Journal of Digital Contents Society Vol. 14 No. 1 Mar. 2013(pp. 15-23)
http://dx.doi.org/10.9728/dcs.2013.14.1.15

소형 블록 DBMS의 데이터/인덱스 페이지 구조 소형화를 통한 NVRAM 성능 개선

배상희*, 이태화**, 차재혁***

요 약

저전력과 새로운 응용의 대용량 데이터 처리 요구에 따라 저장장치로 하드디스크 대신 빠른 입출력 성능을 가진 SSD(Solid State Disk/Drive)를 활용한 저장 시스템이 등장하고 있으며 다양한 처리 데이터 단위와 out-place-update, 제한된 지우기 횟수 등의 SSD 고유의 문제점을 극복하는 방안에 대한 연구가 활발하다. 그러나 빈번한 임의 쓰기를 발생하는 소규모 특정 데이터를 하드디스크나 SSD에 저장하는 경우 성능 및 안정성 저하 문제는 아직 완전히 해결하지 못하고 있다. 본 논문에서는 NVRAM의 바이트 단위의 빠른 읽기/쓰기와 비휘발성 그리고 인덱스 페이지 내 실제 데이터 변경 크기가 블록 크기보다 작다는 특성을 활용하여 빈번한 임의 쓰기를 요구하는 인덱스를 바이트 접근이 가능한 NVRAM에 저장하는 시스템의 구조를 제안한다.

키워드 : 플래시 메모리, 데이터베이스, 인덱스, 비휘발성 메모리

Scaling down data/index page structure of the NVRAM based DBMS with the small size blocks

Sang-Hee Bae*, Taehwa Lee**, Jaehyuk Cha***

Abstract

In response to the demands of large-scale data processing with low-power and new application, a storage system using SSD (Solid State Disk/Drive) with fast input-output performance instead of hard disc has appeared as storage device. Studies on methods to overcome specific problems of SSD such as various processing data units, out-place-update and limited delete count have been actively conducted. However, declining performance and stability have not been resolved yet when storing case specific data with small scale that causes frequent random write in hard disc or SSD. This thesis suggests a system structure that stores index requesting frequent random write in NVRAM capable of byte access by using characteristics such as byte unit fast read / write of NVRAM, non-volatile and smaller size of actual changed data size in index page than block size.

Keywords : flashmemory, DBMS, index, NVRAM

1. 서론

1.1 연구 배경 및 목적

최근 현재 양산중인 메모리들의 근본적인 문제를 해결하기 위하여 기존의 NAND플래시나 NOR플래시 메모리를 대체할 차세대 비휘발성 메모리(Next Generation Non-volatile Memory)에 대한 연구가 진행되고 있다. NVRAM (Nonvolatile RAM)은 기존 RAM과 동일한 성질을 보이면서 디스크와 같이 비휘발성인 차세대 메모리이다. 최근 반도체 기술의 급격한 발전으

※ 교신저자(Corresponding Author): Jaehyuk Cha
접수일:2013년 01월 15일, 수정일:2013년 01월 19일
완료일:2013년 02월 27일
* KT 사원 chopa84@naver.com
** 한양대학교 컴퓨터소프트웨어학과 박사과정
email: alghost@hanyang.ac.kr
*** 한양대학교 컴퓨터공학 교수
Tel: +82-2-2220-1158, Fax: +82-2-2220-4459
email: chajh@hanyang.ac.kr
▣ 본 연구는 한국연구재단 정부지원연구(2010-0013076)지원으로 수행되었음.

로 인하여 FeRAM, MRAM, PRAM등 다양한 형태의 NVRAM이 개발되고 있다[1,2]. 이에 따라 운영체제의 메모리 계층 구조에 대한 변화와 기존 구조상에서 NVRAM을 활용하는 새로운 가능성에 대한 연구가 활발히 진행되고 있다[3-5]. FeRAM, MRAM, PRAM 등의 NVRAM은 NAND 플래시 메모리와 다르게 바이트 단위의 빠른 읽기/쓰기와 in-place-update가 가능하나 블록 단위의 읽기/쓰기 성능이 그리 뛰어나지 못한 한계점을 지니고 있다.

본 논문에서는 NVRAM의 비휘발성, 바이트 단위 빠른 읽기/쓰기 성능, 저장 장치에서 플러시(flush)되는 인덱스 페이지 내 실제 인덱스 변경 크기가 페이지 크기보다 훨씬 작다는 특성을 활용하여 빈번한 임의 쓰기가 발생하는 인덱스를 바이트 접근이 가능한 NVRAM에 저장하는 시스템의 구조를 제안한다.

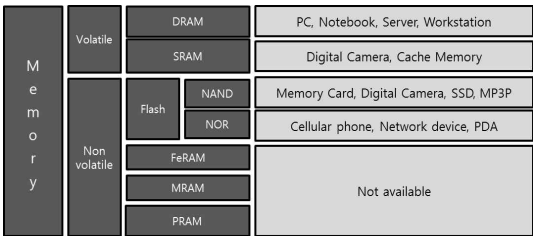
성능 평가를 위해 최근 임베디드 데이터베이스 시스템 라이브러리로 널리 사용되는 오픈 소스 기반의 SQLite[6]을 사용하였고, RL benchmark Trace등을 사용하여 실험 하였다. 실험 결과 DBMS와 저장장치에 따라 최적화된 인덱스 페이지가 다른 것을 확인할 수 있었고 NVRAM 시스템 에서는 다른 시스템과 다르게 인덱스 페이지가 작을 때 더 좋은 성능을 보인다.

2. 배경지식

2.1 휘발성 메모리

반도체는 크게 메모리 반도체와 비메모리 반도체로 분류할 수 있다. 비메모리 반도체는 우리가 흔히 CPU(중앙처리장치)라 부르는 마이크로프로세서 외에 많이 사용되는 비메모리 반도체는 ASSP(표준반도체)와 ASIC(주문형반도체)등이 있다.

(그림 1) 반도체의 분류



(Figure 1) Categorization of semiconductors

(그림 1)은 반도체의 분류를 나타낸 것이다. 휘발성 메모리로는 DRAM과 SRAM이 있고 비휘발성 메모리로는 NAND 플래시, NOR 플래시와 차세대 비휘발성 메모리인 FeRAM, MRAM, PRAM등이 있다.

2.2 플래시 메모리

플래시 메모리는 EEPROM, 디스크 등과 같이 전원이 꺼져도 기록된 데이터가 지워지지 않는 비휘발성 저장장치이다. 기계적인 장치를 사용하는 기존의 디스크와 달리 전기적인 신호에 의해 동작하기 때문에 기계적인 지연이 없다. <표 1>은 DRAM, 플래시 메모리, 그리고 하드 디스크의 성능을 보여주고 있다. <표 1>에서 알 수 있듯이 지연시간에서 NAND 플래시 메모리는 하드 디스크에 비해 읽기의 경우 약 344배, 쓰기의 경우 약 47배의 뛰어난 성능을 지니며 이러한 빠른 연산속도로 인해 디스크를 대체할 수 있는 저장장치로 주목 받기 시작하였고 작은 크기와 강한 내구성, 저 전력 등의 장점으로 이미 임베디드 분야에서 플래시메모리는 하드디스크 보다 우위를 차지하고 있다.

<표 1> 다양한 기억장치 간의 성능비교

Device	Performance		
	Read	Write	Erase
DRAM	60ns(2B) 2.6us(512B)	60ns(2B) 2.6us(512B)	N/A
NOR	150ns(1B) 15us(512B)	211us(2B) 3.5ms(512B)	1.2s(1 28KB)
NAND	10us(1B) 36us(512B)	226us(2B) 266us(512B)	2ms(1 6KB)
HDD	12.4ms(512B)	12.4ms(512B)	N/A

<Table 1>Performance of Storage Devices

2.3 차세대 비휘발성 메모리

최근 NAND 플래시나 NOR 플래시 메모리를 대체할 차세대 비휘발성 메모리인 PRAM, MRAM, FeRAM 등 나노소자 기반의 차세대 비휘발성 저장 소자에 대한 연구 투자가 매우 활발하게 진행되고 있으며, 국내에서도 정부의 전폭적인 연구투자에 힘입어 나노소자 기반의 비휘발성 메모리 기술이 빠르게 발전하고 있다. 차세대 비휘발성 메모리의 가장 큰 특징은 비휘발성이면서도 빠른 바이트 별 무작위 접근(random access)이 가능하고 쓰기 전 소거 동작 없이 직접 갱신(in-place update)이 가능하다는 것이다. 차세대 비휘발성 메모리의 개발은 저장장치를 포함하는 메모리 시스템 설계에 있어서 빠른 부팅 및 응용의 실행 / 종료뿐 아니라 현재 대부분의 시스템에서 다양한 요구를 만족시키기 위해 다양한 종류의 메모리가 사용되는 것을 하나의 차세대 메모리로 통합하여 사용할 수 있게 하여 혁신적인 메모리 시스템을 설계할 수 있다는 기대감을 가져오고 있다. 현재 차세대 비휘발성 메모리라는 이름으로 많은 종류의 메모리들이 연구되고 있다. <표 2>는 차세대 비휘발성 메모리들의 특성을 기존 플래시 메모리와 비교한 결과를 보여준다[9].

<표 2> 각 메모리의 비교

	MRAM	FeRAM	PRAM	SRAM	DRAM	FLASH
Volatile	O	O	O	X	X	O
Scalable	O	△	O	X	O	O
Write Time	10~50 ns	30~100ms	100ns ~	30~70ns	50ns	10,000ns
Read Time	10~50 ns	30~100ns	20~80 ns	30~70ns	50ns	50ns
Re-Write Time	10^{16}	10^{12}~ 10^{16}	10^{12}	10^{15}	10^{15}	10^{6}
Data keeping	10 years	10 years	10 years	0.1s	0.1s	10 years
Read method	Not Destroy	Destroy	Not Destory	Not Destroy	Destory	Not Destory
Power consumption	~30uW	~10uW	~30uW	300mW	300mW	30mmW
Standby power	~1uA	~1uA	~1uA	100uA~	100uA~	~1uA

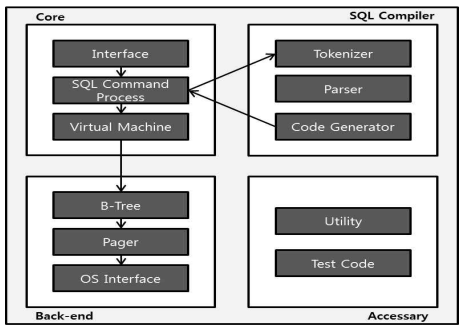
<Table 2> Comparison of memory

현재 가장 주목 받고 있는 차세대 비휘발성 메모리는 MRAM, FeRAM 그리고 PRAM이다. MRAM, PRAM은 플래시(flash memory)로부터 파생한 것이며, FeRAM은 DRAM으로부터 파생한 것으로 볼 수 있다.

2.4 SQLite 구조

SQLite는 오픈 소스의 임베디드 SQL 데이터베이스 엔진으로 별도의 서버 구동 없이 동작이 가능한 임베디드 기반의 데이터베이스 관리 시스템이다. 구글의 안드로이드나 애플의 아이폰 등에서 사용되는 솔루션으로 사용자 측면과 관리자 측면에서 모두 뛰어난 접근성을 가지고 있다. 따라서 데이터 관리에 중요한 임무를 담당하는 SQLite는 시간이 지날수록 보급이 확대될 것으로 예상된다. SQLite의 내부구조는 (그림 2)와 같다. 코어, SQL 컴파일러, 백 엔드가 주요 핵심 요소이고, 이외에 액세서리 역할을 하는 유틸리티와 테스트 코드로 이뤄져 있다.

(그림 2) SQLite 구조

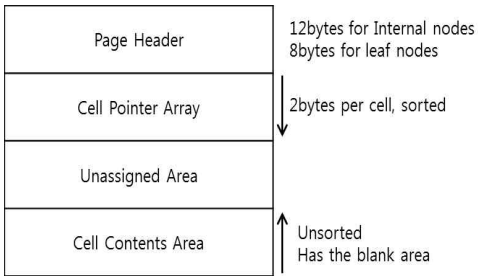


(Figure 2) Structure of SQLite

2.5 페이지 구조

각 데이터베이스 파일은 고정된 크기의 페이지들로 나뉘어져 있다. B+-트리 모듈은 모든 페이지들을 관리 한다. 각 페이지는 트리 페이지(내부 또는 리프), 오버플로우 페이지, 프리 페이지로 되어 있다. SQLite 트리 페이지는 셀로 분리되어 있다. 셀은 트리 페이지에서 공간 할당과 해체를 위한 단위이다. 트리 페이지의 구조 는 (그림 3)과 같다.

(그림 3) 트리 페이지 구조



(Figure 3) Structure of Tree Page

페이지 헤더는 페이지를 관리하기 위한 정보를 포함하고 있다. 헤더는 항상 페이지의 시작주소에 저장된다. (페이지 1의 경우는 페이지 헤더 전에 100바이트 파일 헤더를 포함하고 있다. 파일 헤더는 오직 페이지 1에만 포함되어 있다.) 페이지 헤더는 <표 3>의 구조로 구성되어 있다.

셀(cell)이라는 것은 테이블 페이지일 경우 하나의 레코드가 된다. 레코드에 레코드 정보와 열 식별자(row id)를 합쳐서 하나의 셀이 만들어 진다. 인덱스 페이지의 경우 키 값과 키에 해당하는 데이터가 있는 열 식별자를 합쳐서 셀이 만들어 진다. 그러나 B-트리 모듈에서는 현재 페이지가 테이블인지 인덱스인지 상관없기 때문에 키/데이터 페어로 이루어진 셀이 주어지면 키 순서에 맞게 저장하거나 찾아준다.

<표 3> 트리 페이지 헤더 구조

offset	size	Description
0	1	flag 1: intkey, 2: zerodata 4: leafdata, 8: leaf
1	2	byte offset of first free block
3	2	# of cells of this page
5	2	first byte of cell content area
7	1	# of bytes of fragmentation
8	4	right child node(the Ptr(n) value)

<Table 3> Header Structure of Tree Page

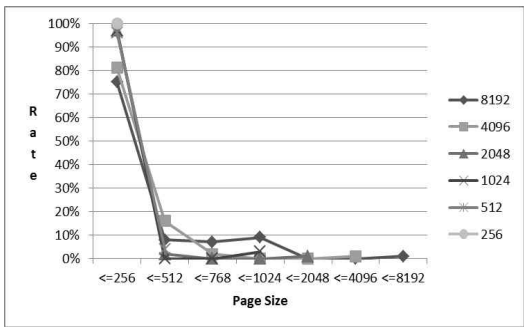
3. NVRAM기반 블록 저장장치를 활용하기 위한 시스템 구조

본 논문에서는 보조 저장장치로 NVRAM을 사용하는 시스템에 대해서 알아본다. 이 시스템 구조는 NVRAM의 특징인 바이트 단위 읽기/쓰기가 가능하고, 비휘발성인 장점을 가지고 있다. 3.1에서 실험결과 SQLite에서 인덱스 페이지 내 실제 데이터 변경 크기가 블록 크기보다 훨씬 작다는 특성을 활용할 수 있다.

3.1 DBMS에서 NVRAM 활용 가능성 실험

일반적으로 사용하는 DBMS에서는 B-트리와 B+-트리를 이용하여 인덱스를 관리한다. B-트리 특성상 하나의 엔트리 키 값이 삽입되거나 변경되면 그에 따른 부모 및 형제 노드에 키 값이 변경되기 때문에 실제 데이터가 변경되지 않는 부분도 쓰기 작업을 수행하게 된다. 이 논문에서 실제 실험은 모두 SQLite로 이루어지기 때문에 SQLite에서 페이지 크기에 따른 실제 인덱스 변경 개수와 비율을 측정해 보았다. 실험환경은 Ubuntu Linux 10.04에서 RL 벤치마크 트레이스 중 인덱스 관련부분을 이용하여 실험 하였다. 인덱스 파일에 대한 쓰기 작업이 발생할 때 데이터를 따로 저장한 후, 서로 데이터를 비교하여 실제 변경된 부분을 측정하였다. 페이지 차이를 비교할 때 페이지 크기를 기준으로 데이터를 나눈 후에 변경된 범위를 가지고 크기로 측정하였다. (그림 4)는 SQLite 페이지 크기에 따라 실제 변경된 개수와 비율을 측정한 표이다. 실제 인덱스 페이지 변경된 크기가 작기 때문에 소형페이지를 지원하도록 SQLite 페이지 핸들러를 변경하였다.

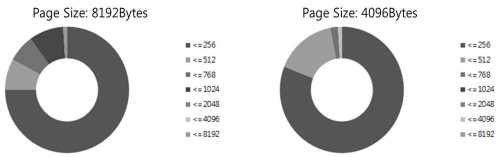
(그림 4) 변경된 인덱스 비율



(Figure 4) Changed Index Size

(그림 4)를 보면 페이지 크기에 상관없이 실제 인덱스가 변경된 크기는 대부분 512바이트 이하에서 이루어진 것을 알 수 있다. 또한, 한 페이지에서 인덱스가 변경된 비율도 대부분 256바이트 이하이다. (그림 5)는 페이지 크기가 8192바이트와 4096바이트 일 경우 실제 변경된 인덱스의 비율을 보기 쉽게 나타낸 것이다. 8192의 경우 약 75%, 4096의 경우 약80%정도가 256바이트 이하의 크기로 인덱스가 변경되는 것을 알 수 있다.

(그림 5) 변경된 인덱스의 크기

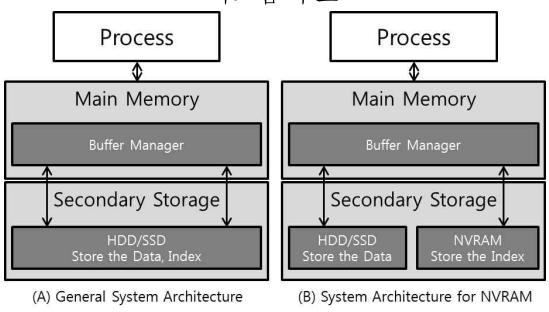


(Figure 5) Rate of changed index

이와 같이 페이지 내에 실제 업데이트 크기가 일반적인 논리 페이지 크기(8KB, 4KB)보다 훨씬 작게 측정 되었다. 본 논문에서는 참조 빈도가 높고 빈번한 임의 쓰기를 요구하는 인덱스 저장을 위해 바이트 단위의 빠른 읽기/쓰기가 가능한 NVRAM을 활용하는 시스템 구조에 효과에 대하여 실험을 진행하였다.

3.2 NVRAM이 보조 기억 장치로 사용하는 시스템

(그림 6) 일반적인 시스템 구조와 NVRAM 시스템 구조



(Figure 6) General vs NVRAM System Architecture

(그림 6-(A))는 일반적인 시스템 구조를 나타낸 그림이다. 대부분의 DBMS의 경우 이와 같은 구조를 사용하는데 데이터와 인덱스 저장을 같은 영역에 저장하기 때문에 빈번한 업데이트를 요구하는 인덱스는 성능의 병목이 되고 있다. 본 논문에서는 인덱스 저장을 위해서 NVRAM을 보조 기억 장치로 따로 사용하는 시스템을 기반으로 실험을 진행하였다. 해당 시스템 구조는 (그림 6-(B))와 같다. NVRAM의 경우는 바이트 단위의 빠른 읽기/쓰기가 가능하고 비휘발성인 특성을 가지고 있기 때문에 소형 단위로 빈번한 업데이트가 일어나는 경우에는 오히려 대형 블록 단위 보다 소형 블록 단위에서 더욱 효과적이다. 앞서 본 논문에서 언급한 바와 같이 인덱스 내 실제 업데이트 되는 크기는 8196바이트의 경우 80%이상이 512바이트 미만이었다. 또한 플래시 메모리와 다르게 NVRAM은 삭제 연산이 필요 없기 때문에 소형 블록 단위에서 더욱 효과를 발휘할 수 있다. 반면에 블록이 소형화 되면 노드 분할 횟수 증가 및 인덱스를 관리하는 B-트리의 최대 높이가 증가되는 문제점을 가지고 있어 오히려 성능이 저하되는 원인이 될 수 있다. 따라서 인덱스를 저장하기 위해서는 적절한 블록 크기, 불필요한 노드 구조 제거, 버퍼 개선 및 해싱 정책 개선 등이 필요하다.

4. 성능 평가

본 논문에서 실험은 Intel® CoreTM i5 CPU 750 @ 2.67GHz CPU와 DDR3 4.00GB RAM 컴퓨터에서 실험을 진행하였다. 운영체제는 Ubuntu 10.10 리눅스 커널 2.6.35 환경에서 SQLite 3.7.4 DBMS를 사용하였다. 또한 NVRAM실험을 위하여 바이트 접근이 가능한 램 디스크 메커니즘인 tmpfs에서 실험을 하였다. NVRAM중에 PRAM을 기준으로 실험을 진행하였다.

4.1 트레이스 분석

실험에서 활용한 트레이스는 SQLite 홈페이지에서 제공된 트레이스로 실제 안드로이드 기반 스마트폰에서 성능 측정 도구로 개발된 RL benchmark에서 사용하는 트레이스이다. 이 트레

이스는 JOIN에 대한 성능 측정이 빠져 있기 때문에 몇 가지 워크로드를 추가하였다. <표 4>은 20가지 워크로드를 표로 나타낸 것이고 인덱스 관련 부분은 색깔은 다르게 표시하였다.

<표 4> RL Benchmark 기반 트레이스

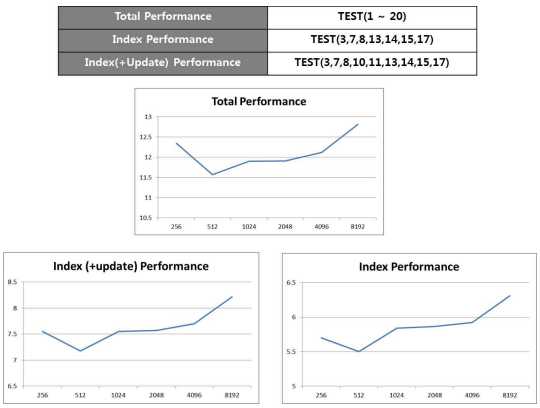
num	Workload
1	1000 INSERTs
2	25000 INSERTs in a transaction
3	25000 INSERTs into an indexed table
4	100 SELECTs without an index
5	100 SELECTs on a string comparison
6	INSERT JOIN without an index
7	Creating an index
8	5000 SELECTs with an index
9	1000 UPDATEs without an index
10	25000 UPDATEs with an index
11	25000 text UPDATEs with an index
12	INSERTs from a SELECT
13	INNER JOIN with index on one side
14	INNER JOIN on text field with index on one side
15	100 SELECTs with subqueries. Subquery is using an index
16	DELETE without an index
17	DELETE with an index
18	A big INSERT after a big DELETE
19	A big DELETE followed by many small INSERTs
20	DROP TABLE

<Table 4> Trace based on RL Benchmark

4.2 NVRAM 기반 실험 결과

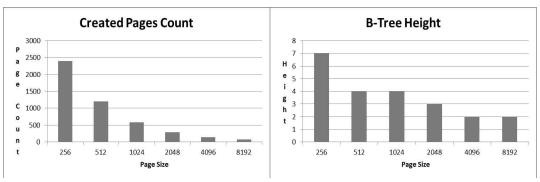
(그림 7)은 RL Benchmark 기반 트레이스의 전체 성능을 나타낸 것이다. 먼저 전체 TEST를 종합한 성능 테스트 결과를 보면 페이지 크기가 작을수록 성능이 좋아지다가 256byte 페이지 크기에서 성능이 안 좋아지는 것을 알 수 있다. 그 이유는 페이지 크기가 512byte ~ 8192byte의 경우 B-Tree의 높이가 2에서 4인 반면 256byte에서는 B-Tree의 높이가 7로 갑자기 증가한 것을 알 수 있다. 즉, 데이터와 페이지를 저장하기 위하여 256byte에서 많은 페이지가 필요하기 때문에 오히려 성능의 저하가 발생하게 되는 것이다. 반면에, 인덱스, 인덱스(업데이트 포함)의 경우 여전히 512byte가 가장 성능이 좋지만 256byte에서도 다른 페이지보다 성능이 좋은 것을 알 수 있다. 그 이유는 빈번한 업데이트가 발생하는 인덱스의 경우 실제 페이지 내에서 변경되는 크기가 대부분 256byte 크기 미만이기 때문에 오히려 페이지가 작을수록 성능이 향상되는 것이다. 추후 연구 과제로 NVRAM의 바이트 접근 특성을 이용하여 256byte와 512byte 사이의 최적의 페이지를 찾는 연구를 진행한다면 모든 면에서 완벽한 페이지를 찾을 수 있을 것이라고 생각한다. 512byte에 비해 256byte에서 성능이 좋지 않은 이유를 찾기 위하여 다음과 같은 여러 부분에서 실험을 하였다.

(그림 7) 인덱스 성능 실험 결과



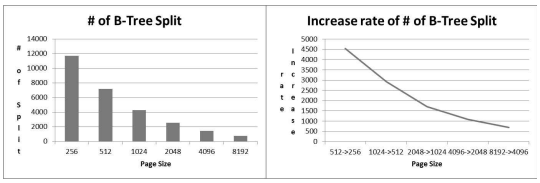
(Figure 7) Performance of Index

(그림 8) 페이지 생성 개수 및 B-Tree 높이



(Figure 8) Created Page Count & B-Tree Height

(그림 9) B-Tree 분할 횟수 및 증가율

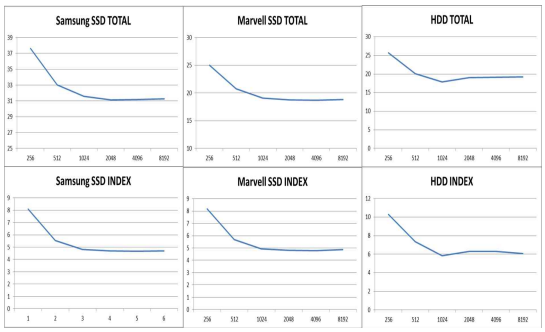


(Figure 9) and Increase rate of B-Tree Split

4.3 SSD, 하드디스크 성능 실험

이번 절에서는 NVRAM 이외의 저장장치에서 수정된 SQLite를 이용하여 RL Benchmark Trace를 가지고 실험을 하였다. 실험에 사용한 저장장치로는 WDC WD5000AAKS-00V1AO 500GB 하드디스크와 ReakSSD C300 128GB, Samsung MZ-5PA 128GB를 사용하였다.

(그림 10) SSD, HDD 성능 테스트



(Figure 10) Performance of SSD and HDD

이 논문에서 제안한 빈번한 업데이트가 발생하는 인덱스를 NVRAM을 저장장치로 사용한 구조의 경우 256byte, 512byte 페이지에서 가장 성능이 좋다는 것을 증명하기 위하여 NVRAM 이외의 저장장치에서 실험을 진행하였다. (그림 10)실험 결과처럼 SSD와 HDD의 경우 오히려 소형 페이지에서 성능이 나쁜 것을 알 수 있다. 그 이유는 플래시 메모리의 경우는 잦은 삭제 연산과 out-of-place 때문에 오히려 소형페이지에서 성능이 저하된다. HDD의 경우는 기계적인 지연으로 성능이 지연됨을 추측할 수 있다.

5. 수식 모델 정의 및 검증

이번 장에서는 본 논문에서 제안한 시스템 구조에서 인덱스 저장을 위해 NVRAM을 활용할 때의 성능 수식 모델을 정의한다. B-트리 I/O 기반으로 수식 모델을 정의하였으며 CPU캐시는 본 논문에서 고려하지 않는다.

5.1 B-트리 엔트리 검색 수식 모델 정의

NVRAM을 보조 기억 장치로 활용하는 구조에서 인덱스 노드가 메인 메모리의 버퍼에 있을 경우 버퍼히트가 되고, 이는 DRAM의 읽기 성능으로 노드 접근이 가능하다. 반면 인덱스 노드가 메인 메모리의 버퍼에 없을 경우 버퍼미스가 되고, 이는 보조 저장장치에서 DRAM에 노드를 저장하고 그 이후 DRAM의 읽기 성능으로 인덱스 노드에 접근하게 된다. B-트리에서 인덱스 노드의 검색작업은 최상위 노드에서 단말 노드까지의 검색을 수행하게 된다. 이는 트리의 높이에 매우 의존적이며 페이지 크기가 작아질수록 트리의 높이는 로그 스케일로 증가하게 된다. 이 논문에서 사용한 수식 모델 정의는 슬롯 자료구조를 활용하는 시스템에서 엔트리 검색 시 노드 내 이진 탐색을 사용하는 구조를 기반으로 수식을 정의하였다. 다음은 B-트리 수식 모델 기호를 정의한 표이다.

<표 5> B-트리 수식 모델 기호 정의

기호	정의	기호	정의	기호	정의
N	B-트리에 구축된 레코드 수	R	검색할 레코드 수	I	삽입할 레코드 수
ND	노드 크기	ET	엔트리 크기	MD	메타데이터 크기
KEY	KEY 크기	HR	HIT 비율	SL	슬롯 크기
Rdram	바이트 당 DRAM 읽기 시간	Wdram	바이트 당 DRAM 쓰기 시간	Rstorage	바이트 당 저장장치 읽기 시간
Wstorage	바이트 당 저장장치 쓰기 시간				

<Table 5> Definition of Symbol for B-Tree expression

다음 수식은 <표 5> 수식 모델 기호를 기반으로 B-트리 엔트리 검색 과정을 수식으로 정의하였다.

$$R * \log_{\frac{ND-MD}{ET+SL}} N * HR * R_{dram} * \log_2 (0.7 * \frac{ND-MD}{ET+SL}) * (SL+KEY)$$
$$+ R * \log_{\frac{ND-MD}{ET+SL}} N * (1-HR)$$
$$* (R_{storage} * ND + R_{dram} * \log_2 (0.7 * \frac{ND-MD}{ET+SL}) * (SL+KEY)$$

5.2 B-트리 엔트리 삽입 수식 모델 정의

B-트리에서 인덱스 노드의 삽입작업은 최상위 노드에서 단말 노드까지의 검색하는 과정과 해당 단말 노드에 엔트리를 삽입하는 과정으로 이루어진다. 단말 노드에 엔트리를 삽입하기 위하여 검색하는 과정은 B-트리 엔트리 검색 과정과 동일하며 엔트리 삽입 과정은 앞서 정의한 검색과정에서 추가해 주어야 한다. 일반적으로 슬롯 자료구조를 활용하는 시스템 구조에서는 단말 노드에 엔트리를 삽입하는 경우 다른 엔트리의 이동은 발생하지 않고 슬롯 자료구조만이 이동한다. NVRAM을 보조기억 장치로 사용하는

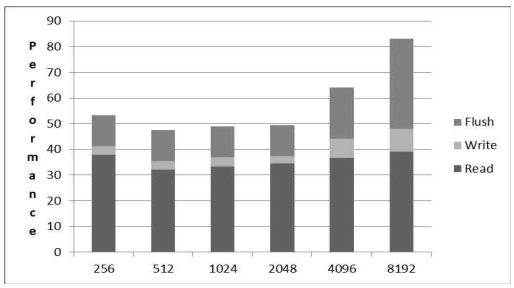
시스템에서도 슬롯 자료구조를 활용하는 시스템을 기반으로 수식 모델을 정의하였다.

$$(R+I)*\log_{\frac{ND-MD}{ET+SL}}(N+\frac{I}{2})*HR*R_{dram}*\log_2(0.7*\frac{ND-MD}{ET+SL})*(KEY+SL)$$
$$+(R+I)*\log_{\frac{ND-MD}{ET+SL}}(N+\frac{I}{2})*(1-HR)$$
$$*(R_{storage}*NS+R_{dram}*\log_2(0.7*\frac{NS-MD}{ET+SL})*(KEY+SL)$$
$$+I*(ET*W_{dram}+\frac{0.7*SL}{2}*(\frac{ND-MD}{ET+SL})*(R_{dram}+W_{dram}))$$
$$+\frac{I*(ET+SL)}{0.4*(ND-MD)}$$
$$*(ET*W_{dram}+(\frac{ND-MD}{2}+\frac{0.7*SL}{2}*(\frac{ND-MD}{ET+SL})*(R_{dram}+W_{dram}))$$
$$+(I+2*\frac{I*(ET+SL)}{0.4*(ND-MD)})*ND*W_{storage}$$

5.3 각 저장장치 실험에 기반한 수식 모델 검증

본 논문에서는 SQLite 시스템 환경에서 인덱스 페이지 읽기 횟수, 인덱스 페이지 버퍼히트 횟수, 인덱스 페이지 분할 횟수 등을 측정, 정의한 수식 모델을 검증하고 이를 활용하여 NVRAM을 보조 장치로 활용하는 시스템 구조의 성능을 평가 하였다. 이번 절에서는 정의한 수식모델을 검증하기 위하여 실험을 진행하였다. 실험결과 실제 시스템에서 측정한 실험결과와 다소 차이를 보였지만 페이지 크기에 따른 실험결과 특성이 비슷한 것을 결과를 통해 알 수 있다.

(그림 11) 수식 모델 기반 실험 결과



(Figure 11) Experiment result of expression

6. 결론

본 논문에서는 빈번한 임의 쓰기를 요구하는 인덱스를 바이트 접근이 가능한 NVRAM에 저장하는 시스템 구조를 제안하였다. 제안하는 시스템 구조에서 인덱스 페이지 크기 및 노드 구조를 최적화하여 NVRAM에 소형 크기로 인덱스를 저장할 때 성능 향상이 있음을 실험적으로 보였다. 그리고 결과적으로 NVRAM의 바이트 단위의 빠른 읽기/쓰기 성능을 활용하여 업데이트 크기에 최적화 한 페이지 단위로 NVRAM에 저장함으로써 불필요한 쓰기와 쓰기 시간을 줄임으로써 시스템적인 낭비를 최소화하고 결과적으로 시스템 성능 향상을 보였다. 추후 연구 과제로 NVRAM의 바이트 접근 특성을 이용하여 256byte와 512byte 사이의 최적의 페이지를 찾는 연구를 진행한다면 모든 면에서 완벽한 페이지를 찾을 수 있을 것이라고 생각한다.

References

[1] H. H. Kim, Y. J. Song, and S. Y. Lee, "Novel Integration Technologies for Highly Manufacturable 32Mb FRAM," In Proceedings of the 2002 Symposium on VLSI Technologies, Digest of Technical Paper, pp. 210-211.

[2] S. J. Ahn, Y. N. Hwang, and Y. J. Song, "Highly Reliable 50nm Contact Cell Technology for 256Mb PRAM," In Proceedings of the 2005 Symposium on VLSI Technologies, Digest of Technical Paper, pp.98-99.

[3] F. Wang, "A Modified Architecture for High-Density MRAM," ACM SIGARCH Computer Architecture News, Vol.29, pp.16-22.

[4] H. G. Lee and N. Chang, "Energy-Aware Memory Allocation in Heterogeneous Non-Volatile Memory Systems," In Proceedings of the 2003 International Symposium on Low Power Electronics and Design, pp.420-423.

[5] S. Ivanov, "Solid State Disk Architecture on the Basis of FRAM," In Proceedings of the International Conference on Computer Systems and Technologies, pp. 32-36.

[6] SQLite, http://www.sqlite.org

[7] Intel corporation, “3 Volt Synchronous Intel StrataFlash Memory,” http://www.intel.com/.

[8] Samsung Electronics, “256M x 8 Bit / 128M x 16

Bit NAND Flash Memory," http://www.samsung-electronics.com/.

[9] NAND(K9F2G08U0M-Samsung Electronics), FeRAM(FM22L16, Ramtron), MRAM(MR4A08B-Everspin Technologies) Datasheet.

[10] Michael Owens, "The Definitive Guide to SQLite," Apress, 2006

[11] Goetz Graefe, Harumi Kuno, "Database software for non-volatile byte-addressable memory, hpi.hp.com, 2011

[12] Non-volatile Random Access Memory and NAND Flash Memory Integrated Solid-State Drives with Adaptive Codeword Error Correcting Code for 3.6 Times Acceptable Raw Bit Error Rate Enhancement and 97% Power Reduction, 2011

[13] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in HPCA-16: The 16th IEEE International Symposium on High-Performance Computer Architecture. IEEE, Jan 2010, pp. 141 - 153

[14] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, Jude A. Rivers Scalable High Performance Main Memory System Using Phase-Change Memory Technology The 36th International Symposium on Computer Architecture (ISCA 2009)

[15] Hyo-Gi Sim, Kyounghon Yoon, Sungmin Park, Ho-young Jung, Sooyong Kang, Jaehyuk Cha "A Study of the Merging Layers of the Storage System for Flash-Based DBMS", Journal of Digital Contents Society Vol.8 no.4 p593-600.

배 상 희

2010년 : 국민대학교 컴퓨터공학과 (공학사)
2012년 : 한양대학교 전자컴퓨터통신학과 (공학석사)
2012년~현 재: KT 사원
관심분야 : 데이터베이스, 파일시스템, 플래시 메모리, 클라우드

이 태 화

2011년 : 안양대학교 컴퓨터학과 (공학사)
2012년 : 한양대학교 전자컴퓨터통신학과 (공학석사)
2013년~현 재 : 한양대학교 대학원 컴퓨터 소프트웨어학과 (박사과정)
관심분야 : 데이터베이스, 파일시스템, 플래시 메모리, NVRAM

차 재 혁

1987년 : 서울대학교 계산통계학과 (이학사)
1991년 : 서울대학교 컴퓨터공학과 (공학석사)
1997년 :서울대학교 컴퓨터공학과 (공학박사)
1998년~2001년 : 한양대학교 컴퓨터교육과 조교수
2002년~현 재 : 한양대학교 컴퓨터공학부 교수
관심분야 : 데이터베이스, 파일시스템, 플래시 메모리, 이러닝, 콘텐츠변환 등