Journal of The Korea Society of Computer and Information
Vol. 20, No. 5, May 2015
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.5.001

NAND 플래시 메모리 기반 B+ 트리에서 페이지 매핑 로그를 이용한 색인 관리 기법

김 선 환*, 곽 종 욱*

Index Management Method using Page Mapping Log in B+-Tree based on NAND Flash Memory

Seon Hwan Kim*, Jong Wook Kwak*

요 약

낸드 플래시 메모리는 저전력, 빠른 접근 속도, 저렴한 가격 등의 특징을 가지고 있어 저장장치로 널리 사용되고 있다. 하지만 낸드 플래시 메모리는 제자리 덮어쓰기가 지원되지 않아 기존의 하드 디스크 기반 응용 프로그램을 구동하기 위해서는 FTL(Flash Translation Layer)이 필요하다. FTL은 주소 매핑, 가비지 컬렉션, 마모도 균등화 작업 등을 포함하고 있어 저사양 임베디드 장치에 구현하기에는 메모리와 연산에 대한 비용이 많이 든다. 그래서 이런 장치들을 위해 낸드 플래시 메모리에 최적화된 색인 자료구조들이 연구되고 있다. 연구된 방법들은 쓰기에 소요되는 시간을 줄여 성능을 향상시켰지만 레코드 탐색에 소요되는 시간이 증가된다는 단점을 가지고 있다. 레코드 탐색 시간을 증가시키지 않고 쓰기 횟수를 줄이기 위해 본 논문에서는 페이지 매핑 로그 테이블을 이용한 색인 관리 기법을 제안한다. 낸드 플래시 메모리의 단점인 제자리 덮어쓰기 불가로 인해 발생하는 페이지 쓰기 횟수를 줄이기 위해 매핑 로그 테이블은 B+ 트리에서 변경된 노드 페이지 주소를 저장하고 레코드 검색 시 이를 이용한다. 실험 평가를 통해 제안된 기법은 다른 기법들과 비교 시 레코드 탐색에서 발생하는 페이지 읽기 횟수를 최대 약 61% 줄였으며, 레코드 삽입에서 페이지 쓰기 횟수를 최대 약 31% 줄일 수 있었다.

▸ Keywords : 플래시 메모리, B+ 트리, 색인 자료구조, 임베디드 소프트웨어, 저장장치

Abstract

NAND flash memory has being used for storage systems widely, because it has good features which are low-price, low-power and fast access speed. However, NAND flash memory has an in-place update

•제1저자 : 김선환 •교신저자 : 곽종욱
•투고일 : 2015. 4. 9, 심사일 : 2015. 5. 7, 게재확정일 : 2015. 5. 16.
* 영남대학교 컴퓨터공학과(Department of Computer Engineering, Yeungnam University)
※ 이 논문은 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임.
(No. NRF-2014R1A1A2057146)

problem, and therefore it needs FTL(flash translation layer) to run for applications based on hard disk storage. The FTL includes complex functions, such as address mapping, garbage collection, wear leveling and so on. Futhermore, implementation of the FTL on low-power embedded systems is difficult due to its memory requirements and operation overhead. Accordingly, many index data structures for NAND flash memory have being studied for the embedded systems. Overall performances of the index data structures are enhanced by a decreasing of page write counts, whereas it has increased page read counts, as a side effect. Therefore, we propose an index management method using a page mapping log table in B^+-Tree based on NAND flash memory to decrease page write counts and not to increase page read counts. The page mapping log table registers page address information of changed index node and then it is exploited when retrieving records. In our experiment, the proposed method reduces the page read counts about 61% at maximum and the page write counts about 31% at maximum, compared to the related studies of index data structures.

▸ Keywords : Flash memory, B^+-Tree, Index data structure, Embedded software, Storage system

I. 서 론

플래시 메모리는 일반 하드 디스크에 비해 접근속도가 빠르고 소형화 및 경량화가 쉬워 저장 장치로 많이 사용되고 있다. 플래시 메모리는 비휘발성 메모리로 외부적인 전원 공급이 없더라도 저장된 데이터를 유지하며 기계적인 장치를 사용하지 않아 직접 접근 시간이 빠르다. 이런 특징으로 SSD(Solid State Drive)를 포함하여 USB 메모리, 센서 노드, 휴대폰 등에 많이 활용되고 있으며, 플래시 메모리의 직접도가 지속적으로 늘어나고 가격이 저렴해지고 있어 해마다 사용량이 늘어나고 있는 추세이다.

하지만 플래시 메모리는 하나의 페이지에 덮어쓰기가 지원되지 않으며, 하나의 블록에 지울 수 있는 횟수가 제한되어 있다는 단점을 가지고 있다. 또한 쓰기 및 읽기는 페이지 단위로 동작하지만 지우기는 블록단위로 동작한다. 그래서 이러한 제약 특성들을 고려하여 플래시 메모리를 관리하여야 한다. 이런 제약들을 극복하기 위해 여러 방법들을 사용하고 있다. 덮어쓰기 작업을 수행하는 경우 다른 빈 페이지에 갱신된 데이터를 저장하고, 그 데이터가 저장된 페이지의 정보를 주소 매핑 테이블에 기록해서 해당 데이터에 읽기가 발생할 때 주소 매핑 테이블을 검색하여 변경된 페이지의 주소를 찾아 데이터를 읽는다. 다음으로, 각 블록 당 지우기 횟수가 제한되어 있는 점은 전체 블록의 지우기 횟수를 평준화하는 마모도 균등화(wear leveling) 기법을 사용하여 해결한다. 이런 주소 매핑, 마모도 균등화 작업 이외에도 가비지 컬렉션, 복구 기능 등을 사용하여 플래시 메모리를 관리하는 계층을 FTL(Flash Translation Layer)이라고 한다. FTL의 주요 기능인 주소 매핑은 크게 페이지 단위 매핑, 블록 단위 매핑, 하이브리드 매핑 등으로 나눌 수 있다. 페이지 단위 매핑은 속도가 빠르며 마모도 균등화에 유리하지만 메모리 공간이 많이 필요하다. 블록 단위 매핑은 작은 메모리 공간을 사용하지만 매핑 속도가 느리다. 하이브리드 매핑은 앞서 설명한 두 개의 기법을 혼용한 방식이다[1, 2, 3].

FTL은 플래시 메모리에 기존의 하드 디스크를 기반으로 하는 어플리케이션, 파일 시스템, 데이터베이스 등을 사용할 수 있게 한다. 하지만 대표적인 색인 자료구조라고 할 수 있는 B+ 트리는 하드 디스크를 고려하여 제안된 것이기 때문에 플래시 메모리에서 최적의 성능을 발휘할 수 없다. FTL이 있다고 하더라도 제자리 덮어쓰기 불가와 가비지 컬렉션의 수행 때문에 성능이 저하된다. 특히 저사양의 임베디드 시스템에서는 필요한 메모리와 연산기능 때문에 FTL을 탑재하기 힘들어 B+ 트리의 성능이 더욱 저하된다. 이를 해결하고자 플래시 메모리의 특성을 고려한 색인 자료구조들이 다양하게 연구되고 있다[4-14].

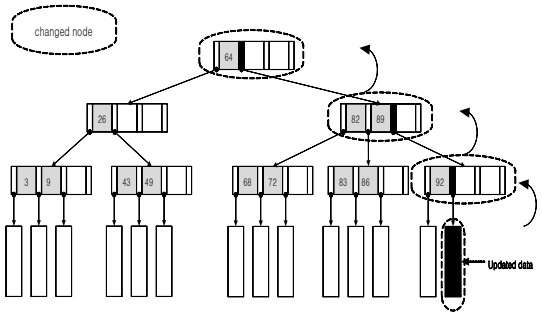


그림 1. 업데이트 파생 문제
Fig 1. Update propagation problem

플래시 메모리의 특성을 고려한 B+ 트리들은 플래시 메모리에서 페이지 쓰기 작업이 페이지 읽기 작업보다 느리다는 점에 착안하여 페이지 쓰기 횟수를 줄여 전체적인 성능을 향상 시켰다. 하지만 이런 기법들은 페이지 쓰기 횟수를 줄이기 위해 대부분 색인의 정보를 여러 페이지에 나누어 저장하거나 로그형태로 추가적인 페이지에 기록하기 때문에 레코드 탐색에 소요되는 시간이 증가되는 단점이 있다. 특히 레코드 갱신 및 삽입보다 탐색에 대한 비율이 높아진다면 오히려 전체적인 성능이 저하될 수 있다.

이를 해결하기 위해 본 논문에서는 낸드 플래시 메모리 기반 B+ 트리에서 페이지 매핑 로그(PML: Page Mapping Log)를 이용한 색인 관리 기법을 제안한다. PML을 이용한 색인 관리 기법은 레코드 연산에서 색인 노드의 갱신으로 변경되는 낸드 플래시 메모리의 페이지 위치를 PML 테이블에 저장하고 색인 노드 탐색 시 저장된 위치를 활용한다. 제안하는 기법은 기존 하드 디스크를 기반으로 하는 일반 B+ 트리보다 페이지 쓰기 횟수를 줄이면서 레코드 탐색 성능은 저하시키지 않는다. 이하 본 논문에서는 2장에 하드 디스크 기반 B+ 트리를 플래시 메모리에 적용할 경우 생기는 문제에 대해서 언급하고 플래시 메모리 기반의 색인 자료구조 연구들을 설명한다. 3장에서는 제안하는 기법을 소개한다. 4장에서 실험환경을 구축하고 성능 및 평가를 수행한다. 그리고 5장에서 결론을 맺는다.

II. 배경 지식 및 관련 연구

이 장에서는 낸드 플래시 메모리에 하드 디스크를 기반으로 하는 일반 B+ 트리를 적용할 경우 발생되는 문제점을 설명하고, 이를 해결하고자한 연구들과 이와 관련된 낸드 플래시 메모리에 최적화한 색인 자료구조에 대해서 설명한다.

1. 플래시 메모리에서 B+ 트리의 문제점

B+ 트리는 다계층의 색인 노드로 구성되어 있으며 말단 색인 노드는 레코드가 정렬되어 있는 데이터 노드와 연결되어 있다. B+ 트리는 항상 균형 트리를 유지하여 레코드 탐색 시 모두 동일한 탐색 속도를 가지는 특징이 있다. 색인 노드에서는 1/2 이상의 키를 저장하고 있고 저장할 수 있는 키의 개수가 초과하면 노드의 분할 작업이 발생한다.

B+ 트리는 하드 디스크 기반이기 때문에 낸드 플래시 메모리에 적용하면 페이지 쓰기 횟수가 많이 발생하여 전체적인 성능을 저하시킨다. 이는 낸드 플래시 메모리의 단점인 제자리 덮어쓰기가 되지 않고, 읽기 및 쓰기 단위와 지우기 단위가 다르다는 것을 고려하지 않았기 때문이다. 이로 인해 생기는 주요 성능 저하 요인이 업데이트 파생 문제(update propagation problem)이다. 업데이트 파생 문제는 다른 연구에서는 트리 배회 문제(wandering tree problem)로 언급되기도 한다[4, 5, 6, 15]. 예를 들자면, 트리의 가장 오른쪽에 위치하고 있는 레코드가 갱신될 경우 플래시 메모리의 특성상 덮어쓰기가 되지 않기 때문에 다른 페이지에 갱신된 데이터를 써야 된다. 그리고 페이지 주소가 변경된 해당 말단 노드는 B+ 트리의 특성상 부모노드가 자식노드의 주소를 저장하기 때문에 부모노드 또한 데이터를 갱신해야 한다. 같은 방식으로 부모노드의 데이터가 갱신되면 덮어쓰기 문제로 인하여 상위 노드가 모두 갱신되어야 한다.

그림 1은 이와 같은 업데이트 파생 문제를 나타내고 있다. 높이가 4이며 노드의 차수가 3인 B+ 트리가 있다. 93값을 키로 가지는 레코드를 갱신한다고 할 때 트리를 탐색하여 레코드가 저장되어 있는 가장 오른쪽에 있는 데이터 노드를 찾아 갱신한다. 그러면 데이터 노드의 페이지 위치가 변경되어 상위 노드인 색인 노드도 가리키는 위치를 변경해야 하기 때문에 색인 노드도 변경되고 반복적으로 최상위 색인 노드까지 갱신이 되게 된다. 따라서 그림 1의 예처럼 하나의 레코드를 갱신하기 위해 총 4회의 쓰기가 발생하게 된다. 결국 B+ 트리의 레코드를 갱신하기 위해서는 트리의 높이만큼 추가적인 쓰기 연산이 발생하게 되며, 이는 성능저하의 주요 원인이 된다.

2. 플래시 메모리를 위한 색인 자료구조

μ-트리는 업데이트 파생 문제를 해결하기 위해 갱신이 발생 할 경우 상위 노드들을 한 개의 페이지에 모두 저장한다. 한 개의 페이지에 상위 색인 노드들을 모두 저장하기 위해서 상위 노드를 저장할 때마다 페이지의 영역을 1/2로 나누어 저장한다[4]. μ*-트리는 μ-트리에서 공간 효율성을 높이기 위해 B+ 트리의 높이와 키의 개수 등을 분석하고 계산을 통하여 페이지 영역을 할당하여 사용한다[5]. IPL(In-Page Logging)은 B+ 트리에서 키의 갱신이나 삽입이 발생하면 낸드 플래시 메모리 블록의 일정 영역에 해당 정보를 로그 형태로 저장한다. 키 검색 시 변경된 값은 같은 블록의 로그영역에서 참조하게 된다[6]. D-IPL은 IPL이 SLC 낸드 플래시 메모리에서 활용될 수 있지만 MLC 낸드 플래시 메모리는 각 블록의 페이지들이 순차적으로 써야 하는 단점을 가지고 있어 MLC 낸드 플래시 메모리에 적용할 수 있도록 수정하였다[7]. AD(Adaptive Durable)-트리는 레코드의 변경 이력을 콜드(cold) 버퍼와 핫(hot) 버퍼에 저장하고 이를 플래시 메모리에 저장한다. 그리고 트리의 재구성 비용을 줄이고자 형제 노드들을 하나의 페이지에 저장한다[8].

BFTL(B+-Tree Flash Translation Layer)은 예약 버퍼(reservation buffer)를 사용하여 갱신이나 삽입된 키를 버퍼에 저장하고 버퍼의 내용을 연속된 페이지에 저장하여 쓰기 횟수를 줄인다. 각 색인 노드의 내용이 여러 페이지에 나누어 저장되기 때문에 노드 변환 테이블(node translation table)을 두어 레코드 탐색 시 이를 이용한다[9]. LA(Lazy-Adaptive)-트리는 트리를 세부 구역으로 나누고 각 구역별로 버퍼를 할당하여 사용한다. 갱신과 삽입이 발생하면 각 버퍼에 내용을 저장하고 각 구역의 버퍼 내용이 차면 하위 구역에 있는 버퍼에 데이터를 이주하며 버퍼의 사용 빈도에 따라 버퍼의 크기를 조절한다. 그리고 추후 버퍼와 색인 노드를 같이 저장한다[10]. FlashDB는 트리의 레코드 연산을 감시하여 레코드의 탐색이 많을 때는 B+ 트리 형태를 유지하고 레코드의 갱신이 많을 때는 BFTL의 형태를 유지한다[11].

이상의 사실로부터 주어진 기법들을 상호 비교 분석하자면 다음과 같다. μ-트리와 μ*-트리는 상위 노드로 갈수록 저장할 수 있는 키의 개수가 제한되어 트리의 높이가 증가하며, IPL, D-IPL AD-트리는 각 색인 노드를 접근할 때 플래시 메모리에 있는 버퍼 또는 로그영역을 참조하기 때문에 레코드 탐색 시 페이지의 읽기가 많이 발생한다는 단점이 있다.

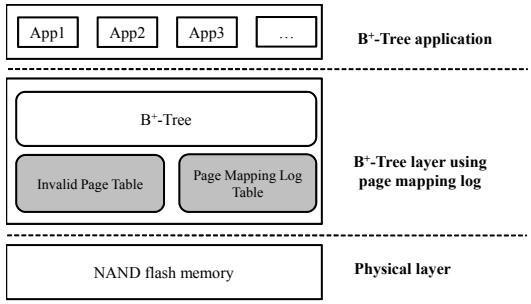


그림 2. PML 이용한 B+ 트리의 전체 구성
Fig. 2. An overview of B+-Tree layer using PML

BFTL, LA-트리, FlashDB는 FTL이 있는 SSD 장치 등에 적용이 가능하지만, 플래시 메모리를 직접적으로 제어하고 FTL을 구현하기 힘든 저사양 임베디드 시스템에는 적용할 수 없다.

III. PML을 이용한 색인 관리

이 장에서는 제안하는 B+ 트리 계층의 전체 구조를 설명하고, 제안된 기법에서 사용되는 무효 페이지 테이블과 PML 테이블의 용도를 설명한다. 그리고 B+ 트리에서 레코드를 삽입, 삭제, 수정하는 방법을 기술한다.

1. 전체 구성

제안하는 B+ 트리 계층은 낸드 플래시 메모리 계층 상위에 위치하며, 최상단에는 이를 사용하는 B+ 트리 관련 응용 프로그램이 위치하고 있다. 그림 2에서처럼, B+ 트리 계층에는 변경된 페이지를 관리하는 무효 페이지 테이블과 PML 테이블을 가지고 있다. PML을 사용한 B+ 트리는 FTL을 적용하기 힘든 임베디드 시스템을 고려하였기 때문에, 낸드 플래시 메모리의 페이지 위치를 직접적으로 접근하여 데이터를 처리한다. 제안한 기법은 하드 디스크를 기반으로 하는 일반 B+ 트리에 PML 테이블과 무효 페이지 테이블만 추가하여 색인 노드의 접근에 각 테이블을 참조하기 때문에 구현이 용이하고 B+ 트리의 구조 변경이 필요 없다.

2. PML 테이블

PML 테이블은 레코드 연산으로 변경된 페이지의 위치를 등록하고 색인 노드를 접근할 때 등록된 정보를 탐색하여 변경된 페이지 위치를 찾을 수 있도록 한다. 낸드 플래시 메모리의 특성상 덮어쓰기 불가 문제로 인하여 레코드 연산 시 색

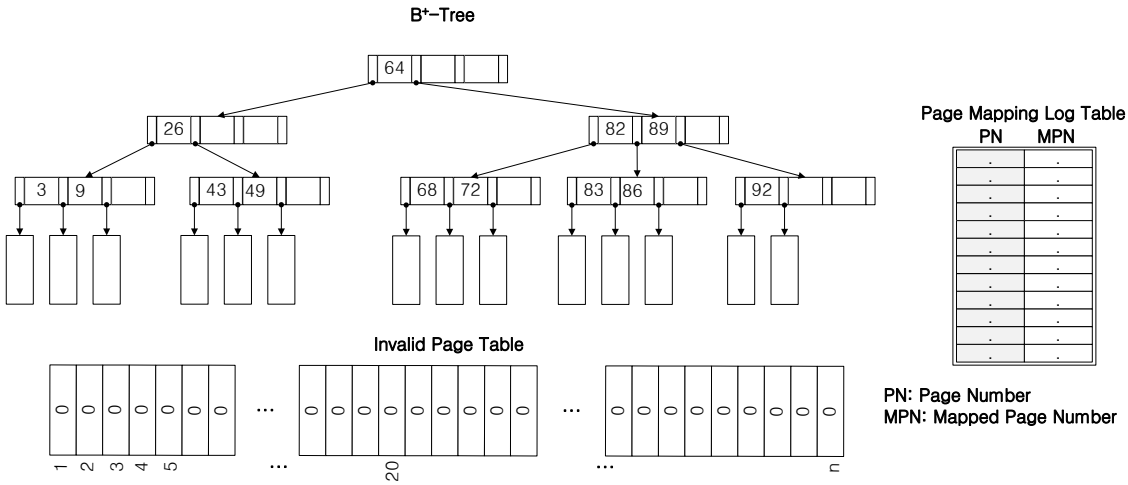


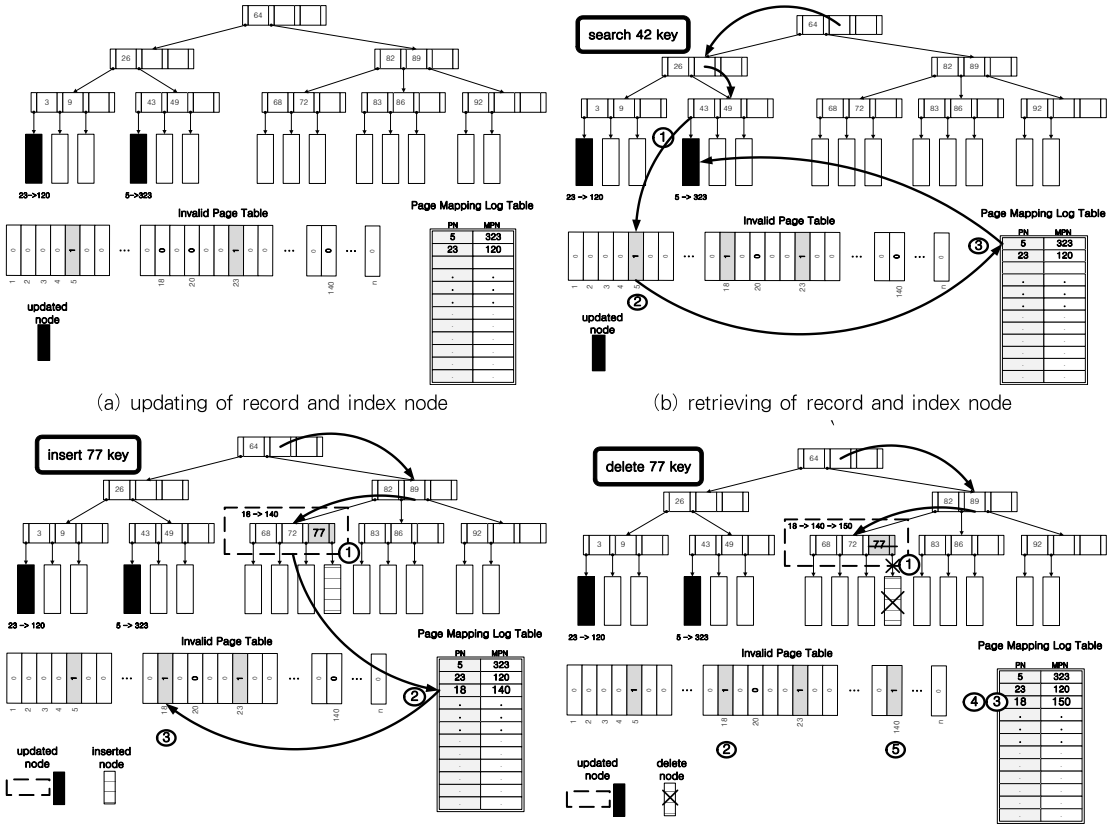
그림 3. PML을 이용한 B+ 트리 전체 구조
Fig. 3. A data structure of B+-Tree using PML

인 노드의 페이지 위치가 자주 변경된다. 이로 인해 일어나는 업데이트 파생 문제를 방지하기 위해 PML 테이블을 사용한다. PML 테이블의 엔트리는 2가지의 정보를 기록한다. 2가지 정보는 페이지 번호(PN: Page Number)와 매핑된 페이지 번호(MPN: Mapped Page Number)의 집합으로 구성된다. 색인 노드 또는 레코드가 저장되어 있는 데이터 노드가 변경이 되면 변경전의 페이지 번호를 PN에 저장하고 변경된 페이지의 번호를 MPN에 저장한다. 그래서 PML 테이블을 참조하게 되면 추가적인 상위 색인 노드의 갱신이 필요 없기 때문에 업데이트 파생 문제를 방지할 수 있다. 레코드를 검색하기 위해 키의 값에 따라 색인 노드를 탐색할 때 색인 노드가 저장된 페이지 위치에 페이지 읽기 연산을 수행하게 된다. 이 때 해당 페이지의 번호를 PML 테이블에서 검색하여 엔트리가 존재한다면 MPN의 해당 위치에서 페이지 읽기 연산을 수행한다. 그리고 색인 노드를 탐색할 때 PML 테이블의 검색 비용을 줄이기 위해 무효 페이지 테이블에서 해당 색인 노드의 페이지가 무효 페이지로 판단될 때만 PML 테이블 검색을 수행한다.

PML 테이블에 더 이상 엔트리를 추가할 수 없으면 상위 색인 노드에 PML 테이블의 변경된 페이지 위치들을 반영한다. PML 테이블의 엔트리를 색인 노드에 반영하게 되면 변경된 색인 노드를 페이지에 다시 기록을 해야 하기 때문에 추가적인 페이지 쓰기가 발생한다. 하지만 PML 테이블의 크기는 가변적으로 변경할 수 있기 때문에 시스템에서 가용할 수 있는 여분의 메모리를 많이 할당할수록 추가적인 페이지 쓰기 발생을 방지할 수 있다. 또한 동일한 레코드의 반복적인 갱신은 PML 테이블의 엔트리만 수정하면 되기 때문에 PML 테이블의 엔트리 증가 없이 업데이트 파생 문제를 효율적으로 방지할 수 있다. 그림 3은 B+ 트리에 무효 페이지 테이블과 PML 테이블이 구성된 구조를 표현하고 있다.

3. 무효 페이지 테이블

무효 페이지 테이블은 비트맵으로 구성이 되어 있으며 각 페이지를 무효 페이지와 유효 페이지로 구분하는 용도로 사용된다. 테이블에 저장된 값이 '0' 비트이면 유효 페이지로 인식하고 '1' 비트이면 무효 페이지로 인식한다. 테이블의 공간은 전체 낸드 플래시 메모리의 페이지 수만큼 비트 공간을 사용한다. 그래서 일차원 비트 배열로 구성되어 있기 때문에 해당 페이지가 무효 및 유효 페이지인지 빠르게 구분할 수 있다. 또한 무효 페이지 테이블은 낸드 플래시 메모리에서 페이지를 관리하기 위해 기본적으로 필요한 테이블이다. 낸드 플래시 메모리를 직접적으로 제어하고 구동하는 프로그램들은 가용 페이지를 찾기 위해 무효페이지 테이블을 이용해야 하고 가비지 컬렉션 수행에서는 희생 블록을 선정하기 위해 참조해야 한다. 그래서 PML을 이용한 B+ 트리에서는 추가로 새로운 테이블을 하지 않고 무효 페이지 테이블을 활용하여 색인 노드가 변경되었다는 이력을 알 수 있고 PML 테이블의 탐색을 줄이도록 한다. 색인 노드를 접근할 때 무효 페이지 테이블을 참조하여 무효 페이지 유무를 판단하고, 무효 페이지라고 판단되면 해당 색인 노드나 데이터 노드가 변경된 페이지 위치에 존재한다고 인식하여 PML 테이블을 검색하게 된다.



(a) updating of record and index node

(b) retrieving of record and index node

(c) inserting of record and index node

(d) deleting of record and index node

그림 4. B+ 트리에서 PML 테이블을 이용한 각 레코드 연산 과정
Fig. 4. Operations of record using PML table in B+-Tree

4. B+ 트리 색인 관리

이 장에서는 레코드의 연산 과정과 PML 테이블의 엔트리를 활용하고 관리하는 방법을 서술한다.

4.1 갱신

레코드의 변경으로 새로운 페이지에 색인 노드나 데이터 노드가 저장되면 이전에 저장된 페이지는 무효 페이지로 전환되어 무효 페이지 테이블에 해당 무효 페이지의 식별 정보를 비트로 저장한다. 그리고 이전의 페이지 번호, 즉 색인 노드가 가리키고 있었던 페이지 번호와 변경된 페이지 번호를 PML 테이블의 엔트리에 등록한다.

그림 4의 (a)는 갱신 시 구성된 B+ 트리와 PML 테이블 및 무효 페이지 테이블의 변경과정을 보여 주고 있다. 검은색으로 표시된 노드는 변경된 노드를 의미한다. 그림 4의 (a)에서는 2개의 노드가 변경이 되었으며 각각 23번과 5번 페이지에 저장되어 있는 노드들이 120번과 323번 페이지에 새로이 저장되었다. 그래서 변경된 노드의 위치 이력을 저장하기 위해 PML 테이블에 2개의 엔트리인 (5, 323), (23, 120)의 정보를 저장하고 무효 페이지 테이블에 5번과 23번 페이지 정보를 '1'로 설정한다. 이렇게 테이블들이 구성되면 추후 검색 시 이를 활용하게 되고 2개의 노드가 갱신이 되어도 쓰기는 2번만 발생하기 때문에 업데이트 파생 문제를 방지하게 된다.

4.2 검색

레코드를 검색하기 위해 색인 노드나 데이터 노드가 저장되어 있는 페이지를 접근하기 전에 무효 페이지 테이블을 참조하고 무효 페이지라고 판단되면 PML 테이블을 검색한다. 그림 4의 (b)는 그림 4의 (a)에서 레코드를 검색하는 과정을 나타내고 있다. 레코드에 관련된 키가 42이고 이와 관련된 데

이터를 찾기 위해 색인 노드를 탐색한다. 해당 레코드가 323번 페이지에 있다고 가정하면 ①까지 탐색이 진행되기 전의 색인 노드들은 무효페이지가 아니기 때문에 PML 테이블을 검색하지 않는다. ①까지 탐색이 진행되면 상위 색인 노드에 저장되어 있는 레코드 노드의 위치가 페이지 5번으로 저장되어 있어 해당 페이지에 접근을 시도할 것이다. 이 때 ②에서 무효 페이지 테이블을 참조해서 5번 페이지가 무효 페이지이기 때문에 ③에서 PML 테이블의 엔트리를 검색하여 실제 데이터가 저장되어 있는 323번 페이지에서 읽기를 수행한다.

4.3 삽입

새로운 레코드가 삽입되어 말단 데이터 노드에 추가되면 바로 상위에 있는 색인 노드도 수정이 이루어져야 한다. 이 때 변경되는 색인 노드의 새로운 페이지 번호를 PML 테이블에 등록한다. 그리고 무효 페이지 테이블에 색인 노드의 이전 페이지 번호를 '1'로 설정하여 무효 페이지로 설정한다. 만약 레코드에 관련된 키를 색인에 추가하는 과정에서 색인 노드에 공간이 부족하다면 색인 노드는 분할된다. 색인 노드가 분할되면 2개의 색인 노드로 나누어지게 되고 각 색인 노드에 키를 나누어 저장한다. 2개의 색인 노드가 저장되려면 2개의 새로운 페이지가 할당되고 2번의 페이지 쓰기 작업이 발생한다. 이때 분할되기 전에 색인 노드가 저장된 페이지 번호는 무효 페이지 테이블에 등록한다. 색인 노드가 새로 갱신되어 저장될 때는 PML 테이블의 엔트리를 검색하여 하위 노드의 위치가 엔트리에 존재한다면 모두 색인 노드에 반영하여 저장한다. 결국 색인 노드가 분할되면 PML 테이블의 엔트리를 줄일 수 있는 기회가 많아지기 때문에 PML 테이블의 효율성이 높아진다. 이렇게 색인 노드의 분할 및 합병으로 인해 추가적인 페이지 쓰기 작업이 없이 PML 테이블의 엔트리를 줄인다는 것은 PML 테이블 탐색의 비용을 줄이고 추후 PML 테이블의 공간이 부족한 상황을 늦추게 되어 전체적인 성능을 유리하게 만든다.

그림 4의 (c)는 레코드와 키 값 77이 삽입되는 과정을 보여주고 있다. 최상위 색인 노드에서 탐색이 이루어지고 해당 키가 존재한다면 갱신이 이루어지고 키가 존재하지 않는다면 삽입이 이루어진다. ①에서 새로운 레코드가 삽입되고 키가 존재하지 않아 색인 노드가 갱신이 된다. 해당 색인 노드가 페이지 18번에 저장이 되었었고 새로운 페이지 140번에 저장되기 때문에 ②에서 (18, 140) 엔트리를 PML 테이블에 저장한다. 그리고 ③에서 무효 페이지 테이블에 18번 페이지를 무효 페이지로 등록한다.

4.4 삭제

레코드 삭제가 발생하면 레코드를 저장한 페이지 번호는 무효 페이지 테이블에 등록하고 관련 키를 삭제하기 위해 색인 노드를 갱신한다. 색인 노드의 갱신과정에서 발생하는 페이지 변경 이력은 PML 테이블에 등록한다. 키가 삭제되면 합병되는 색인 노드들이 생길 수 있다. 삽입과 마찬가지로 색인 노드가 변경이 될 때는 PML 테이블의 엔트리를 반영하여 저장하도록 한다.

그림 4의 (d)는 레코드 삭제로 인하여 77번 키가 삭제되는 과정을 나타내고 있다. ①에서 색인 노드를 접근할 때는 ②에서 무효 페이지 테이블을 참조한 결과 18번 페이지가 무효 페이지이기 때문에 ③에 PML 테이블 엔트리를 검색한 후 140번에서 페이지 읽기를 수행한다. 레코드의 삭제로 색인을 갱신하고 새로운 페이지 쓰기를 수행하면 변경된 페이지의 번호 150을 ④에서 엔트리에 등록한다. 엔트리를 등록할 때 이미 18번에 해당하는 엔트리가 존재하기 때문에 새로 등록하지 않고 (18, 150)으로 수정한다. 엔트리 등록 작업이 완료되면 기존에 색인 노드가 저장된 140번 페이지는 ⑤에서 무효 페이지 테이블에 무효 페이지로 등록한다.

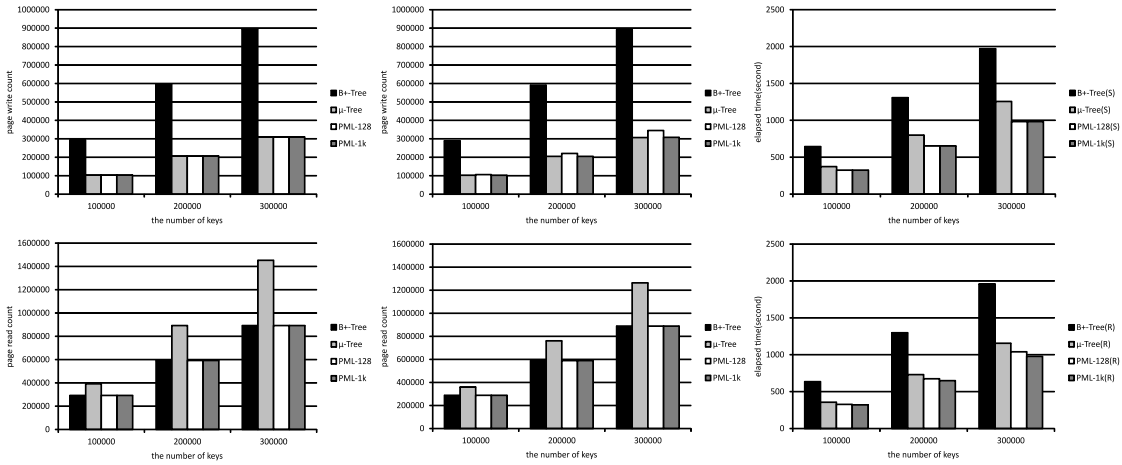
5. 시스템 유지

5.1 가비지 컬렉션

낸드 플래시 메모리에 적용한 하드 디스크 기반 B+ 트리는 가비지 컬렉션에서 발생하는 페이지의 이주로 페이지와 블록의 위치가 변경이 된다면 트리의 재구성을 유발시켜 성능을 저하시키게 된다. 이런 트리의 재구성은 업데이트 파생 문제와 비슷하게 변경된 노드의 위치를 반영하기 위해 상위 색인 노드를 모두 갱신하여 쓰기 횟수가 증가하게 된다. PML을 적용한 B+ 트리는 가비지 컬렉션에도 PML을 이용하여 추가적인 페이지 쓰기 횟수를 줄일 수 있다. 가비지 컬렉션으로 이주된 페이지의 위치는 PML에 등록하여 해결한다. 가비지 컬렉션은 희생블록의 지우기 연산으로 무효 페이지들이 가용 페이지로 전환되기 때문에 해당 페이지들은 무효 페이지 테이블에서 비트가 '0'으로 설정된다. 이때 해당 페이지가 무효 페이지 테이블과 PML에 모두 등록이 되어 있다면 해당 페이지의 엔트리를 색인 노드에 반영하도록 한다.

5.2 시스템 복구

시스템이 정상적으로 종료되었을 때는 메타데이터를 특정 블록으로 설정된 슈퍼 블록에 저장한다. 메타데이터에는 최상



(a) sequence insertion (b) random insertion (c) elapsed time (S: sequence, R: random)

그림 5. 순차 및 무작위 삽입 시 발생하는 읽기 횟수, 쓰기 횟수, 전체 소요 시간
Fig. 5. Elapsed time and occurred counts of page write/read in insertions of sequence and random

위 색인 노드의 페이지 위치, 무효 페이지 테이블, PML 테이블, 최근에 저장된 노드의 타임스탬프를 저장한다. 순간적인 전원의 차단에 대비하여 각 노드에서는 타임스탬프와 해당 노드가 위치하는 높이 값을 저장한다. 말단 노드는 높이가 0이 되고 최상위 색인 노드의 높이 값은 트리의 높이가 된다. 비정상적인 종료가 감지되면 메타데이터의 타임스탬프를 비교하여 이후의 노드들을 최신 노드로 판단하여 트리를 재구성한다.

IV. 실험환경 및 평가

1. 실험환경

실험 장치는 AllWinner 사의 A20 칩을 사용하는 'Cubieboard 3' 보드를 사용하였다[16]. 해당 보드의 사양은 표 1과 같다. A20 칩은 ARM사의 Cortex A7을 기반으로 하는 듀얼 코어 CPU이다. 실험환경은 리눅스 Lubuntu 12.10을 사용하였으며 커널 3.4.79버전에 구현되어 있는 낸드 플래시 메모리 제어 함수를 수정하였다. 수정된 함수는 시스템 호출을 통해 B+ 트리를 구현하는 프로그램에서 사용된다. B+ 트리의 차수(fanout)값은 128로 설정하여 구현하였다. PML 테이블의 각 엔트리의 메모리 크기는 실험환경의 페이지 수가 2^{20}개이기 때문에 5바이트($2^{20}+2^{20}$)로 설정하였다.

표 1. Cubieboard 3 상세사양
Table 1. Specification of Cubieboard 3

CPU		AllWinner A20
RAM		2GB DDR3@480MHZ
NAND		H27UCG8T2ATR-BC
	전체 용량	8GB
	전체 블록 개수	4096
	블록 당 페이지 개수	256
	페이지 크기	8KB
	지우기 연산 속도	5ms
	페이지 쓰기 속도	1500us
	페이지 읽기 속도	211us

2. 성능평가

키의 삽입에서 발생하는 페이지의 쓰기와 읽기 횟수를 측정하기 위해 3가지의 기법을 적용하여 B+ 트리를 구현하였다. 3가지의 기법은 일반 B+ 트리, μ-트리, 제안 기법(PML)이며 각 기법별로 30만개의 키를 삽입하였다. 키의 값은 순차적인 값으로 삽입하는 것과 무작위의 값으로 삽입하는 2가지의 경우로 구분하였다.

그림 5는 각 기법별로 키 삽입 시 발생하는 페이지의 쓰기 및 읽기 횟수와 전체 소요 시간을 나타내고 있다. 일반 B+ 트리는 업데이트 파생 문제로 인하여 순차 삽입과 무작위 삽입에 모두 가장 많은 페이지 쓰기가 발생 하였다. μ-트리는 삽입 시 발생하는 쓰기 횟수는 적었지만 읽기 횟수는 다른 기법들보다 많았다. μ-트리는 구조상 트리의 높이 크기에 해당

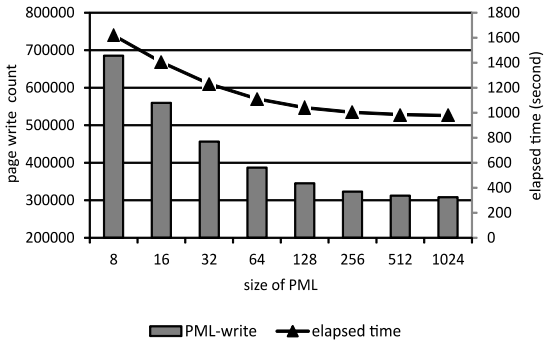


그림 6. PML 테이블 크기에 따른 페이지의 쓰기 횟수와 전체 소요 시간

Fig. 6. A count of page write and elapsed time by each PML size

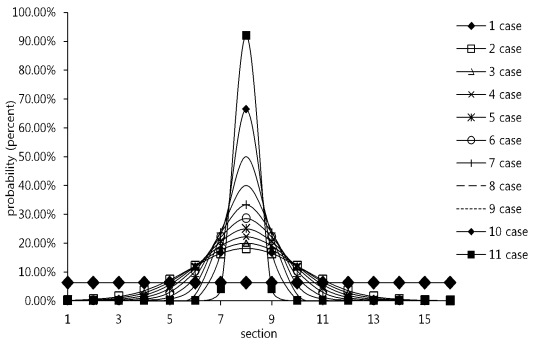


그림 7. 각 상황에 따른 각 구간의 확률 그래프

Fig. 7. Probabilities of each a section with experimental cases

하는 개수의 색인 노드들을 하나의 페이지에 저장할 수 있도록 공간을 분리하여 할당하였고 상위 색인 노드로 갈수록 저장할 수 있는 키의 값은 1/2만큼 반복적으로 줄어든다. 그래서 μ-트리의 읽기 횟수가 많은 이유는 공간 활용성이 낮아 색인 노드는 많아지고 그에 따라 트리의 높이가 커지기 때문이다. 한편 순차 삽입과 무작위 삽입을 비교할 때 읽기 횟수가 순차 삽입이 더 많아졌다. B+ 트리의 기본 구조상 색인 노드의 공간이 부족하게 되면 분할 작업을 통해 2개의 색인 노드로 나누어지고 각 색인 노드에 키가 1/2로 저장이 된다. 그러면 순차 삽입은 트리의 오른쪽에서 지속적으로 분할이 이루어지게 되고 결국 트리의 왼쪽에 위치하는 색인 노드들은 키의 개수가 1/2인 확률이 많아진다. 따라서 μ-트리는 이런 B+ 트리의 기본 구조를 따르고 순차 삽입은 공간 활용성이 낮아지는 것을 더욱 가속화하기 때문에 무작위 삽입보다 순차 삽입에서 페이지 읽기 횟수가 많이 발생한다.

PML을 적용한 트리는 그림 5에서 PML의 엔트리 개수에 따라 128과 1024의 경우로 구분하였다. PML을 적용한 결과 해당 트리는 키 삽입 시 읽기 횟수를 μ-트리에서 발생한 읽기 횟수의 최대 약 61% 줄일 수 있었으며, 쓰기 횟수는 일반 B+ 트리에서 발생한 쓰기 횟수를 최대 약 31% 줄일 수 있었다. 그리고 전체 소요된 시간은 PML를 적용한 트리가 B+ 트리와 μ-트리를 각각 최대 약 50%, 22% 줄일 수 있었다. 무작위 삽입에서 PML을 적용한 트리는 PML의 크기가 128일 때 μ-트리의 쓰기 횟수보다 약 12% 증가하였지만 전체 소요 시간은 최대 약 10% 감소하였다. PML 크기를 고려할 때, 순차 삽입에서 PML을 적용한 트리는 PML 크기가 128인 경우에서도 μ-트리와 동일한 쓰기 횟수를 기록하였다. 이를 자세히 설명하면, 순차 삽입이 무작위 삽입보다 순차 삽입의 경우 키의 값이 순차적으로 큰 키의 값이 삽입된다. 그러면 트리가 구성될 때 트리에서 가장 오른쪽 색인 노드에 삽입이 발생하고 결국 반복적으로 삽입되어 색인 노드의 분할이 발생한다. 노드의 분할이 발생하면 부모 색인 노드가 수정되는데, PML을 적용한 트리는 말단 노드를 제외한 색인 노드의 갱신이 이루어질 때 하위 노드의 위치 정보를 PML 테이블에서 검색하여 변경된 위치가 존재하면 반영하고 해당 엔트리를 테이블에서 삭제한다. 결국 순차 삽입은 트리의 오른쪽이 지속적으로 변경되기 때문에 노드가 분할할 때 갱신되는 상위 노드는 많은 PML 테이블의 엔트리를 반영할 수 있다. 그래서 순차 삽입에서는 작은 PML의 크기로도 추가적인 쓰기의 발생이 없이 트리를 구성할 수 있었다.

그림 6은 30만개의 키를 무작위로 삽입 할 때 PML 테이블의 크기에 따라 발생하는 페이지의 쓰기 횟수와 경과된 시간을 나타내고 있다. 그림 5에서 무작위 삽입 시 발생한 μ-트리와 B+ 트리의 페이지 쓰기 횟수를 그림 6의 결과와 비교하기로 한다. 8개의 엔트리를 가지는 PML 테이블은 μ-트리보다 약 2배 이상의 페이지 쓰기가 더 발생하였다. 이는 엔트리의 개수가 너무 적어서 생긴 결과이다. 하지만 B+ 트리에서 발생한 페이지 쓰기 횟수를 약 33% 줄일 수 있다. 그리고 PML 테이블의 엔트리 개수가 512이상이면 μ-트리보다 많이 발생하는 쓰기 횟수가 약 1% 이내로 줄여 거의 동일한 쓰기 횟수가 발생하였다. 또한 키를 삽입할 때 소요되는 전체 소요 시간을 보았을 때 PML 테이블의 엔트리 개수가 많아질수록 줄어준다는 것을 알 수 있다. 결과적으로 그림 5와 그림 6에서 PML 테이블의 엔트리 개수가 128개 이상일 경우 PML을 적용한 B+ 트리가 읽기 횟수를 줄여 μ-트리보다 전체적인 성능이 더 좋다는 것을 알 수 있다.

트리의 키 갱신 시 PML을 적용한 트리의 성능을 측정하기 위해 30만개의 키가 삽입되어 구성된 트리에서 60만개의 키를 갱신하고 이 때 발생하는 페이지의 쓰기 횟수를 측정하

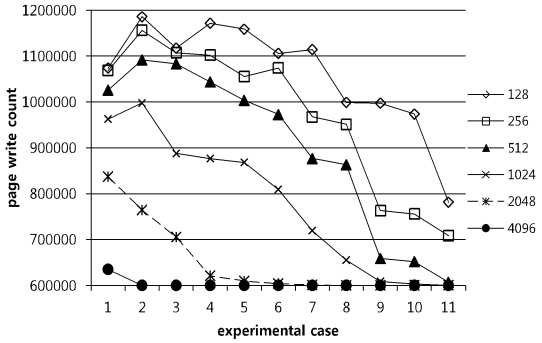


그림 8. PML 크기에 따른 각 상황별 페이지 쓰기 횟수
Fig. 8. Counts of page write with PML size and experimental cases

였다. 갱신이 자주 발생하는 구간을 모델링하기 위해 가우스 정규분포(gaussian normal distribution) 모델을 사용하였으며 편차 값에 따라 11개의 상황을 만들었다. 사용한 가우스 정규분포 모델은 식 1과 같으며 트리를 16개의 구간(x)으로 나누어 각각의 확률밀도(p)를 적용하였다.

$$p = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2}) \qquad \text{식 1}$$

그림 7은 각 상황에 적용된 확률밀도를 나타내고 있다. 각 구간은 30만개의 키를 18750개씩 나누어 균등하게 분배되었다. 11개의 상황 중에 1개는 전 구간이 동일한 확률밀도를 가지도록 하였다. 나머지 10개의 상황은 'σ'의 값을 0.4~2.4까지 적용하였다. 1번 상황에서 11번 상황으로 갈수록 갱신 확률의 분산이 작아지고 있다.

그림 8은 PML 테이블의 엔트리 개수에 따라 각 상황에서 발생하는 페이지 쓰기 횟수를 나타내고 있다. 60만개의 키를 갱신하였을 때 μ-트리는 약 60만 번의 페이지 쓰기가 발생하였고 일반 B+ 트리는 약 180만 번의 페이지 쓰기가 발생하였다. 그림 8에서 PML를 이용한 B+ 트리는 PML 테이블의 크기가 커지고, 갱신되는 키의 분산정도가 작을수록 쓰기 횟수가 적어진다. 갱신되는 키의 분산이 커질수록 쓰기 횟수가 많아져 전체적인 성능이 저하된다는 것을 알 수 있다. 이는 갱신 작업이 말단 노드에서만 이루어지게 되어 상위 색인 노드에 PML 엔트리를 반영할 기회가 적어지기 때문이다. 갱신과 삽입 상황을 비교했을 때, 삽입 시 새로운 레코드의 키로 인해 상위 색인 노드가 변경되고 이를 새로 기록할 때 PML 테이블에 있는 엔트리를 반영하는 기회가 많아지게 된다. 하지만 갱신에서는 색인 노드의 변경이 없기 때문에 PML 테이블의 엔트리가 가득차서 더 이상 등록이 되지 않을 때 상위 색인 노드들이 새로 기록된다. 그리고 이때 PML 테이블에 있는 엔트리가 반영되게 된다. 결국 PML 테이블의 공간을 확보하기 위해 강제로 상위 색인을 기록하면 추가적인 페이지 쓰기가 발생하기 때문에, 갱신은 삽입보다 페이지 쓰기 횟수가 많아진다.

3가지의 기법을 성능을 비교할 때 PML을 적용한 트리는 삽입 시 발생하는 페이지 읽기 횟수를 최대 약 61%로 줄일 수 있었고, 페이지 쓰기 횟수를 최대 약 31%로 줄일 수 있었다. 그리고 전체 소요된 시간은 PML를 적용한 트리가 B+ 트리와 μ-트리를 각각 최대 약 50%, 22% 줄일 수 있었다. 갱신 시 PML을 적용한 트리는 μ-트리보다 페이지 쓰기 횟수가 많이 발생하였지만 여전히 일반 B+ 트리보다 페이지 쓰기 횟수가 적으며, 키가 갱신되는 구간이 국부적인 곳에 집중된다면 μ-트리와 차이가 적어졌다. 결국 제안된 기법은 키의 탐색 비율이 높거나 갱신이 국부적인 곳에 집중된다면 전체적인 성능이 다른 기법보다 높다는 것을 알 수 있다. 또한 PML 테이블의 크기를 가변적으로 변경할 수 있어 가용할 수 있는 메모리의 공간을 많이 할당하면 전체적인 성능이 높아진다.

V. 결 론

본 논문에서는 낸드 플래시 메모리를 기반으로 하는 B+ 트리를 위해 PML 테이블을 이용한 색인 관리 기법을 제안하였다. 하드 디스크를 기반으로 하는 B+ 트리를 낸드 플래시 메모리에 적용할 경우 낸드 플래시 메모리의 특성상 제자리 덮어쓰기가 되지 않아 업데이트 파생 문제를 야기한다. 그로 인해 페이지 쓰기 횟수가 증가하여 B+ 트리의 전체적인 성능을 저하시킨다. 이를 해결하기 위해 본 논문에서 제안된 기법은 B+ 트리의 레코드 연산에서 색인 노드의 갱신으로 변경되는 낸드 플래시 메모리의 페이지 위치를 PML 테이블에 저장하고 탐색 시 저장된 위치를 활용한다. 이를 통해 업데이트 파생 문제에서 발생하는 페이지 쓰기 횟수를 줄이며, 레코드 탐색에서 추가적인 페이지 읽기를 발생하지 않도록 한다. 그리고 제안된 기법은 추가적인 FTL의 구현 없이 적용이 가능하여 경량 데이터베이스를 사용하는 저사양 임베디드 장치나 단순 정보를 지속적으로 기록하는 센서 노드에 활용이 가능하다. 실제 낸드 플래시 메모리가 탑재되어 있는 임베디드 장치에 실험환경을 구축하고 성능 평가를 실시한 결과 레코드의 삽입 시 발생하는 페이지 읽기 횟수를 다른 기법들과 비교할 때 최대 약 61% 줄일 수 있었고, 페이지 쓰기 횟수를 최대 약 31% 줄일 수 있었다.

본 논문의 향후 연구 과제는 다음과 같다. PML 테이블은 단순 일차원 배열의 자료구조를 사용하고 있다. PML의 크기는 시스템의 메인 메모리 공간에 따라 가변적으로 할당할 수 있으나 PML의 크기가 너무 커지게 되면 탐색비용이 증가된다. 이에 탐색비용을 줄이기 위한 PML 테이블의 자료구조와 동작 기법을 추가 개선하는 연구가 필요하다.

REFERENCES

[1] Chung, Tae-Sun, et al. "A survey of flash translation layer." Journal of Systems Architecture, Vol. 55, No. 5, pp. 332-343, May. 2009.

[2] Kwon, Se Jin, et al. "FTL algorithms for NAND-type flash memories." Design Automation for Embedded Systems, Vol. 15. No. 3-4, pp. 191-224, Dec. 2011.

[3] Ma, Dongzhe, Jianhua Feng, and Guoliang Li. "A survey of address translation technologies for flash memories." ACM Computing Surveys (CSUR), Vol. 46, No. 3, pp. 1-39, Jan. 2014.

[4] Kang, Dongwon, et al. "μ-Tree: an ordered index structure for NAND flash memory." Proceedings of the 7th ACM & IEEE international conference on Embedded software. ACM, pp. 144-153, Sep. 2007.

[5] Ahn, Jung-Sang, et al. "μ*-Tree: An Ordered Index Structure for NAND Flash Memory with Adaptive Page Layout Scheme." Computers, IEEE Transactions, Vol. 62, No. 4, pp. 784-797, Apr. 2013.

[6] Na, Gap-Joo, Bongki Moon, and Sang-Won Lee. "In-page logging B^+-tree for flash memory." Database Systems for Advanced Applications. Springer Berlin Heidelberg, pp. 755-758, Apr. 2009.

[7] Na, Gap-Joo, Sang-Won Lee, and Bongki Moon. "Dynamic In-Page Logging for B^+-tree Index", Knowledge and Data Engineering, IEEE Transactions on, Vol. 24, No. 7, pp. 1231-1243, Jul. 2012.

[8] Fang, H., et al. "An Adaptive Endurance-Aware B+-Tree for Flash Memory Storage Systems." Computers, IEEE Transactions, Vol. 63, No. 11, pp. 2661-2673, Nov. 2014.

[9] Wu, Chin-Hsien, Tei-Wei Kuo, and Li Ping Chang. "An efficient B-tree layer implementation for flash-memory storage systems." ACM Transactions on Embedded Computing Systems (TECS), Vol. 6, No. 3, pp. 19-es, Jul. 2007.

[10] Agrawal, Devesh, et al. "Lazy-adaptive tree: An optimized index structure for flash devices." Proceedings of the VLDB Endowment, Vol. 2, No. 1, pp. 361-372, Aug. 2009

[11] Nath, Suman, and Aman Kansal. "FlashDB: dynamic self-tuning database for NAND flash." Proceedings of the 6th international conference on Information processing in sensor networks. ACM, pp. 410-419, Apr. 2007.

[12] Sanghun Yun, Haengrae Cho. "An Embedded Text Index System for Mass Flash Memory", Journal of KSCI, pp. 1-10, Vol. 14, No. 6, Jun. 2009.(in Korean)

[13] Li, Xiang, Zhou Da, and Xiaofeng Meng. "A new dynamic hash index for flash-based storage." Web-Age Information Management, 2008. WAIM'08. The Ninth International Conference on. IEEE, pp. 93-98, Jul. 2008.

[14] Sang-Ho Hwang, Jong Wook Kwak. "CL-Tree: B+ tree for NAND Flash Memory using Cache Index List", Journal of KSCI, pp. 1-10, Vol. 20, No. 4, Apr. 2015.(in Korean)

[15] Bityuckiy, Artem B. "JFFS3 design issues." Memory technology device (MTD) subsystem for Linux, 2005.

[16] Hardware spec, http://docs.cubieboard.org/tutorials/cubietruck/start.

저 자 소 개

김 선 환

2006: 영남대학교
컴퓨터공학과 공학사.
2009: 영남대학교
컴퓨터공학과 공학석사.
현 재: 영남대학교
컴퓨터공학과 박사과정.
관심분야: 임베디드 시스템,
무선 센서 네트워크
Email : amexist@ynu.ac.kr

곽 종 욱

1998: 경북대학교
컴퓨터공학과 공학사.
2001: 서울대학교
컴퓨터공학과 공학석사.
2006: 서울대학교
전기컴퓨터공학부 공학박사
현 재: 영남대학교
컴퓨터공학과 부교수
관심분야: 컴퓨터 구조,
고성능 컴퓨팅,
임베디드 시스템
Email : kwak@yu.ac.kr