DOCTORAL THESIS

Indexing and query processing for flash-memory based database systems

Li, Yu

Date of Award:
2012

Indexing and Query Processing for Flash-Memory Based Database Systems

LI Yu

A thesis submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Principal Supervisor: Dr. Jianliang XU

Hong Kong Baptist University

Jan 2012

Abstract

NAND flash-memory based storage has been widely used in embedded and mobile systems in the past decade. Recently it also becomes popular in personal computers and even in enterprise computing infrastructures in the form of Solid-State Drive (SSD). With its continuously increasing capacity and dropping price, we envision that some database systems will operate on flash-memory based storage devices in the near future.

Comparing to conventional magnetic disks, flash memory has a number of unique I/O characteristics. In particular, flash memory has an excellent random read performance, which is as fast as the sequential read. On the other hand, the random write of flash memory is not only much slower than the sequential write in speed, but also may cause other issues such as reducing the endurance of flash memory. Furthermore, such performance characteristics may vary for different types of flash memory devices. As a result, the state-of-the-art database algorithms, which assumed I/O characteristics of magnetic disks, become suboptimal when implemented on flash-memory based storage devices.

In this thesis, we investigate how to optimize indexing and query processing techniques for databases running on flash-memory based storage devices. With a careful study of the flash I/O characteristics, we re-examine the data structures and algorithms involved in the imple-

mentation of database management systems. We identify several database components with either performance issues or potential for achieving higher performance. We then optimize them by exploiting the unique characteristics of flash memory. More specifically, firstly, we study the indexing structure. The features of flash memory, such as the erase-before-write constraint and the asymmetric read/write cost, severely deteriorate the performance of the traditional B^+-tree algorithm. We propose a new *lazy-update* strategy for B^+-tree to overcome the limitations of flash memory. The idea is to defer the time of committing update requests to the B^+-tree by buffering them in a segment of main memory. They are later committed in groups so that each write operation can be amortized by a bunch of update requests. We identify a victim selection problem for the *lazy-update* B^+-tree and develop two heuristic-based commit policies to address this problem. Experiment results show that the proposed *lazy-update* method, along with a well designed commit policy, greatly improves the update performance of the traditional B^+-tree while preserving the query efficiency.

Secondly, we develop a novel technique called *StableBuffer* to optimize the random write performance for write intensive database applications. As motivated by a recently discovered *focused write pattern*, we propose to write pages temporarily to a small, pre-allocated storage space on the flash device, i.e., *StableBuffer*, instead of directly writing to their actual destinations. We then recognize and flush efficient write patterns of the buffer to achieve a better write performance. In contrast to prior log-based techniques, the StableBuffer solution does not require modifying the driver of flash memory devices and hence is device-independent. We discuss the detailed design and implementation of the StableBuffer solution. Experiment results based on a TPC-C benchmark trace shows that StableBuffer significantly improves the response time and throughput of write operations in comparison with a direct write-through

strategy.

Finally, we study the core of query processing — join processing — on flash-memory based storage devices. We propose a new framework called *DigestJoin* to optimize the join performance by reducing the intermediate result size and exploiting fast random reads of flash memory. DigestJoin consists of two phases: (1) projecting the join attributes followed by a join on the projected attributes; and (2) fetching the full tuples that satisfy the join to produce the final join results. While the problem of tuple/page-fetching with the minimum I/O cost (in the second phase) is intractable, we propose three heuristic page-fetching strategies for flash memory. We implement DigestJoin and conduct extensive experiments on a real flash memory device. Experiment results based on TPC-H datasets show that DigestJoin clearly outperforms the traditional sort-merge join and hash join under a wide range of system configurations.

Table of Contents