In-Page Logging B-Tree for Flash Memory*

Gap-Joo Na[1], Bongki Moon[2], and Sang-Won Lee[1]

[1] Sungkyunkwan University, Suwon, Korea
{factory,wonlee}@ece.skku.edu
[2] University of Arizona, Tucson, AZ, U.S.A.
bkmoon@cs.arizona.edu

Abstract. We demonstrate the IPL B$^+$-tree prototype, which has been designed as a flash-aware index structure by adopting the *in-page logging (IPL)* scheme. The IPL scheme has been proposed to improve the overall write performance of flash memory database systems by avoiding costly erase operations that would be caused by small random write requests common in database workloads. The goal of this demonstration is to provide a proof-of-concept for IPL scheme as a viable and effective solution to flash memory database systems.

1 Introduction

Since NAND flash memory was invented as a sector addressable non-volatile storage medium about two decades ago, its density has increased approximately twice annually and the trend is expected to continue until year 2012 [1]. Due to its superiority such as low access latency and low energy consumption, flash-based storage devices are now considered to have tremendous potential as an alternative storage medium that can replace magnetic disk drives.

On the other hand, due to the erase-before-write limitation of flash memory, updating even a single record in a page results in invalidating the current page containing the record and writing a new version of the page into an already-erased area in flash memory. This leads to frequent write and erase operations. In order to avoid this, we have proposed the in-page logging (IPL) scheme that allows the changes made to a page to be written (or *logged*) in the database, instead of writing the page in its entirety [2]. Since flash memory comes with no mechanical component, there is no compelling reason to write log records sequentially as long as it does not cause extra erase operations. Therefore, under the in-page logging approach, a data page and its log records are co-located in the same physical location of flash memory, specifically, in the same erase unit. Since we only need to access the previous data page and its log records stored in the same erase unit, the current version of the page can be recreated efficiently

* This work was partly supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2008-0641) and MKE, Korea under ITRC IITA-2008-(C1090-0801-0046). This work was also sponsored in part by the U.S. National Science Foundation Grant IIS-0848503. The authors assume all responsibility for the contents of the paper.

X. Zhou et al. (Eds.): DASFAA 2009, LNCS 5463, pp. 755–758, 2009.

under this approach. Consequently, the IPL approach can improve the overall write performance considerably. The IPL scheme uses physiological log records primarily for improving write performance, but the log records can also be used to realize a lean recovery mechanism [2].

The goal of this work is to demonstrate that IPL is a viable and effective solution to flash memory database systems that enables them to deliver high performance promised by the desired properties of flash memory. We have designed a new index structure called IPL B$^+$-tree as a variant of B$^+$-tree for computing platforms equipped with flash memory as stable storage. We have implemented the IPL B$^+$-tree on a development circuit board running Linux 2.6.8.1 kernel.

In addition to providing the proof-of-concept implementation of the IPL scheme, this demonstration will showcase (1) the design of IPL B$^+$-tree for flash memory database systems, (2) the IPL B$^+$-tree engine that implements the in-page logging mechanism for B$^+$-tree indexes, (3) the development platform that allows the IPL storage manager to be in full control of address mapping for flash memory.

2 System Description

The traditional B$^+$-tree is designed for disk-based storage systems, and yields poor write performance with flash-based storage systems. This section presents the design and implementation details of the IPL B$^+$-tree.

2.1 Design of IPL B$^+$-tree

In an IPL B$^+$-tree, as is illustrated in Figure 1, the in-memory copy of a tree node can be associated with a small in-memory log sector. When an insertion or a deletion operation is performed on a tree node, the in-memory copy of the tree node is updated just as done by traditional B$^+$-tree indexes. In addition, a physiological log record is added to the in-memory log sector associated with the tree node. An in-memory log sector is allocated on demand when a tree node becomes dirty, and is released when the log records are written to a log sector in flash memory.

The log records in an in-memory log sector are written to flash memory when the in-memory log sector becomes full or when the corresponding dirty tree node is evicted from the buffer pool. When a dirty tree node is evicted, it is not necessary to write the content of the dirty tree node back to flash memory, because all of its updates are saved in the form of log records in flash memory. Thus, the previous version of the tree node remains intact in flash memory, but is augmented with the update log records.

When an in-memory log sector is to be flushed to flash memory, its content is written to a flash log sector in the erase unit which its corresponding tree node belongs to. To support this operation, each erase unit (or a physical block) of flash memory is divided into two segments – one for tree nodes and the other for log sectors, as shown at the bottom of Figure 1.

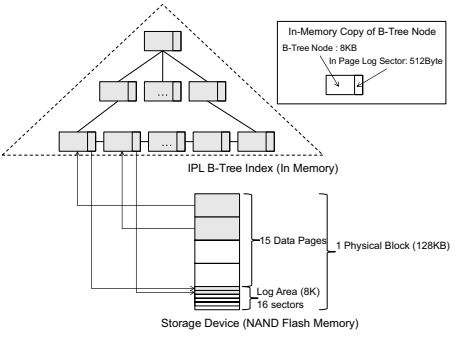

Fig. 1. IPL B$^+$-tree structure

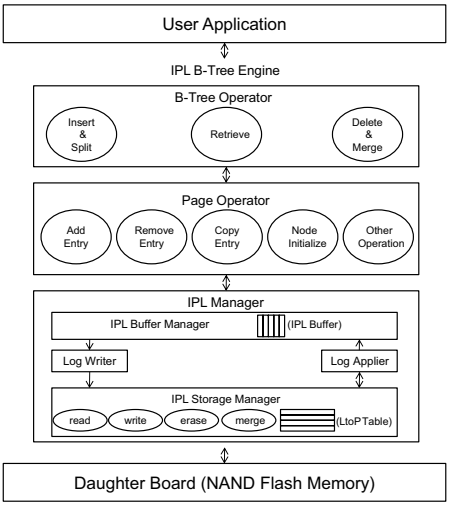

Fig. 2. IPL B$^+$-tree Engine

2.2 IPL B$^+$-tree Engine

Figure 2 shows the IPL B$^+$-tree engine that consists of three components: (1) B$^+$-tree operator module, (2) page operator module, and (3) IPL manager. The B$^+$-tree operator module processes traditional B$^+$-tree operations such as insertion, deletion and search. If a single operation involves more than a tree node, this is divided into multiple single-node requests, each of which is processed by the page operator module. For each single-node request, the page operator module adds its physiological log record to the corresponding in-memory log sector.

The most important component of the IPL B$^+$-tree engine is the IPL manager that provides the in-page logging mechanism for B$^+$-tree indexes. The IPL manager consists of four internal components: (1) IPL buffer manager, (2) IPL storage manager, (3) Log writer, and (4) Log applier. The IPL buffer manager maintains in-memory tree nodes and their corresponding in-memory log sectors in an LRU buffer pool. When an in-memory log sector becomes full and needs to be flushed to flash memory, the IPL buffer manager determines whether the log area of the corresponding erase unit in flash memory can accommodate the in-memory log sector. If it does, the in-memory log sector is written to flash memory. Otherwise, a merge request is sent to the IPL storage manager. Then, the Log applier computes the current version of tree nodes by applying the log records to the previous version of tree nodes. Since the entire tree nodes in the merged erase unit are relocated to a physically different region in flash memory, the logical-to-physical mapping is updated by the IPL storage manager, when a merge is complete.

When a tree node is to be read from flash memory, the Log applier sends a read request to the IPL storage manager, which returns an in-flash copy of the tree node along with its log records. Then, the Log applier creates the current version of the tree node by applying its log records to the tree node.

3 Demonstration Scenario

The demonstration will be set up with a host system and a target system as shown in Figure 3. The IPL B^+-tree runs on the target system that consists of an EDB9315A processor board and a flash daughter board [3].[1] The requests for B^+-tree operations are submitted from the host system, and the progress and performance of B^+-tree operations executed on the target system can be monitored on the host system.

In order to graphically illustrate the progress and performance of the B^+-tree operations to be demonstrated, we have implemented a performance monitor with GUI that runs on the Labview environment, as is shown in Figure 4. For more extensive performance comparison, a standard B^+-tree running on a comparable computing platform with a magnetic disk will also be available in the demonstration.

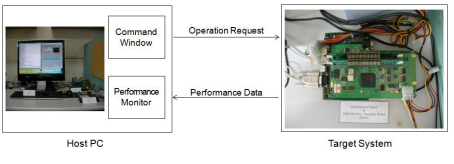

Fig. 3. Demonstration System Setup

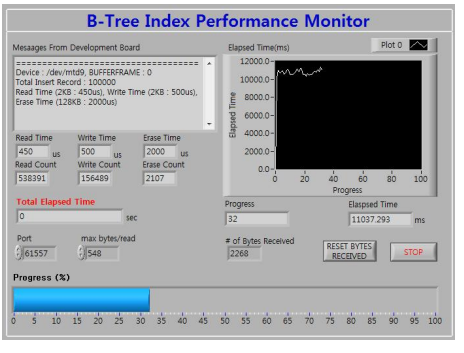

Fig. 4. Index Performance Monitor

References

1. Hwang, C.-G.: Nanotechnology Enables a New Memory Growth Model. Proceedings of the IEEE 91(11), 1765–1771 (2003)
2. Lee, S.-W., Moon, B.: Design of Flash-Based DBMS: An In-Page Logging Approach. In: Proceedings of the ACM SIGMOD, Beijing, China, pp. 55–66 (June 2007)
3. Cirrus Logic. EP9315 Data Sheet - Enhanced Universal Platform System-on-Chip Processor (March 2005) DS638PP4

[1] Most of the flash memory SSD products come with an on-device controller that runs a flash translation layer to deal with address mapping as well as command interpretation. Consequently, the logical-to-physical mapping is completely hidden from outside the flash memory storage devices. That is the reason why we have chosen an EDB9315 processor board as a hardware testbed.