IBM announced the development of a stable, reliable, multi-bit phase change memory device with high performance. Flash memories, in turn, are already widely used as computer components (e.g., pen drive, flash-based SSD drive, enterprise SSD). Therefore, we focus our discussion on flash-based memories.

Flash memory is a computer chip which can be electrically reprogrammed and erased. Flash memory stores data in an array of floating-gate transistors, called cells. Bits are represented by means of the voltage level in a cell. A cell with high voltage level (typically greater than 5v) represents a bit 1 (default state), whereas a low voltage level represents a bit 0. Depending on how many bits can be stored in a cell, we may have two different types of flash devices: single-level cell (SLC) or multilevel cell (MLC). In SLC only one bit is stored in a cell. In turn, MLC flash devices are able to store more than one bit per cell. For instance, there are MLC devices which store 2 bits/cell by using four different voltage levels: low (00), medium low (01), medium high (10) and high (11).

A flash device is composed of several planes, each of which has a set of blocks. In turn, each block is divided into pages. Figure 2a depicts an abstract model of a flash device.

[1]At http://eetimes.com/electronics-news/4088727/Samsung-to-ship-MCP-with-phase-change
[2]At http://www.engadget.com/2011/06/30/embargo-ibm-develops-instantaneous-memory-100x-faster-than-fl/

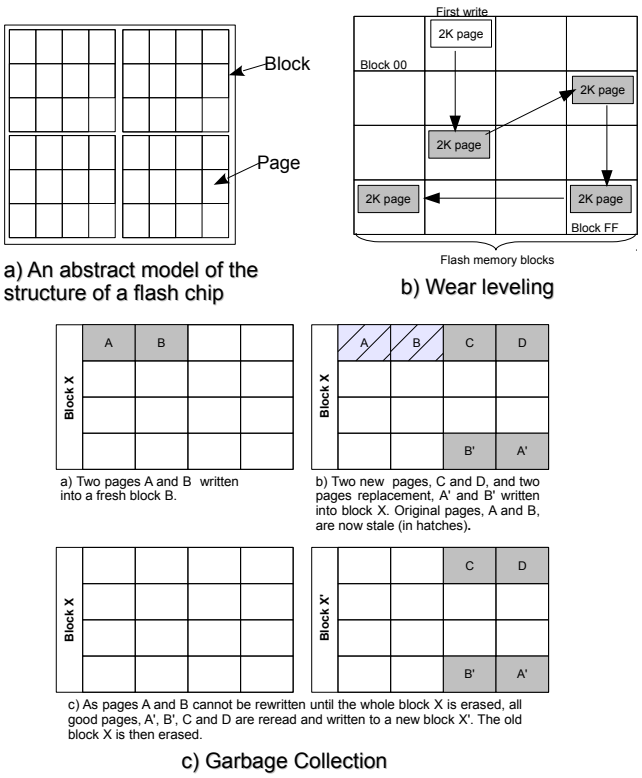


Fig. 2: SCM characteristics.

There are three operations which should be executed on a flash device: read, erase and program [Kim and Koh 2004]. A *read* operation may randomly occur anywhere in a flash device. An *erase* operation is applied to a given block of a flash device and sets all bits to 1. A *program* operation sets a bit to 0. It is important to note that a program operation can only be executed on a "clean" (free) block, which is a block with all bits set to 1.

Depending on the way of cells are interconnected, flash devices can be categorized in NAND flash and NOR flash. A NOR flash device takes its name because the internal circuitry arrangement (parallel interconnection of cells) resembles a logical NOR gate. The programming (write) unit of a NOR flash memory is a byte, whereas the erasing unit is a block. In other words, in a NOR flash bit values can be set to 0 (program operation) in byte-to-byte fashion (byte-addressable). Block sizes are typically, 64, 128 or 256 Kbytes. NOR flash devices are assembled on SLC cells. For that reason, its density is low and it is less impacted by voltage noise, making it more robust to cell failures and having low read latency.

On the other hand, NAND flash name derives from the internal arrangement of transistors resembling a NAND logical gate (i. e., several transistors interconnected in series). A NAND flash chip is page-addressable for executing read and program operations. Pages are typically of 512, 1024 or 2048 bytes. Erasing unit in NAND flash memory remains a block (set of pages). Since a program operation can only be applied to clean blocks, to change the values of a set of bytes in a block B of a NAND flash device, the entire block should be first erased (i.e., setting all bits to 1). Thereafter, it is possible to execute the program operation on B for changing the values of the set of bytes. Thus, with a clean block B, any location in B can be programmed, but once a bit in B is set to 0, to change this bit to

1, erasing the entire block is mandatory. In other words, flash memory offers random-access read and program operations, but does not offer arbitrary random-access rewrite or erase operations.

Although NAND flash devices share many characteristics with NOR flash, there exist significant differences between them. First of all, write endurance, i.e., the number of write cycles beyond which the memory deteriorates its integrity, varies a lot between NAND and NOR flash. SLC NOR flash presents typically a write endurance greater than that of MLC NOR and NAND flash. MLC NAND and NOR flash have similar write endurance. For example, a SLC NAND flash is normally rated at about 100k cycles; a SLC NOR, 100k to 1M cycles; a MLC NAND flash, 1k to 10k cycles; and a MLC NOR flash is typically rated at 100k cycles. Consequently, NAND and NOR flash memories have a finite number of write cycles. In other words, flash memory devices (NAND and NOR) have their lifetime determined by the number of write operations. Another important difference between NOR and NAND memories is that NOR flash devices were developed targeting a more economical rewritable ROM memory. NAND flash devices, in turn, were designed to reduce the cost per bit and to increase the maximum chip capacity trying to compete with hard disk drives.

NAND flash devices present higher storage capacity than NOR flash. This is because NAND flash memory uses multilevel cell (MLC), thus more than one bit can be stored in a cell. A side-effect of using MLC is the increasing of bit error rate (BER) [Mielke et al. 2008]. In order to reduce BER level, NAND flash devices need to implement error correction code (ECC) and bad block management.

In NAND flash memory, there is a controller chip, which is responsible for a logical-to-physical mapping. This way, NAND flash memories can be accessed like HDDs, i.e., in a block-wise fashion. For instance, when a high-level application (e.g., a file system) requires an access to a logical block, the controller chip makes a map to the correspondent physical block in NAND memory. The chip maintains two map tables (direct map and inverse map) to perform such mapping and to mark (physical) bad blocks. These mapping tables (also called logical block addressing – LBA) can be stored internally in the controller or in main memory.

Since flash memory devices have their lifetime determined by the number of write operations, a technique called *wear levelling* is applied to prolong flash memory useful life. Wear levelling is also performed by the controller chip in flash device. The key goal is to evenly spread write operations out across the storage area of the medium. Thus, once a program write operation is executed on a page P of a physical block B, the next write operation in P requires that the entire block B be erased. In order to avoid the erase operation to be executed in-place, the wear leveling procedure searches for a fresh block[3] B' and writes (moves) $B's$ data into B' and marks B as clean (free) block. If there is no new fresh block, a clean block is searched for and used to write $B's$ data, again B set as clean, Figure 2b illustrates the wear levelling technique.

The *garbage collector* process is triggered when a free block is necessary to be used for a replacement operation in a page P of a block X and there are no fresh blocks any more in the storage medium. We can only rewrite pages in a block if the whole block is erased. Thus, in a certain moment of time, in a block, there may be fresh pages (all bytes set to 1), with no write operations yet; needed pages (called good pages), from which data can be read, and stale (also called invalid) pages, pages written before but no more possible to be rewritten. When there is a need to rewrite stale pages in a block, all good pages are reread and written to a free block X' (i.e., a clean block previously erased), and pages are moved to it. The old block X is erased (see Figure 2c).

A phenomenon, called *write amplification* occurs as a consequence of wear levelling and garbage collection. Wear levelling and garbage collection, in fact, physically moves user data more than once across the medium. This increases the number of writes/erasures over the medium lifetime, shortening the time it can be considered reliable to operate.

[3]A block is fresh if no write operation has been executed on it.

3. RELATED WORK

In this section, the most relevant buffer-replacement policies are described and analysed. For the sake of clarity, the policies are classified into disk-oriented and SCM-oriented policies. The former group contains the policies whose may goal is to reduce the amount of disk access. In turn, the latter group presents policies which are SCM-aware. Before describing the buffer-replacement policies, we present some concepts. A dirty page represents a page in the buffer pool, which has been modified. Its counterpart is a clean page. A highly referenced page is called a hot page.

3.1 Disk-oriented Policies

LRU (Least Recently Used), CLOCK[Corbato 1969] and Second Chance [Tanenbaum 2007] algorithms are considered the most widely used replacement policies. CLOCK and Second chance algorithm are functionally identical and reach hit ratios close to LRU.

CLOCK and Second Chance are improvements of FIFO (First In First Out) replacement policy. Both can be implemented by using circular lists. The difference between them is in the way they manage the circular list. Second Chance always tries to pick the page in the front of list (like FIFO) and verify its reference bit. If the bit is 1, the page is inserted back into the list resetting reference bit to 0. Otherwise, the page is swapped out. CLOCK in turn maintains an iterator (called hand) pointing to the last page examined in the list. In a page fault event, the reference bit in iterator's location is inspected. If the bit is 1, it is cleared and the process is repeated until a page is replaced (bit set to 0).

LRU has been designed considering that (magnetic) disk access is at least one order of magnitude slower than main memory access. Every time a page is referenced, its respective reference counter is incremented. Therefore, LRU works by evicting a page (or pages) with the lowest reference counter. In other words, if a page P is a hot page, it tends to be kept in the buffer pool. Thus, if a clean page P has a reference counter greater than a dirty page P', LRU chooses the dirty page P' to be evicted. In this case, P' must be written back to disk. Observe that if the clean page P had been chosen to be removed from the buffer pool, no write operation would be necessary.

Moreover, in LRU, page reference does not take into account which operation (read or write) is performed in a page nor the frequency in which each operation is executed on the page. Therefore, the LRU behavior tends to generate more write operations into media, which means a negative side-effect for SCM performance.

LRU presents also a well-known weakness when sequential (and long) access patterns are dominants or at least mixed in workloads. In this case, LRU shows a performance degeneration because hot pages are pushed away by sequentially accessed pages. In the majority of DBMS implementations, this issue is mitigated by using a double ended queue (deq) that has a LRU end and a MRU (Most Recently Used) end. Thus, sequentially accessed pages are inserted into MRU end, such that, in a evicting situation, pages at MRU end are strong candidates to be replaced. Nevertheless, LRU assumes a (magnetic) disk access with no read-write asymmetry.

3.2 SCM-oriented Policies

There are some works on buffer replacement specifically designed for SCM media [Park et al. 2006; Jung et al. 2008; Li et al. 2009; Jo et al. 2006; Kim and Ahn 2008; Ou et al. 2009]. All of them are derivations of LRU or Second Chance policies.

FAB (Flash-Aware Buffer) [Jo et al. 2006] and BPLRU (Block-level LRU) [Kim and Ahn 2008] are block-based (instead of page-based) LRU algorithms. In other words, they are specifically designed to flash memories (block-addressable), lacking, thus, the generality for other SCM media types. FAB

evicts a block that contains the largest number of pages. BPLRU, in turn, employs a page-padding technique to write buffer inside SCM media. BPLRU tries to compensate LRU problem for sequential writes by evicting blocks sequentially written prior to randomly written blocks. Rather, SCM-BP uses a page-based replacement policy. In fact, the application of FAB and BPLRU to PCM media is quite difficult, since PCM is byte-addressable.

LRU-WSR (LRU Write Sequence Reordering) [Jung et al. 2008] and its extension CCF-LRU (Cold-Clean-First LRU) [Li et al. 2009] are based on LRU and Second Change algorithms. LRU-WSR evicts clean and cold-dirty pages prior to hot-dirty pages. In this point, SCM-BP is similar to LRU-WSR. CCF-LRU refines the LRU-WSR idea by distinguishing between cold-clean and hot-clean pages. The problem with LRU-WSR and CCF-LRU is their cost-based evicting process that assumes directly read-write asymmetry cost. Consequently, it is difficult to them decide if and when a cold-dirty page should be evicted over a hot-clean page.

CFLRU (Clear-First LRU) [Park et al. 2006] and its extension CFDC (Clear-First Dirty-Clustered) [Ou et al. 2009] are policies where clean pages are always selected to be evicted over dirty pages. SCM-BP follows the same policy. Although, all of them use a two-region scheme, there are, however, differences among them. CFLRU uses a clean region defined by a tuning parameter called windows size (w) to be used in page eviction. The definition of w depends on read-write asymmetry of the media and the intensity of the workload. Therefore, it is difficult to determine an optimal value for w. CFDC address the clean region problem by creating two queues, one for clean pages and one for dirty pages. This eliminates the cost for search a clean page in occasion of a buffer fault. Further, CFDC clusters pages in dirty queue and maintain a priority queue of page clusters. This is an attempt to deal with the locality and sequential access pattern and turns CFDC similar to FAB and BPLRU. However, clusters have variable sizes. CFDC has a time complexity higher than LRU due to its cluster's prioritization formula. SCM-BP is simpler than CFDC, not too much time-dependent and does not uses directly read-write asymmetry as a component for its evicting decision.

To conclude this section, we summarize the discussion on SCM-oriented Policies with a qualitative comparison, which is depicted in Table I. For that, we have applied the following criteria. *Page-based replacement* indicates whether the policy uses a page as replacement unit and, therefore, it may be applicable to other SCM media types than flash (e.g., PCM). *Asymmetry level dependency* criterion means that a policy is dependent directly or indirectly on the specific-media asymmetry factor. Policies that depend directly on a media-specific level of asymmetry are hard to tune when they have to be applied to other SCM media type. Since most published policies are an extension of LRU, the *access pattern supported* criterion indicates whether a policy handles both sequential and random access patterns (see Section 3.1). The last criterion indicates that policies taking into account operation types (read or write) and the execution frequency of such operations on the buffer pool are more efficient to SCM media.

	Policies				
Criteria	FAB	BPLRU	LRUWSR	CFLRU	CFDC
Page-based replacement	no	no	yes	yes	yes
Asymmetry level dependency	indirectly	indirectly	directly	directly	indirectly
Access pattern supported	sequential and random	sequential and random	random	random	sequential and random
Operation types and their frequencies taken into account	no	no	no	no	no

Table I: Summary of SCM replacement policies

4. SCM-BP: A BUFFER MANAGEMENT TECHNIQUE FOR SCM IN DATABASE CONTEXT

The main characteristic of the SCM-BP is its ability to privilege write-intensive pages. Thus, SCM-BP tries to keep in the buffer pages with the highest numbers of write operations. By doing so, SCM-BP tends to issue a few number of write operations to SCM media and contributes, consequently, to a better database performance w.r.t. SCM.

Figure 3 brings an abstract model of the buffer pool managed by SCM-BP. SCM-BP maintains two regions in the buffer: read region and write region. The area allocated to each region is autonomously and automatically adjusted by SCM-BP. Written pages are assigned to write region, whereas pages, on which only read operations occurred, stay in read region. Whenever a page in read region has to be modified, it is migrated to the write region. Such migration does not necessarily swap page off the buffer, it is just a management issue. Nevertheless, a migration may trigger an increasing or decreasing operation for a given region. If there is no free pages in a region, an increasing operation may be executed for that region, whereas the other region size has to be decreased. In Section 4.1 such operations are described and analysed.

Obviously, it is necessary to set up limits after which a region increasing should stop. Otherwise, a given workload would monopolize all buffer resources. ***Read threshold (r-threshold)*** and ***write threshold (w-threshold)*** are defined as database system parameters (see Figure 3). Therefore, ***r-threshold*** and ***w-threshold*** determine a maximum value for region increasing. They can be modified by the user, whenever the database system starts up. Read threshold and write threshold should be set according to the behaviour of the expected database workload (e.g., OLTP or OLAP). Internally, SCM-BP always keeps the current size of each region.

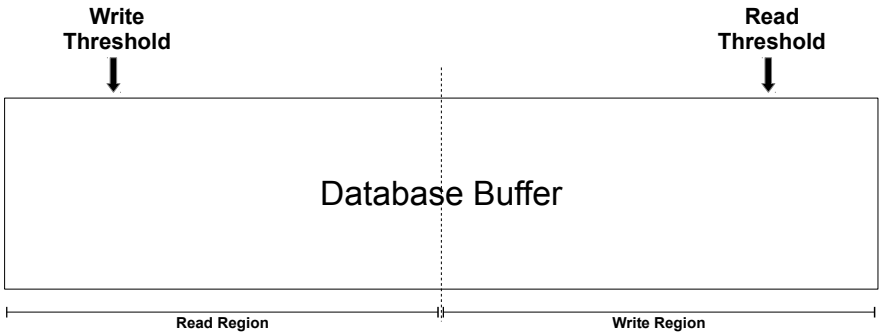
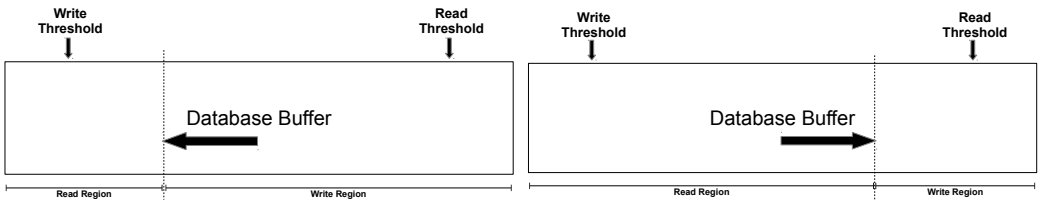


Fig. 3: SCM-BP overview



(a) Increasing Write Region (b) Increasing Read Region

Fig. 4: SCM-BP Technique

4.1 SCM-BP's Intelligent Behaviour

The proposed policy keeps a read-counter for read region and a write-counter for write region recording the number of (read and write) operations happened in each region. The sum of read and write counters provides the overall number of operations in the buffer. Additionally, the hit ratio for each region is also maintained by SCM-BP.

For each page in the buffer pool, SCM-BP maintains two different data types. The first data type, denoted the *load-timestamp*, has the semantic of capturing the moment in which a page has been loaded to the buffer. The second data type, called *operation-counter*, registers the number of operations executed on a given page in the buffer pool. Thus, whenever a page is loaded to the buffer, its load-timestamp is recorded and the page goes to the read region. Every time a write or read operation is executed on a given page, the page's operation-counter is incremented and the read/write-counter of the read/write region is also incremented.

Those counters and the load-timestamp provide the necessary support for identifying variations in the DBMS workload over the time. More specifically, page counters along with region counters and the (r-/w-)thresholds make possible to define the most appropriate time to increase or decrease a region in order to react to workloads variations (read-intensive to write-intensive and vice-versa).

A parameter passed to SCM-BP (e.g., by the DBA) defines how much to increase write (or read) region. Let us call it *write-region increasing (***wri***) factor*. Similarly, *read-region increasing (***rri***) factor* indicates how much the read region size should be increased. Whenever a page has to be written, it is migrated to the write region. Thus, SCM-BP tries to increase the write region and consequently to decrease the read region (Figure 4a). SCM-BP has to deal with two different situations:

(1) If ***w-threshold*** is already reached. In this case, we cannot augment write region size then we apply buffer replacement policy of the SCM-BP which evicts from write region a page whose write counter is the lowest one;
(2) Case ***w-threshold*** has not been reached yet, SCM-BP will enlarge write region area. The *wri factor* is applied and write region size is enlarged. Consequently, read region size is decremented. This situation induces SCM-BP to remove pages from the read region. The victim page is chosen in the read region by the LRU replacement algorithm. Clearly, we sacrifice a read page. However, recall that a read operation in SCM media is normally two orders of magnitude faster than those on magnetic disks and a write operation is even slower than those in magnetic disks [Ou et al. 2009]. Moreover, a write operation in SCM media is up to one order of magnitude slower than a read operation.

Nonetheless, there is the case that SCM-BP may increment the read region and consequently shrink write region (see Figure 4b). As already mentioned, a key issue implemented by SCM-BP is to privilege written pages in order to keep them in main memory as long as possible. For that reason, the decision to augment read region size should be based on the DBMS workload.

In this sense, SCM-BP implements a *workload-sensitive function* (for short, *wsf*) to decide to increment or not the read region. The intuition behind this function is the following: When the miss ratio of read region raises up and at same time the number of write operations in the write region drops down (or does not increase in a given time window), the function should be applied. It is important to note that the higher the miss ratio in the read region, the lower the hit ratio and the higher swap-in operation frequency. The return value provided by *wsf* is the quantitative indicator for augmenting (or not) the read region.

Thus, *wsf* is defined by $WSF(x, y) = x^4/y^9$ where x is the swap-in frequency in the read region observed in a period of time Δt and y represents the write-operation frequency on pages located in the write region (observed in the same Δt). Figure 5 depicts the behavior of the implemented workload-

sensitive function *wsf*. Observe that the curve slope of *wsf* guarantees that when read-region's miss ratio raises up in Δt and the write frequency of write region drops down or not increase in Δt, is an indication of the need to enlarge read region. Δt is the difference between current timestamp (decision time) and the oldest load-timestamp of read region. In order to decide the moment to increment the read region, SCM-BP implements the following criterion: whenever $WSF(x, y) = 1$ read region area is incremented. The *rri factor* is thus applied and read region is enlarged. Such an operation may trigger swap-out operations in write region. In this case, the SCM-BP's replacement policy takes place evicting pages with lowest write counters.

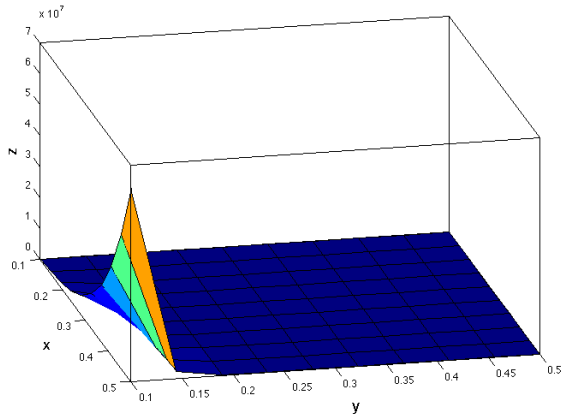


Fig. 5: Workload-sensitive function graph

4.2 SCM-BP algorithms

In this section, the algorithms implemented by SCM-BP are detailed. There are three important procedures in SCM-BP: *(i)* The read procedure (Algorithm 1) in which a page is read from the non-volatile media and loaded into the buffer; *(ii)* The write procedure (Algorithm 2) in which a write operation on a page located in the buffer pool is serviced, and; *(iii)* The *adaptRegionSize* procedure regulates the region's size (Algorithm 3).

Algorithm 1: read

Input: a request for a page p to be loaded into buffer
Output: the requested page p

```
1  begin
2      if buffer.lookup(pageID) then
3          updateBufferProfile();
4          return getPage(pageID);
5      else
6          if (WSF == 1) then
7              adaptRegionsSize(read region);
8          Fetch the requested page p from SCM media and store on buffer;
9          updateBufferProfile();
10         return getPage(pageID);
```

Read and write procedures may trigger the execution of *adaptRegionSize*. This may occur whenever a decision on increasing the read or write region should be made (lines 6-7 of read procedure and line

3 of write procedure). For example, consider that a requested page p is not in buffer (read procedure - Algorithm 1). Thus, p has to be brought from SCM media to read region. This operation may trigger a change of read-region's size. For that, the workload-sensitive function $WSF(x, y)$ is applied (lines 6-7 of Algorithm 1). Case $WSF(x, y) = 1$, the procedure *adaptRegionSize* is called.

The function *updateBufferProfile* (lines 3 and 9 of Algorithm 1 and line 5 of Algorithm 2) is responsible to maintain counters of the buffer manager. Therefore, for each read and write operation on a page of the buffer pool, a call to *updateBufferProfile* is executed. For example, when an update in-place happens in the buffer (line 4 of Algorithm 2), *updateBufferProfile* increments write counter and the hit counter, case the requested page is already in buffer).

Algorithm 2: write

```
Input: a page p and data to be written on that page
1 begin
2     if buffer.lookup(pageID) == false then
3         adaptRegionsSize(write region);
4     Perform a buffer in − place update;
5     updateBufferProfile(pageID);
```

The *adaptRegionSize* procedure is applied to each region. First, we verify whether the actual region size plus the region increment factor (***rri*** or ***wri***) exceeds the region's threshold (***r-/w-threshold***) (line 3 for write region and line 10 for read region). If this check evaluates to true then some page replacement policy is applied. For example, if write region size has to be adjusted, we may apply LRU or SCM-BP policy depending on verification evaluation. If read region is to be shrunk, LRU is applied on it (line 4 of Algorithm 3). Otherwise, if we cannot enlarge write region SCM-BP takes place (line 8 of Algorithm 3). Similar reasoning is used for read region (lines 9-15 of Algorithm 3). In both cases, region's size is adjusted accordingly (lines 5-6 and 12-13 of Algortihm 3).

Algorithm 3: adaptRegionsSize

```
Input: a request to adapt the buffer regions size in a region r
1  begin
2      if (r == write region) then
3          if (wr.actualSize + wri <= wr.w − threshold) then
4              apply LRU policy to read region;
5              wr.actualSize ⟵ wr.actualSize + wri;
6              rr.actualSize ⟵ rr.actualSize − wri;
7          else
8              apply SCM − BP policy to write region;
9      if (r == read region) then
10         if (rr.actualSize + rri <= rr.r − threshold) then
11             apply SCM − BP policy to write region;
12             rr.actualSize ⟵ rr.actualSize + rri;
13             wr.actualSize ⟵ wr.actualSize − rri;
14         else
15             apply LRU policy to read region;
```

5. EVALUATION

In order to make an assessment of our proposal, we have performed some empirical tests on SCM-BP. As comparative evaluation, we have also implemented LRU policy and submit both to the same workload measuring the performance in terms of hit ratio, number of write operations performed into media, and the average write time. The number of write operations makes possible to evaluate the SCM-awareness of the buffer management policy.

We have generated two sets of workloads to the buffer manager. A set containing sequences of random operations (read and write) and another on encompassing mixed sequences of random and sequential operations. For each random sequence, we varied the number of read and write operations: 20% writes (respectively 80% reads), 40% writes (resp. 60% reads), 50% writes (50% reads), 60% writes (40% reads), and 80% writes (20% reads).

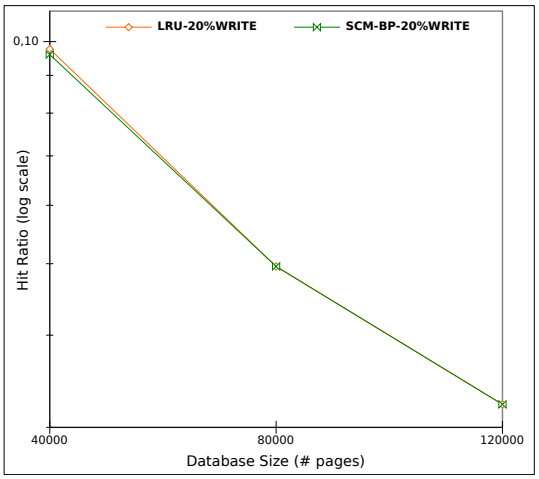
The database size used was also specified. We have experimented database sizes (db size) with 40,000, 80,000, and 120,000 pages. Each sequence of operations was applied to each db size tested. The buffer size, in terms of number of pages, has been set to a constant value of 4,000 pages. The decision to test with a very small buffer size is to stress the limits of both LRU and SCM-BP policies.

The experiments have been carried out with an Intel Pentium Quad Core 1.83 GHz server machine using a 64-bit Linux operating system. This machine has a 4-Gbytes main (RAM) memory. Since we aim at empirically evaluating buffer techniques and SCM drives from a database point of view, we have run experiments with the operating system and the database files stored on an SSD drive. The used SSD drive has been a Corsair Force 2.5' SSD drive with 120 Gbytes attached to server machine through SATA II interface. Each test has been run three times and we have plotted the average value gathered. The experiments have been implemented as an external layer to a DBMS.

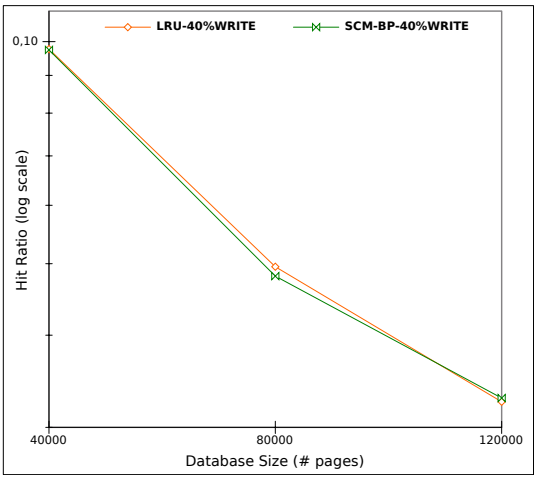
In Figure 6, we vary the database size and gather the hit ratio for each size for LRU and SCM-BP policies. As depicted in Figure 6, SCM-BP presents a hit ratio close to LRU. This means that SCM-BP is quite successful to detect and keep hot pages (whether read or written pages) into the buffer. Figure 6 also shows an accentuate decreasing of hit ratios. This is an expected tendency because with 40,000 pages we have a buffer size of 10% of db size, and, hit ratios reach values close to 10%. When db sizes increase, buffer size becomes too small w.r.t db size (e.g., for 120,000 db pages, buffer size is approximately 3% of db size) and hit ratios drop down accordingly.

It is well know that write operations executed on SCM media may jeopardize system performance, due to read/write asymmetry, and shorten SCM life time. Therefore, SCM-awareness has been studied and its results are depicted in Figure 7. Again, we vary the database size and as we can see, SCM-BP enables a number of write operations far fewer than that of LRU. Two extreme workloads (20% write and 80% write) favour respectively LRU and SCM-BP due to own nature of policies. Nevertheless, in a workload clearly favourable to LRU (20% write), for a db size of 120,000 pages, LRU reaches 42 write operations whereas SCM-BP has only 23 write operations (i.e., almost 55% fewer). However, these workloads are not common in practice. Thus, other workloads have been also evaluated (see Figure 7). For such workloads, the ratio between write operations executed on the SCM when using LRU and write operations when using SCM-BP varies from 70% (in 40% write workload) to 80% (in 60% write workload). Accordingly, a DBMS running on an SCM and implementing LRU-based replacement policy may negatively impact the system performance and may shorten SCM life time. SCM-BP, in turn, contributes to a better performance of the DBMS and may increase the life time of an SCM drive. Thus, taking into account the number of write operations makes the proposed buffer management strategy SCM-aware (see Table II).

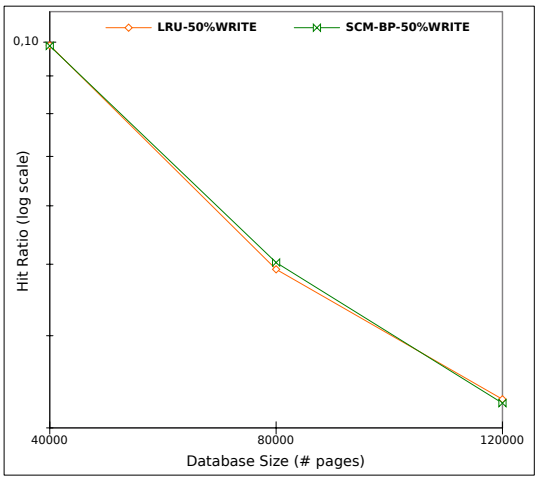
The effect of sequential access patterns in the tested buffer replacement policies has been investigated as well. In Figure 8, we have plotted the results for hit ratio (Figure 8a) and the number of write operations (Figure 8b) for a workload with 40% of write operations. The other workloads have been omitted because they presented similar results.



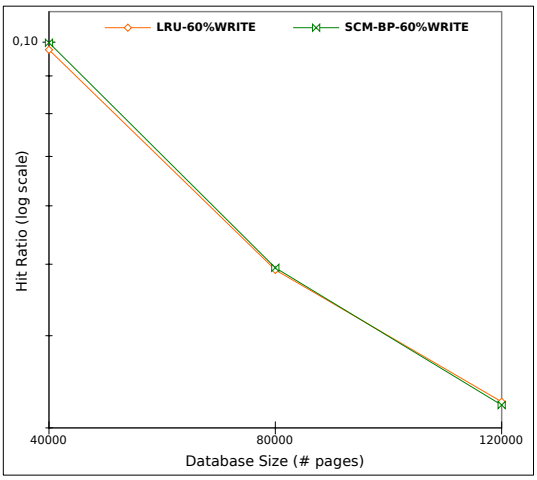
(a) 20% Write Workload



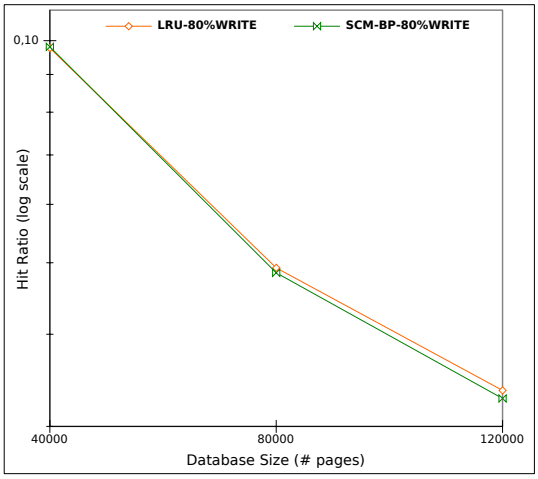
(b) 40% Write Workload



(c) 50% Write Workload

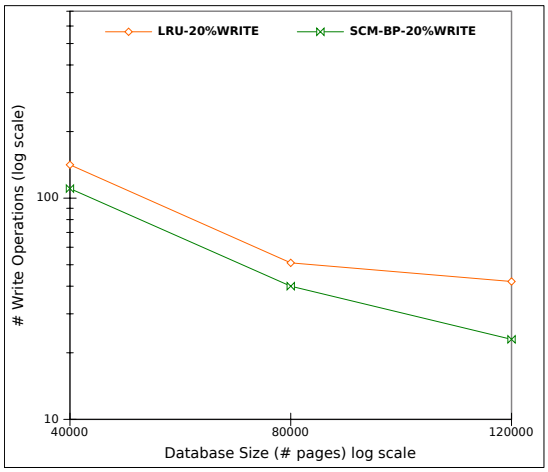


(d) 60% Write Workload

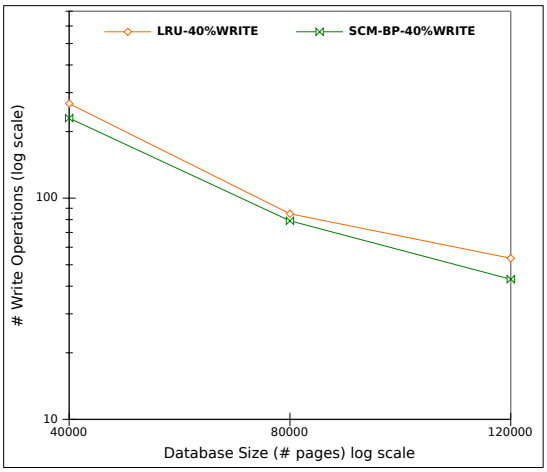


(e) 80% Write Workload

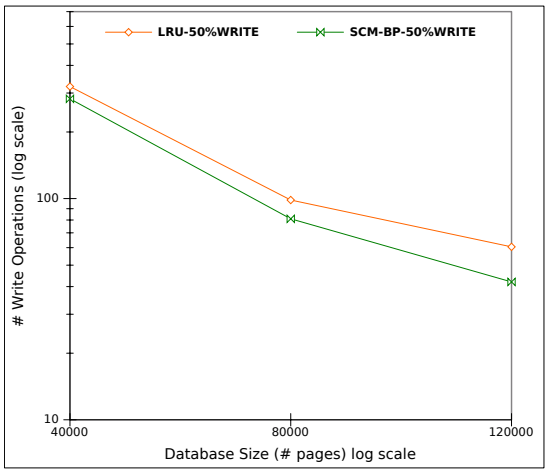
Fig. 6: Comparative study of **hit ratio** – random access pattern



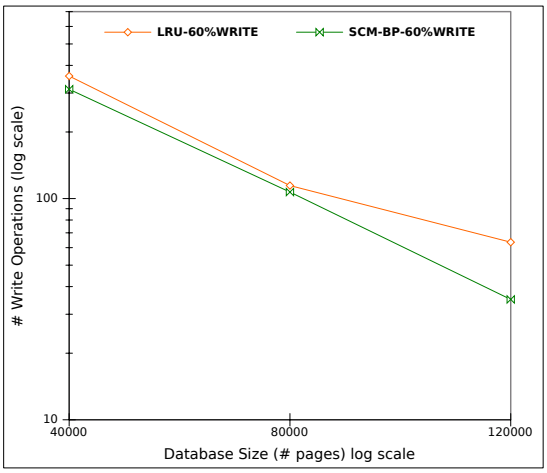
(a) 20% Write Workload



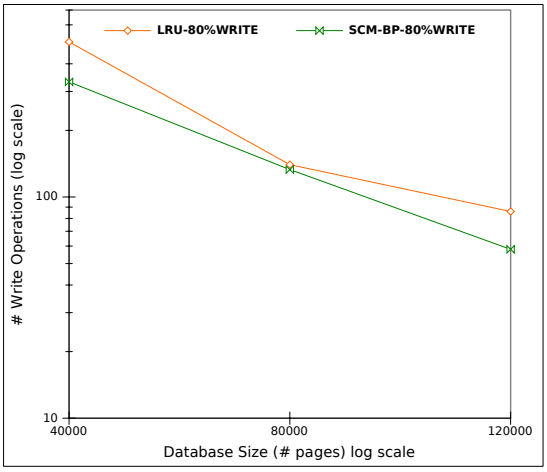
(b) 40% Write Workload



(c) 50% Write Workload



(d) 60% Write Workload



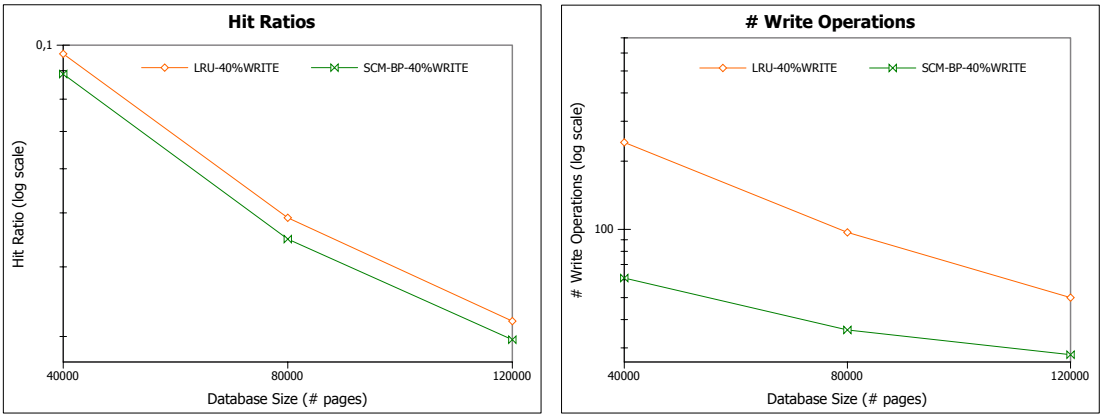
(e) 80% Write Workload

Fig. 7: Comparative study of **write operations** – random access pattern

	Policies					
Criteria	FAB	BPLRU	LRUWSR	CFLRU	CFDC	SCM-BP
Page-based replacement	no	no	yes	yes	yes	yes
Asymmetry level dependency	indirectly	indirectly	directly	directly	indirectly	indirectly
Access pattern supported	sequential and random	sequential and random	random	random	sequential and random	sequential and random
Operation types and their frequencies taken into account	no	no	no	no	no	yes

Table II: SCM-BP and other SCM replacement policies

Looking more closely to Figures 8a and 8b, one can observe that SCM-BP has shown a slightly worse hit ratio w.r.t. the LRU hit ratio, on average SCM-BP's hit ratio is 10% lesser. However, the number of write operations of SCM-BP varies from 178% (db size = 120,000) to 300% (db size = 40,000) less than that of LRU. This means that while SCM-BP is more sensitive to workload presenting sequential access patterns w.r.t. hit ratio, it handles better write operations in face of such patterns.



(a) Hit ratios for SCM-BP and LRU

(b) # of write operations for SCM-BP and LRU

Fig. 8: Comparative study of sequential access patterns – **Hit Ratio and Write Operations**

6. CONCLUSIONS AND FUTURE WORK

In this article, we have proposed a SCM-aware buffer replacement strategy called SCM-BP. We have also executed comparative tests to investigate SCM-BP performance against that of well-known LRU policy in SCM media.

SCM-BP presents a hit ration close to that of LRU indicating the efficiency of our approach. Furthermore, SCM-BP outperforms LRU by keeping a greater number of written pages (hot dirty pages) in memory buffer whether when random operations take place or in face of sequential access patterns. By dealing with better write operations, SCM-BP enables a longer mean life for SCM media.

In near future, we plan improve SCM-BP to reach even better hit ratios in the presence of sequential patterns. Another improvement is to define a way to autonomously and dynamically adapt database parameters (e.g. read and write thresholds – *r-threshold* and *w-threshold* – and the increasing rate of read and write regions – *rri* and *wri*) when a workload varies over the time. Extending comparative evaluation w.r.t. SCM-oriented Policies is also planned. Nevertheless, we intend to implement SCM-BP in PostgreSql.

Some authors advocate that quite often OLTP databases can fit within main memory [Harizopoulos et al. 2008]. Of course, such a feature reduces the impact of the buffer management component on DBMS performance. Nonetheless, we claim that this is not the case of OLTPs running on very large databases. Furthermore, for main memory databases, at least the log file has to reside in non-volatile memory and the recovery process may take too much time, which is not reasonable for many OLTP applications at all.

Finally, as an long-term extension to our work, we should investigate the influence of SCM-BP policy in logging techniques and database recovery process [Fang et al. 2011].

REFERENCES

Corbato, F. J. A Paging Experiment with the Multics System. In *Festschrift: In Honor of P. M. Morse*. MIT Press, USA, pp. 217–228, 1969.

Fang, R., Hsiao, H.-I., He, B., Mohan, C., and Wang, Y. High performance database logging using storage class memory. In *Proceeding of the IEEE International Conference on Data Engineering*. ICDE'11. IEEE, USA, pp. 1221–1231, 2011.

Harizopoulos, S., Abadi, D. J., Madden, S., and Stonebraker, M. Oltp through the looking glass, and what we found there. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD '08. ACM, New York, NY, USA, pp. 981–992, 2008.

Jo, H., Kang, J.-U., Park, S.-Y., Kim, J.-S., and Lee, J. FAB: flash-aware buffer management policy for portable media players. *IEEE Transactions on Consumer Electronics* 52 (2): 485–493, 2006.

Jung, H., Shim, H., Park, S., Kang, S., and Cha, J. LRU-WSR: integration of lru and writes sequence reordering for flash memory. *IEEE Transactions on Consumer Electronics* 54 (3): 1215–1223, 2008.

Kim, H. and Ahn, S. BPLRU: a buffer management scheme for improving random writes in flash storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*. FAST'08. USENIX Association, Berkeley, CA, USA, pp. 16:1–16:14, 2008.

Kim, K. and Koh, G.-H. Future memory technology including emerging new memories. In *Proceedings of the IEEE 24th International Conference on Microelectronics*. ICM'04. IEEE, USA, pp. 377–384, 2004.

Li, Z., Jin, P., Su, X., Cui, K., and Yue, L. CCF-LRU: a new buffer replacement algorithm for flash memory. *IEEE Transactions on Consumer Electronics* 55 (3): 1351–1359, 2009.

Mielke, N., Marquart, T., Kessenich, J., Belgal, H., Schares, E., Trivedi, F., Goodness, E., and Nevill, L. R. Bit error rate in NAND flash memories. In *Proceedings of the IEEE International Reliability Physics Symposium*. IRPS '08. IEEE, USA, pp. 9–19, 2008.

Ou, Y., Härder, T., and Jin, P. CFDC: a flash-aware replacement policy for database buffer management. In *Proceedings of the Fifth International Workshop on Data Management on New Hardware*. DaMoN '09. ACM, New York, NY, USA, pp. 15–20, 2009.

Park, S.-y., Jung, D., Kang, J.-u., Kim, J.-s., and Lee, J. CFLRU: a replacement algorithm for flash memory. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*. CASES '06. ACM, New York, NY, USA, pp. 234–241, 2006.

Tanenbaum, A. S. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.