PBFilter: Indexing Flash-Resident Data through Partitioned Summaries

Shaoyi Yin [*,**,***]
*INRIA Rocquencourt
78153 Le Chesnay - France
Fname.Lname@inria.fr

Philippe Pucheral [*,**]
**PRiSM Lab, Univ. of Versailles
78035 Versailles - France
Fname.Lname@prism.uvsq.fr

Xiaofeng Meng [***]
***Renmin Univ. of China
100872 Beijing – China
{yinshaoy, xfmeng}@ruc.edu.cn

ABSTRACT
NAND Flash has become the most popular persistent data storage medium for mobile and embedded devices. The hardware characteristics of NAND Flash (e.g. page granularity for read/write with a block-erase-before-rewrite constraint, limited number of erase cycles) preclude in-place updates. In this paper, we propose a new indexing scheme, called PBFilter, designed from the outset to exploit the peculiarities of NAND Flash.

Categories and Subject Descriptors: D.4.3
[Operating Systems]: File Systems Management – *Access methods, file organization.*

General Terms: Algorithms, Performance, Design.

Keywords: Indexing method, NAND Flash, Bloom Filter.

1. INTRODUCTION
NAND Flash memory stands out as the best adapted storage medium for an ever wider spectrum of mobile and embedded devices (PDAs, cell phones, cameras, sensors, smart cards, USB keys, mp3 readers etc), because of its excellent properties such as shock resistance, size, energy consumption, performance and ease of on-chip integration. However, NAND Flash exhibits specific hardware characteristics making database management very challenging. Reads and writes are done at a page granularity, as in traditional disks, but writes are more time and energy consuming than reads. In addition, a page cannot be rewritten before erasing the complete block containing it, a costly operation. Finally, a block wears out after 10^5 to 10^6 repeated write/erase cycles. Due to this, updates are usually performed "out-of-place", meaning that a page is copied in another location before being modified, entailing address translation and garbage collection overheads.

Database storage models dedicated to NAND Flash have been proposed in [4, 5]. The strong point of [5] is to offer a storage and buffering approach hiding the Flash peculiarities to the upper layers of the DBMS. Updates in Flash are delayed thanks to a log stored in each physical block and the accurate version of each page is rebuilt at load time. The updates are physically applied to a page when the corresponding log region becomes full. While elegant, this general method is not well suited to manage hot spot data like index pages because of frequent log overflows. Recent works addressed this issue by adapting B+Tree-like structures to NAND Flash [2, 6, 7]. While different in realization, these methods rely on a similar approach: to delay the index updates thanks to a log dedicated to the index and batch them with a given frequency with the objective to group updates related to a same index node. We call these methods *batch methods* hereafter. Batch methods entail a new trade-off between the number of reads and writes but good performance for both is difficult to achieve at once due to the Flash characteristics. In addition, read/write performance is no longer the sole metrics of interest in the Flash context. First, RAM consumption is a primary concern in many mobile and embedded devices, today the main target of Flash chip manufacturers. Second, the data organization in Flash impacts the costs of address translation and stale data reclamation. These indirect costs have been shown high and not easily predictable [6].

Hence, we argue that a larger set of metrics has to be considered when designing an indexing technique for Flash-resident data, namely:

- **M1:** Read cost at lookup time.
- **M2:** Read and write cost at insertion time.
- **M3:** Flash usage in terms of space occupancy and effort to reclaim stale data.
- **M4:** RAM consumption.
- **M5:** Performance and resource consumption predictability.

2. PBFILTER INDEXING SCHEME
Instead of adapting traditional indexes, as batch methods do, this paper proposes a new indexing scheme, called PBFilter, designed to exploit natively the peculiarities of NAND Flash. PBFilter performs index updates eagerly and makes this acceptable by organizing the complete database as a set of sequential data structures, as pictured in Figure 1. The objective is to transform database updates into append operations so that writes are produced sequentially, an optimal scenario for NAND Flash.

When a new record is inserted, it is added at the end of the record area (RA). Then, a new index entry composed by a couple <key, pt> is added at the end of the key area (KA), where key is the primary key of the inserted record and pt is the record physical address[1]. If a record is deleted, its identifier is inserted at the end of the delete area (DA) but no update is performed in RA nor in KA. A record modification is implemented by a deletion (of the old record) followed by an insertion (of the new record value). To search for a record by its key, the lookup operation first scans KA, retrieves the required index entry if it exists, check that pt $\notin$DA and gets the record in RA. Assuming a buffering policy allocating one buffer in RAM per sequential data structure, this database updating process never incurs rewriting a same page in Flash.

The benefits provided by this simple database organization are obvious with respect to the metrics introduced above. Indeed, a

CIKM'08, October 26-30, 2008, Napa Valley, California, USA.
ACM 978-1-59593-991-3/08/10.

[1] Like all state of the art methods, we concentrate the study on primary keys. The management of secondary keys is discussed in [8].

lower bound is reached in terms of reads/writes at insertion time (metric M2) and in terms of Flash space occupancy (metric M3), a single RAM buffer of one page is required per sequential structure RA, KA and DA (metric M4) and finally PBFilter can bypass the FTL address translation layer[2] and avoids the need for garbage collection, two major sources of performance unpredictability (metric M5).

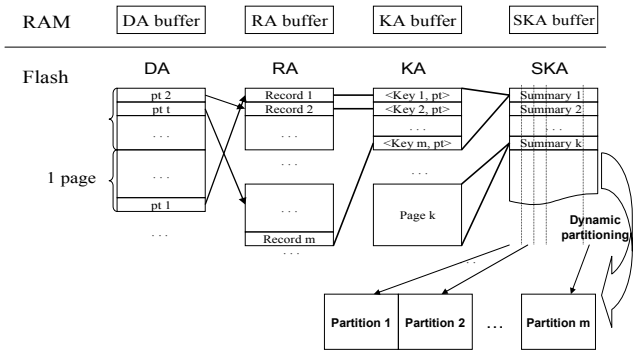

Figure 1. Database organization

The drawbacks of this organization are obvious as well. The performance for metric M1 reaches (|KA |/2 + |DA | + 1) on the average, where | | denotes the page cardinality of a structure. We introduce two additional principles to circumvent this drawback with a minimal degradation of the benefits listed above.

Summarization: Summarization condenses each KA page into a summary record. Summary records are sequentially inserted into a new structure called SKA through a new RAM buffer of one page. Then, lookups do a first sequential scan of SKA and a KA page is accessed for every match in SKA in order to retrieve the requested key (if it exists). Summarization introduces an interesting trade-off between the compression factor *c* (c= |KA |/ |SKA |) and the fuzziness factor *f* (i.e., probability of false positives) of the summary, the former decreasing the I/O required to traverse SKA and the latter decreasing the I/O required to access KA. The net effect is replacing the lookup cost in KA by (|KA |/2c + f | |KA | |/2) on the average, where || || denotes the element cardinality of a structure. The positive impact on metrics M1 and M5 can be very high for favorable values of *c* and *f*. The negative impact on M4 is limited to a single new buffer in RAM. The negative impact on M2 and M3 is linear with |SKA | and then depends on *c*. Different algorithms can be considered as candidate "condensers", with the objective to reach the higher *c* with the lower *f*, if only they respect the following property: *summaries must allow membership tests with no false negatives*. In [8], we show that Bloom Filter [3] is a very good condenser and provides subtle tuning capabilities for c and f. Bloom Filter summaries can then adapt to different requirements in terms of lookup performance and targeted hardware platform.

Partitioning: the idea is to vertically split a sequential structure into *p* partitions so that only *p'*<*p* partitions have to be scanned at lookup time. PBFilter applies partitioning to SKA in such a way that the membership test can be done without considering the complete summary value. The larger *p*, the higher the partitioning benefit and the better the impact on metrics M1 and M5. On the other hand, the larger *p*, the higher the RAM consumption (*p* buffers) or the higher the number of writes into the partitions (less than *p* buffers) with the bad consequence of reintroducing page moves and garbage collection. Again, different partitioning strategies can be considered with the following requirement: *to increase the number of partitions with neither significant increase of RAM consumption nor need for garbage collection*. In [8], we propose a dynamic partitioning mechanism for Bloom Filter summaries which matches the above requirements at the price of introducing a few reads and extra writes at insertion time.

[2] Several FTL provide different API levels (e.g., FTL, VFL, FIL levels in Samsung S-SIM FTL) allowing bypassing some functionalities (e.g., bad block management, ECC management, address translation, etc).

3. CONCLUSION

NAND Flash has become the most popular persistent storage medium for mobile and embedded devices, urging the database community to design Flash-aware indexing methods. Previous works in this area defer index updates thanks to a log and batch them with the objective to decrease the cost of writes in Flash. This introduces a complex trade-off between read and write performance and entails side-effects on the RAM consumption, Flash consumption and garbage collection cost.

PBFilter is an alternative to batch methods, particularly well suited to mobile and embedded devices. Its effectiveness on all metrics at once has been shown in [8] through a comprehensive analytical performance study. PBFilter has been implemented and integrated in the storage manager of an embedded DBMS dedicated to the management of secure portable folders [1]. The prototype runs on an experimental secure USB Flash platform provided by Gemalto.

Thanks to its tuning capabilities, PBFilter is easily adaptable to other Flash-based environments and can meet various application requirements in terms of file size, key size, read/write tradeoffs. Investigating other summarization and partitioning strategies could ever enlarge the application domain of PBFilter (e.g. indexing secondary keys), which is part of our current and future work.

4. ACKNOWLEDGMENTS

The authors wish to thank Luc Bouganim and Dennis Shasha for valuable comments on this paper. A special thank is also due to Jean-Jacques Vandewalle from Gemalto for his technical support. This work is partially supported by the French National Research Agency (ANR) and the Natural Science Foundation of China.

5. REFERENCES

[1] Anciaux, N., Benzine, M., Bouganim, L., Jacquemin, K., Pucheral, P., and Yin, S. Restoring the Patient Control over her Medical History. *21th IEEE Int. Symposium on Computer-Based Medical Systems (CBMS)*, 2008.

[2] Bityutskiy, A-B., JFFS3 Design Issues. *Tech. report*, Nov. 2005.

[3] Bloom, B. Space/time tradeoffs in hash coding with allowable errors. *Communications of the ACM, 13(7)*, 1970.

[4] Kim, G., Baek, S., Lee, H., Lee, H., and Joe, M. LGeDBMS: A Small DBMS for Embedded System with Flash Memory. *Int. Conf. on Very Large Data Bases (VLDB),* 2006.

[5] Lee, S-W., and Moon, B. Design of Flash-Based DBMS: An In-Page Logging Approach. *Int. Conf. on Management of Data (SIGMOD)*, 2007.

[6] Nath, S., and Kansal, A. FlashDB: Dynamic Self-tuning Database for NAND Flash. *Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2007.

[7] Wu, C., Chang, L., and Kuo, T. An Efficient B-Tree Layer for Flash-Memory Storage Systems. Int. Conf. on *Real-Time and Embedded Computing Systems and Applications (RTCSA)*, 2003.

[8] Yin, S., Pucheral, P., Meng X. PBFilter: Indexing Flash-Resident Data through Partitioned Summaries. Tech. Rep. RR-6548. INRIA. 2008.