

DOCTORAL THESIS

Indexing and query processing for flash-memory based database systems

Li, Yu

Date of Award:
2012

[Link to publication](#)

General rights

Copyright and intellectual property rights for the publications made accessible in HKBU Scholars are retained by the authors and/or other copyright owners. In addition to the restrictions prescribed by the Copyright Ordinance of Hong Kong, all users and readers must also observe the following terms of use:

- Users may download and print one copy of any publication from HKBU Scholars for the purpose of private study or research
- Users cannot further distribute the material or use it for any profit-making activity or commercial gain
- To share publications in HKBU Scholars with others, users are welcome to freely distribute the permanent URL assigned to the publication

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

Declaration	i
Abstract	ii
Acknowledgements	v
Table of Contents	vi
List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Flash Memory Technology	1
1.2 I/O Features of Flash Memory	4
1.3 The Impact on Data Management	5
1.4 Thesis Organization and Contributions	6
2 Background and Related Works	8
2.1 Background	8

2.1.1	Flash Memory	8
2.1.2	Dynamic Mapping	10
2.1.3	Flash-based Storage Devices	14
2.1.4	I/O Properties of Flash-based Storage Devices	17
2.2	Related Works	19
2.2.1	Optimizing Indexing Performance	21
2.2.2	Optimizing Page Write Performance	22
2.2.3	Optimizing Query Processing	23
3	Lazy-Update B⁺-Tree	26
3.1	Motivation	26
3.2	Background	28
3.3	Overview of Lazy-Update B ⁺ -Tree	30
3.4	Victim Selection Policies	35
3.4.1	The Victim Selection Problem	35
3.4.2	Biggest Size Policy	36
3.4.3	Cost-based Policy	37
3.5	Implementation	40
3.5.1	Data Structure	40
3.5.2	Manipulation Algorithms	41
3.5.3	Super Group	44
3.6	Performance Evaluation	48
3.6.1	Simulation Setup	48

3.6.2	Overall Performance	50
3.6.3	Effect of Lazy-Update Pool Size	51
3.6.4	Effect of B ⁺ -Tree Order	53
3.6.5	Effect of Grouping Optimization	54
3.6.6	Effect of Super Group Forming	55
3.6.7	Effect of Flash I/O Performance	56
3.7	Summary	57
4	StableBuffer	58
4.1	Motivation	58
4.2	Overview	61
4.2.1	StableBuffer	61
4.2.2	Data Structures	63
4.2.3	Page Operations	65
4.3	Page Management	67
4.3.1	Write Pattern Recognition	67
4.3.2	Page Flushing	71
4.3.3	Combining Write-Pattern Recognition and Page Flushing	73
4.4	Performance Evaluation	74
4.4.1	Experiment Setup	74
4.4.2	Comparison With Direct Writes	75
4.4.3	The Impact of StableBuffer Size	79
4.5	Summary	81

5	DigestJoin	83
5.1	Motivation	83
5.2	Overview	88
5.3	Page Fetching Strategies	92
5.3.1	The Page Fetching Problem	92
5.3.2	Naive Fetching Strategy	95
5.3.3	Page-based Fetching Strategy	96
5.3.4	Graph-based Fetching Strategy	98
5.4	Cost Analysis	104
5.5	Performance Evaluation	111
5.5.1	Experiment Setup	111
5.5.2	Segment Selection in Graph-based Page Fetching	114
5.5.3	Impact of Join Result Size	114
5.5.4	Impact of Page Size	118
5.5.5	Impact of Memory Buffer Size	119
5.5.6	Impact of Result Output Size	120
5.5.7	Impact of Digest Entry Size	121
5.5.8	Impact of Join Result Distribution	122
5.5.9	Impact of Magnetic Disks and Flash-based Storage	124
5.6	Summary	125
6	Conclusions and Future Work	126
6.1	Conclusions	126

6.2 Future Work 128

Curriculum Vitae **138**