LETTER

Flash-Aware Page Management Policy of a Navigation-Specialized Mobile DBMS for an Incremental Map Update

KyoungWook MIN, Member and JeongDan CHOI, Nonmember

SUMMARY  The performance of a mobile database management system (DBMS) in which most queries are made up of random data accesses if the NAND flash memory is used as storage media of the DBMS is degraded. The reason for this is that the performance of NAND flash memory is good for writing sequentially but poor when writing randomly. Thus, a new storage structure and querying policies are needed in mobile DBMS when flash memory is used as the storage media. In this letter, we propose a new policy of database page management to enhance the frequent random update performance, and then evaluate the performance experimentally.

key words: mobile DBMS, flash memory, database page management, mobile navigation, navigation map update

1. Introduction

NAND flash memory is the most popular storage medium of mobile devices such as digital cameras, MP3 players, and smart phones. Flash memory has different characteristics from disk memory, which is a general storage medium [1]. As the most unusual features, an in-place update is operated by erasing a block before overwriting the sector causes serious performance degradation owing to the expensive cost of an erase operation; for example, the cost ratio of read, write, and erase operations in a Samsung K9WAG08U1A 16 Gbits SLC NAND model flash memory device is 1 : 2.5 : 18.7.

There are some research and development opportunities to increase the performance of this storage media as these features of the flash memory. There have been several researches about the FTL (Flash Translation Layer). The FTL, which is intermediate software between the file system and flash memory device, provides the same interfaces as a hard disk, allowing flash memory to be used as a block device [2]. Another important role of the FTL is the use of algorithms for an out-place update. When an overwrite is requested at a particular sector, an out-place update algorithm writes data at a new sector instead of erasing the block that contains that sector. Out-place update algorithms include log block policy [3], FAST [4], LAST [5] and so on. But FTL is very limited in embedded market and it is difficult to access the internal flash memory device by user itself.

The performance of a sequential overwrite (update) operation is good in flash memory as the number of merge and erase operations are reduced by the out-place update algorithm in the FTL. However, as continuous random update operations cause frequent merges and erase executions, the performance is degraded. In addition, the performance of a mobile DBMS, in which most queries are made up of random data accesses, is degraded if flash memory is used as the storage media of the DBMS. Thus, it is needed to provide a flash-aware storage structure and query processing methods that are able to guarantee the performance of frequent random update query executions in DBMS.

In [6], the flash-aware DBMS has been studied to avoid the high latency of write and erase operations that would be caused by small random write requests, changes made to a data page are buffered in-memory on the per-page basis instead of writing the page in its entirety, and then the change logs are written sector by sector to the log area in flash memory for the changes to be eventually merged to the database. But this policy is tight coupling between DBMS and storage devices. That is to say, the physical block and sector of the flash memory device is able to be accessed in the DBMS. As before mentioned, it is impossible to access internal flash memory device in mobile application. So development of the flash memory device-independent DBMS is needed.

In this letter, we propose a new policy for managing a database page to enhance the performance of frequent random updates regardless of the FTL algorithms and without accessing internal flash memory device. We applied this policy to a navigation-specialized mobile DBMS that provides incremental map updates, called a Flash-aware Ubiquitous Navigation System (FUNs) [7], [8]. In addition, we evaluated the performance of this policy experimentally.

2. Flash-Aware Database Page Management Policy

To show the performance of a random update, we conducted a simple experiment as shown in Fig. 1. We performed Tasks 1 – Task 4 after creating a file in the flash memory of a mobile device. A page is a logical unit of a read/write in an application of the file system, and the pages shown are mapped to a physical address in the flash memory through an FTL that the applications are unable to access directly.

As the results of the experiment, all tasks have the same cost in the disk media, as the write and overwrite costs are the same; however, there are different costs in flash memory. Task 3 has a particularly poor performance because the continuous random update requests cause frequent merges and expensive block erase operations by the FTL algorithm internally. Thus, we use a simple idea of converting a random...
update sequence into a sequential update sequence. That is to say, as shown in Fig. 1, Task 2 or 3 is converted into Task 4. For example, if the page writing sequence is \{1, 2, 3, 1, 2, 3\}, the converted sequence is \{1, 2, 3, 101, 102, 103, 104, 105, 106\}, and if a read of page 3 is requested, then page 106 is returned instead of page 3.

There are two steps in the flash-aware database page management policy. The first step is a logical page re-mapping (PRM), and the second step is a page re-locating (PRL). The page overwrite sequence is re-mapped into a new sequential sequence in the logical PRM module. When the application using a mobile DBMS is terminated, all re-mapped pages are re-located to their original logical page in the PRL module. Mapping tables are needed for this algorithm, as shown in Fig. 2. A tablespace of FUNs corresponds to a file. All space in the tablespace is for data pages, except for the header information. The PRM area is added to this space for implementing this algorithm. All allocated pages 1 – n are in the page mapping table (PMT) and the mapping value is equal to the page id value initially. If page i is written to, which means the page is updated, a new row is added to the re-mapping table (ReMT), and row i in the PMT points to this new row. In addition, a new data page is added to the PRM area, and the new row of the ReMT points to this page. When another write is requested to page i, the row of ReMT, to which row i of the PMT is pointing, will be marked as invalid, and a new row will be added to the ReMT. In the PRL step, the data pages in the PRM area are copied to the original location by scanning the PMT sequentially.

This PRM and PRL algorithms are implemented on the storage manager in FUNs, which is responsible for the allocation, de-allocation, reading, and writing of the data page as requested by the buffer manager, as shown in Fig. 3. If the pages are dirty and are requested to be written to as per the page replacement policy in the buffer manager, the random writing sequence is converted into a sequential sequence in the storage manager.

3. Experiments

We tested and evaluated the performances of the flash-aware database page management policy, which is implemented in the FUNs storage manager. We used virtual data as well as real map data as the update dataset in the experiment. Table 1 describes the experiment environment and the update tasks to be compared for each performance. The data update experiment in the DBMS is quite different from the experiment, as shown in Fig. 1. The previous experiments were a simple execution of writing and updating a page in a file directly. However, in DBMS consisting of several program layers, record writing and updating queries are executed by a query processor, and the pages are written by the page replacement algorithm in the buffer manager. Finally, the pages are written to the file by the storage manager. A record size is about 100 bytes, and 10,000 records are initially inserted into FUNs sequentially. The update tasks are performed as shown in 1, 2, 4, and 4 in Table 1.

The results of the experiment are shown in Fig. 4. Figure 4 (a) shows the number of updated pages, and Fig. 4 (b) shows the update time as varying the buffer size, 1,000, 500 and 100 pages, for around four update tasks.

In the sequential update, 10,000 records are overwritten sequentially to 1,250 pages, and all records in a page in the buffer memory, which is 8 records in this experiment, can be updated by a single page reading and writing operation. That is, 10,000 records can be updated as reading and writing 1,250 pages only once. However, in a random update, a page reading and writing may be required to update
Table 1  The environment and data used in the experiments.

| Contents            | Description                        |
|---------------------|-------------------------------------|
| mobile device       | LG KCL, CPU 800 Mhz, 128 MB DRAM    |
| storage media       | NAND flash memory 2 GB              |
| database size       | page size: 1,024 byte               |
|                     | buffer size: 100 / 500 / 1,000 pages|
| record size         | about 100 byte                      |
| record insertion    | sequential 10,000 records insertion in initial |
| update tasks        | ① sequential 10,000 records update |
|                     | ② random 10,000 records update     |
|                     | ③ PRM 10,000 records update       |
|                     | ④ PRL update                       |
| result              | update time (sec) and page #        |

![Fig. 4](image-url)  Record update performance.

In this random update, the number of updated pages is 9,971 in a 100-page buffer. Of course, the larger the buffer size, the fewer the number of pages updated, as shown in Fig. 4 (a). In the PRM, the number of updates is the same as in a random update because new pages will be written when a previously recorded page is requested to be updated. In the PRL, the number of updates is the same as in a sequential update, because 1,250 pages in which the final records are written are relocated to the original pages by scanning the PMT.

The time required for a random update is 881 sec, but a PRL update takes 54 sec in a 100-page buffer, as shown in Fig. 4 (b). The time required for a PRL update is 4 sec, which is a little faster than sequential update time since 10,000 records are updated through the layers from the query processor to the storage manager of the DBMS in a sequential update, but only 1,250 pages are updated in the storage manager of the DBMS in a PRL. The experiment results show that the sum of the PRM and PRL update times is much less than the random update time. In reality, the update pattern in a DBMS is random, as shown in Fig. 5 (b). The reason for this is that the record update in the DBMS causes another page update, such as a meta-page, index pages, and so on. Thus, the data update performance can be improved when applying the PRM/PRL algorithm to the storage manager of a mobile DBMS in which flash memory is used as the storage media for a mobile device. With the exception of the random access pattern in Fig. 5 (b), other update sequences show a sequential access pattern. In Fig. 5, x-axis means the update page sequence and y-axis shows the page ID.

An additional experiment was carried out to compare the data update performance of SQLite, which is most popular product in a mobile DBMS. The environment used for the experiment is the same as in the previous test, except for a record size of 300 bytes and a buffer size of 1,000 pages. In Fig. 6, the experimental results show a similar update performance as in SQLite and FUNs. However, the excellence of the PRM/PRM policy of FUNs is shown through its 22 sec update time, whereas the random update time in the SQLite is 366 sec.

The experiments described thus far were conducted to evaluate the update performance by creating a virtual record set. Now, real navigation map update data are used to evaluate the update performance in this experiment. All old map data are stored in the FUNs database, which is about 1 GB in size, and the update (insert/delete/change) object set is updated to this database. The update data are road network data from the Gwangju and Dongtan areas, as shown in Fig. 7. The update map objects of each region are marked in red. These map data are updated in the NODE and LINK tables and cause other update propagation by the network.
relationship.

Figure 8 shows the page update patterns when updating each area map data to the FUNs database. In this figure, x-axis means the update page sequence and y-axis shows the page ID. Figures 8 (a) and 8 (b) show the general page update pattern. In general, the update sequence is random in the DBMS. The numbers of pages to be updated by the map data are 123 and 138, respectively, although the pages are updated 131 and 157 times in a general update. Additionally, these numbers of updates are the same as in the PRM, by which the general sequence is converted into sequential sequence as shown in Figs. 8 (b) and 8 (c).

In a PRL update, 123 and 138 updated pages are re-located to their original location, and the update page pattern is partially sequential as shown in Figs. 8 (e) and 8 (f). The page update time is shown Fig. 9 (b). The general update time of the Gwangju area data is 8.74 sec, whereas the PRM/PRL update time is 4.93 sec. For the Dongtan area data, the general update time is 11.1 sec, with 4.88 sec for the PRM/PRL update. As a result, this page management policy shows about a two-fold improvement in the update performance.

4. Conclusion

In this letter, we proposed a flash-aware page management policy to enhance the performance of frequent random updates in a mobile DBMS in which flash memory is used as the storage media. Logical page re-mapping and re-locating algorithms are implemented in the storage manager of FUNs, which is a navigation-specialized mobile DBMS providing incremental map updates. We used virtual data as well as real navigation map data to evaluate the update performance. As a result, this flash-aware mobile DBMS can provide about a two-fold improvement in the data update performance for a mobile device.

Acknowledgments

This work was supported by the Industrial Strategic Technology Development Program (10035250, Development of Spatial Awareness and Autonomous Driving Technology for Automatic Valet Parking) funded by the Ministry of Knowledge Economy (MKE), Rep. of Korea.

References

[1] S.W. Lee, G.J. Na, J.M. Kim, et al., “Research issues in next generations DBMS for mobile platforms,” Proc. Mobile HCI ’07, pp.9–12, 2007.
[2] E. Gal and S. Toledo, “Algorithms and data structures for flash memories,” ACM Comput. Surv., vol.37, no.2, June 2005.
[3] J.S. Kim, J.M. Kim, S.H. Noh, et al., “A space-efficient flash translation layer for compact flash systems,” IEEE Trans. Consum. Electron., vol.48, no.2, pp.365–375, 2002.
[4] S.W. Lee, D.J. Park, D.H. Lee, et al., “A log buffer-based flash translation layer using fully-associative sector translation,” ACM Trans. Embedded Computing Systems, vol.6, no.3, 2007.
[5] S. Lee et al., “Locality-aware sector translation for NAND flash memory-based storage systems,” SIGOPS Oper. Syst. Rev., vol.62, no.6, pp.36–42, 2008.
[6] S.W. Lee and B.K. Moon, “Design of flash-based DBMS: An in-page logging approach,” Proc. International Conference on Management of Data (SIGMOD), pp.55–66, Aug. 2007.
[7] K.W. Min, K.W. An, J.J. Jang, et al., “A system framework for map air update navigation service,” ETRI Journal, vol.33, no.4, pp.476–486, Aug. 2011.
[8] K.W. Min, K.W. An, J.W. Kim, et al., “The mobile spatial DBMS for the partial map air update in the navigation,” Proc. ITSC 2008, pp.476–481, Oct. 2008.