인덱스 저장을 위한 NVRAM 활용 방안

주원진*, 정호영*, 차재혁*
*한양대학교 컴퓨터공학과

e-mail : {artneer, horong, chajh} @ hanyang.ac.kr

Consideration on Using NVRAM as Index Storage

Won-Jin Joo*, Ho-Young Jung*, Jae-Hyuk Cha*
*Dept. of Computer Engineering, Han-Yang University

요 약

최근 저전력과 새로운 응용의 대용량 데이터 처리 요구에 따라 저장 장치로 하드디스크 대신 빠른 입출력 성능을 가진 SSD 를 저장장치로 활용하는 사례가 증가하고 있으나 빈번한 임의 쓰기를 발생하는 소규모 특정 데이터를 하드디스크나 SSD 에 저장하는 경우 발생하는 성능 및 안정성 저하 문제는 아직 완전히 해결하지 못하고 있다. 본 논문에서는 빈번한 임의 쓰기를 요구하는 인덱스를 바이트 접근이 가능한 NVRAM 에 저장하는 시스템 구조를 제안한다. 제안하는 시스템 구조는 NVRAM 의 바이트 단위 빠른 읽기/쓰기와 인덱스 페이지 내 실제 데이터 변경 크기가 블록 크기보다 훨씬 작다는 특성을 고려하여 인덱스 페이지 크기를 최적화하고 메인 메모리에서 인덱스 페이지 버퍼를 따로 할당하여 관리함으로써 시스템 성능 향상을 기대할 수 있음을 보인다.

1. 서론

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 회수 등의 SSD 고유의 문제점을 극복하는 방안에 대한 연구가 활발하다[1]. 그러나 빈번한 임의 쓰기를 발생하는 소규모 특정 데이터를 하드디스크나 SSD 에 저장하는 경우 발생하는 성능 및 안정성 저하 문제는 아직 완전히 해결하지 못하고 있다[2].

최근 현재 양산중인 메모리들의 근본적인 문제를 해결하기 위하여 기존의 SRAM, DRAM 및 플래시 메모리의 장점만을 융합한 차세대 비휘발성 메모리(Next Generation Non-volatile Memory) 개발이 진행되어 오고 있으며 차세대 비휘발성 메모리로는 FeRAM, MRAM, PRAM 그리고 고분자 메모리 등이 있다[3]. MRAM, PRAM, FeRAM 등의 NVRAM 은 낸드 플래시 메모리와 다르게 바이트 단위의 빠른 읽기/쓰기와 In-place-update 가 가능하나 블록 단위의 읽기/쓰기 성능이 그리 뛰어나지 못한 한계점을 지니고 있다.

본 논문에서는 빈번한 임의 쓰기를 요구하는 인덱스를 바이트 접근이 가능한 NVRAM 에 저장하는 시스템 구조를 제안한다. 제안하는 시스템 구조는 NVRAM 의 바이트 단위의 빠른 읽기/쓰기와 인덱스 페이지 내 실제 데이터 변경 크기가 블록 크기보다 훨씬 작다는 특성을 고려하여 인덱스 페이지 크기를 최적화하고 메인 메모리에서 인덱스 페이지 버퍼를 따로 할당하고 관리함으로써 성능 향상을 기대할 수 있음을 보인다.

<표 1> 메모리 성능 비교 [4]

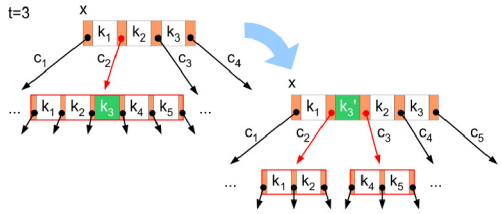
Item	DRAM	FRAM	PRAM	MRAM	NOR	NAND
Byte Addressable	YES	YES	YES	YES	Read only	NO
Non-volatile	NO	YES	YES	YES	YES	YES
Read	10ns	70ns	68ns	35ns	85ns	15us
Write	10ns	70ns	180ns	35ns	6.5us	200us
Erase	none	none	none	none	700ms	2ms
Power consumption	High	Low	High	Low	High	High
Capacity	High	Low	High	Low	High	Very High
Endurance	10^{15}	10^{15}	$> 10^{7}$	10^{15}	100K	100K
Prototype Size		64Mbit	512Mbit	4MBit		

본 논문의 구성은 다음과 같다. 2 장에서는 NVRAM 을 활용한 인덱스 저장의 필요성에 대해 기술한다. 3 장에서는 제안하는 시스템 구조에 대해 기술한다. 4 장에서는 결론 및 향후 과제에 대해 기술한다.

2. NVRAM 을 활용한 인덱스 저장의 필요성

인덱스는 검색 연산을 최적화하기 위해 디스크 상의 데이터 레코드들을 구성하는 데이터 구조이다. 인덱스는 크게 해시 기반의 인덱스와 트리 기반의 인덱스로 구분할 수 있으며, 일반적으로 관계형 데이터베이스 시스템에서는 대표적인 트리 구조 인덱스인 B+TREE 인덱스를 주로 활용한다. B+TREE 인덱스는 데이터를 키와 레코드 형식으로 나누어 인덱스 엔트리라 불리는 단위로 데이터를 관리한다. 또한 저장장치의 입/출력 단위에 적합한 노드를 정의하고 인덱스 엔트리를 저장한다[5].

B+TREE 는 그림 1 에서와 같이 데이터 업데이트에 따른 분할 및 병합 연산으로 부모 및 형제 노드까지 영향을 줄 수 있으며 여러 노드에 동시에 업데이트가



(그림 1) B-TREE 분할 연산의 예[6]

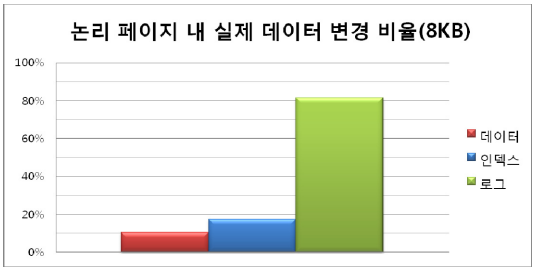
발생할 수 있다. 일반적인 DBMS 는 주기적으로 업데이트가 발생한 모든 페이지를 저장 장치에 기록하는데 이 때 페이지 내에 업데이트되지 않은 데이터 부분도 쓰기가 발생하여 시스템적인 낭비를 초래한다.

본 논문에서는 실제 인덱스 파일에서 페이지 내 실제 업데이트 크기가 어느 정도인지를 측정 하기 위해 실험을 진행하였다.

<표 2> 페이지 내 실제 업데이트 측정 실험 환경

CPU	AMD Athlon 64 X2 Dual Core Processor 420 2.21GHz
Main Memory	2 GB
OS	Ubuntu Linux 9.10 (Kernel Version : 2.6.31)
DBMS	Oracle 10g
벤치마크	TPC-C (hammerora_2.3)
벤치마크 환경	Warehouses : 05 Total Transactions per User : 100(Data, Log) / 500 (Index) Clients : 05 Iteration : 1
파일 크기	데이터 파일 : 490,741,760 인덱스 파일 : 41,099,264 로그 파일 : 52,429,312

<표 2> 와 같은 실험 환경에서 데이터, 로그, 인덱스 파일에서 페이지 내 실제 업데이트 크기 비율을 측정하였다.

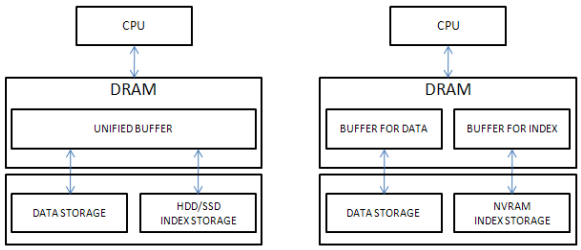


(그림 2) 페이지 내 실제 업데이트 크기 비율

로그는 페이지 내 실제 업데이트 크기 비율이 81.4% 인 것에 비해 데이터와 인덱스는 각각 10.5%, 17.4%로 페이지 내 실제 업데이트 크기가 일반적인 논리 페이지 크기(8KB)보다 훨씬 작게 측정되었다. 이와 같은 실험 결과를 고려하여 참조 빈도가 높고 빈번한 임의 쓰기를 요구하는 인덱스 저장을 위해 바이트 단위의 빠른 읽기/쓰기가 가능한 NVRAM 을 활용함으로써 불필요한 쓰기와 쓰기 시간을 줄이고 결과적으로 전체적인 시스템 성능 향상을 기대할 수 있다.

3. NVRAM 을 활용한 인덱스 저장장치 시스템

본 논문에서는 빈번한 임의 쓰기를 요구하는 인덱스 데이터를 바이트 접근이 가능한 NVRAM 에 저장하는 시스템 구조를 제안한다.



(A) 기존 시스템 구조 (B) 제안 시스템 구조

(그림 3) 인덱스 저장 장치 구조

NVRAM 은 일반적인 페이지 크기인 4KB, 8KB 단위의 I/O 읽기/쓰기 성능이 좋지 못하다. 이에 본 논문에서는 NVRAM 의 바이트 단위의 빠른 읽기/쓰기와 인덱스 파일에서 페이지 내 실제 업데이트 크기가 4KB, 8KB 보다 훨씬 작다는 것을 활용하여 그림 3-(B)와 같은 인덱스 저장 구조를 제안한다. 제안하는 시스템 구조에서 전 영역에 걸쳐 빈번한 임의쓰기를 요구하는 인덱스를 NVRAM 에 저장하며 인덱스 페이지 내 업데이트 크기와 페이지 크기에 따른 분할 비용을 고려하여 인덱스 페이지 크기를 최적화한다. 그리고 메인 메모리에서 인덱스 페이지 버퍼를 따로 할당하여 관리한다.

본 논문에서 제안한 방식은 인덱스 만을 NVRAM 에 저장하므로 비교적 작은 용량의 NVRAM 이 요구되며 1) 적은 비용으로 큰 성능 향상을 기대할 수 있으며 로그, 롤백 세그먼트와 같은 데이터는 버퍼에서 저장 장치에 순차적으로 저장되는 것에 비해 빈번한 임의 쓰기가 요구되는 인덱스를 NVRAM 에 저장함으로써 하드디스크, SSD 에서 문제시 되었던 2)임의 쓰기에서 발생하는 급격한 성능저하의 문제점을 극복할 수 있다. 그리고 결과적으로 NVRAM 의 바이트 단위의 빠른 읽기/쓰기 성능을 활용하여 업데이트 크기에 최적화 한 페이지 단위로 NVRAM 에 저장함으로써 3) 불필요한 쓰기와 쓰기 시간을 줄임으로써 시스템적인 낭비를 최소화하고 결과적으로 시스템 성능 향상을 기대할 수 있다.

4. 결론 및 향후 과제

본 논문에서는 빈번한 임의 쓰기를 요구하는 인덱스 데이터를 바이트 접근이 가능한 NVRAM 에 저장하는 시스템 구조를 제안하였다. 제안하는 시스템 구조에서 인덱스 페이지 크기를 최적화하여 저장하는 경우 시스템적인 낭비를 줄이고 시스템 성능 향상을 기대할 수 있음을 보였다. 하지만 페이지 크기를 줄임으로써 B+TREE 의 분할 연산의 오버헤드 증가를 고려해야 하는 문제점이 남아있다.

향후 과제로는 분할 연산의 오버헤드 증가와 페이지 크기에 따른 쓰기 시간 감소를 고려한 페이지 크기 최적화와 NVRAM 인덱스 구조를 개선하여 시스템의 성능을 보다 효율적으로 증가시키는 것이 필요하다.

참고문헌

[1] Sang-Won Lee, Bongki Moon, Chanik Park, Jae-Myung Kim, Sang-Woo Kim, "A Case for Flash Memory SSD in Enterprise Database Applications" ACM SIGMOD International Conference on Management of Data, Vancouver, Canada, 2008

[2] Bouganim, L., Jónsson, B., Bonnet, P.: uFLIP: Understanding flash IO patterns. In: Fourth Biennial Conference on Innovative Data Systems Research, 2009

[3] Jin Kyu Kim, Hyung Gyu Lee, Shinho Choi, Kyoung Il Bahng: A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems. EMSOFT 2008

[4] Jung, Jaemin and Won, Youjip and Kim, Eunki and Shin, Hyungjong and Jeon, Byeonggil, "FRASH: Exploiting storage class memory in hybrid file system for hierarchical storage", ACM Trans. on Storage, volume:6, number: 1, page(s): 1-25, March, 2010

[5] 나갑주, 이상원, IPL 기반의 B-Tree 인덱스 설계. 한국인터넷정보학회 2007

[6] http://upload.wikimedia.org/wikipedia/commons/5/58/B-tree-splitt.png