뉴메모리를 저장 장치로 하는 파일 시스템에서 데이터 관리를 위한 자료구조에 관한 실험적 고찰

Experimental Evaluation of File System Data Structures for New Memory based Storage

저자
(Authors)

이세권, 문영제, 송현섭, 노삼혁
Se Kwon Lee, Young Je Moon, Hyunsub Song, Sam H. Noh

출처
(Source)

한국정보과학회 학술발표논문집 , 2016.6, 1475-1477 (3 pages)

발행처
(Publisher)

한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL

http://www.dbpia.co.kr/Article/NODE07017878

APA Style

이세권, 문영제, 송현섭, 노삼혁 (2016). 뉴메모리를 저장 장치로 하는 파일 시스템에서 데이터 관리를 위한 자료구조에 관한 실험적 고찰. 한국정보과학회 학술발표논문집, 1475-1477.

뉴메모리를 저장 장치로 하는 파일 시스템에서 데이터 관리를 위한 자료구조에 관한 실험적 고찰

이세권O 문영제 송현섭 노삼혁
울산과학기술원 전기전자컴퓨터공학부
{sekwonlee, yjmoon, hssong1987, samhnoh}@unist.ac.kr

Experimental Evaluation of File System Data Structures for New Memory based Storage

Se Kwon LeeO, Young Je Moon, Hyunsub Song, Sam H. Noh
UNIST, School of Electrical and Computer Engineering

요 약

뉴메모리는 DRAM과 같이 바이트 단위의 임의 접근이 가능하고 높은 접근 속도를 가진다. 또한 기존의 저장 장치와 같이 비휘발성이기 때문에, 차세대 저장 장치로 주목을 받고 있다. 뉴메모리를 저장 장치로 활용하는 파일 시스템에서 데이터를 관리하기 위한 자료구조의 선택은 빠른 접근 속도를 가지는 뉴메모리의 성능을 최대한으로 활용하는데 중요한 요인이 될 수 있다. 이에 따라 본 논문에서는 뉴메모리 전용 파일 시스템에서 자료구조가 전체 파일 시스템 성능에 미치는 영향을 알아보기 위해, 아이노드 테이블 인덱싱 구조에 radix 트리, B+ 트리, 직접 매핑 테이블의 세 가지 계열의 자료구조들을 적용한 후 실험을 통해 비교 분석한다.

1. 서 론

차세대 메모리 기술인 뉴메모리는 기존 DRAM과 같이 바이트 단위의 임의 접근이 가능하고, 나노 초단위의 높은 접근 속도를 가질 뿐 아니라, 비휘발성의 특징을 가지기 때문에 고성능 저장 장치로서의 활용 가능성을 가지고 있다 [1]. 기존의 디스크나 플래시 메모리 기반의 저장 장치를 위한 파일 시스템과 다르게, 뉴메모리 전용 파일 시스템은 DRAM처럼 빠르고 바이트 단위로 접근이 가능한 뉴메모리를 저장 장치로 이용하기 때문에, 버퍼링을 위한 페이지 캐시와 블록 I/O 계층이 사라진다. 이로 인하여 상대적으로 파일 시스템에서 데이터를 관리하는 자료구조에 의한 오버헤드가 파일 시스템 전체 성능에 큰 영향을 끼치게 된다. 자료구조에 의한 오버헤드는 알고리즘 효율로 인한 오버헤드 뿐 아니라, 알고리즘 효율을 위해 존재했던 키 값의 정렬이나 rebalancing 등으로 인해 발생할 수 있는 CPU 캐시 미스의 발생빈도를 포함한다.

본 논문에서는 파일 시스템 데이터의 인덱싱 자료구조에 초점을 맞추며, 오픈 소스인 PMFS (Persistent Memory File System) [2, 3] 을 이용하여, 이 시스템의 인덱싱 구조에 여1)러 자료구조들을 적용해 본다. 이 중 기존 PMFS의 아이노드 테이블들을 인덱싱하는 B 트리 [2] 구조를 페이지 캐시의 radix 트리, 일반 B+ 트리, CSB+ 트리, 직접 매핑 테이블로 바꿔 적용해보고 이들의 성능을 다양한 워크로드에서의 실험을 통해 측정한다.

실험 결과 자료구조로의 데이터 삽입이 발생하는 워크로드에서 직접 매핑 테이블이 가장 좋은 성능을 나타냈다. 또한 데이터 탐색만 발생하는 워크로드에서 자료구조를 구성하는 데이터의 용량이 CPU 캐시의 용량에 비해 작을 경우 자료구조들의 탐색 오버헤드가 파일 시스템의 성능에 미치는 영향은 CPU 캐시에 의해 상쇄될 수 있음을 알 수 있었다.

* 이 논문은 삼성전자 미래기술육성센터의 지원을 받아 수행된 연구임 (과제번호 SRFC-IT1402-09)

2. 배경 지식 및 관련 연구

뉴메모리는 차세대 메모리 기술로서 바이트 단위의 접근이 가능하고 비휘발성의 특징을 가지고 있다. 뉴메모리의 종류로는 PCM (Phase Change Memory), STT-MRAM (Spin Transfer Torque MRAM), 그리고 ReRAM (Resistive RAM)이 있다. 이들의 read, write 접근 속도는 NAND 플래시 메모리에 비해 월등히 뛰어나면서 DRAM에 준하는 성능을 보여주고 있다. 그리고 PCM과 ReRAM의 경우 DRAM에 비해 높은 집적도를 가지고 있다 [1]. 뉴메모리의 이러한 특성들은 기존의 저장 장치인 디스크와 플래시 메모리를 대체할 차세대 저장 장치로서 뉴메모리가 활용될 수 있다는 주장의 근거가 된다.

In-memory 데이터베이스 환경에서 인덱싱 자료구조에 관한 연구는 뉴메모리와 유사한 특징을 가지는 DRAM에 저장되어 동작하는 자료구조의 성능을 최적화하는 방향으로 진행되어 왔다는 점에서 본 논문과 상당한 연관성을 가지고 있다. 이들의 연구는 CPU 캐시와 DRAM간의 성능 차이로 인해 발생할 수 있는 오버헤드를 줄이는 것을 목표로 한다. 블록 기반 저장매체에서 주로 사용되던 B+ 트리의 노드를 CPU 캐시 라인 크기에 맞추고 노드 내에 포인터의 개수를 줄여 노드 당 보유할 수 있는 키값의 수를 증가시킨다. 그리고 트리의 같은 레벨에 속하는 노드들을 연속된 메모리 영역에 배치하여 B+ 트리의 CPU 캐시 효율성을 향상시켰다 [4, 5]. 본 논문에서는 In-memory 데이터베이스인 SAP HANA에서 인덱싱 구조로서 사용되고 있는 CSB+ 트리를 실험 비교군 중 하나로 사용한다 [4].

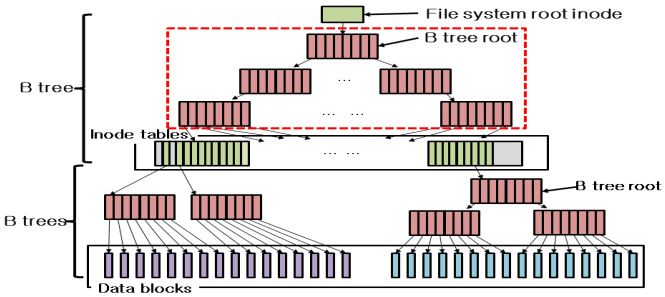

그림 1 PMFS의 구조

3. 실험 설계 및 구현

3.1 인덱싱 자료구조 계층 선정

우리는 뉴메모리 전용 파일 시스템인 PMFS [2, 3]를 실험을 위한 플랫폼으로 이용한다. 그림 1에서 볼 수 있듯이 PMFS의 인덱싱 구조는 크게 두 개의 B 트리들로 이루어져 있다. 위쪽 B 트리는 파일 시스템 루트 아이노드로부터 아이노드 테이블들에 접근하기 위한 인덱싱 구조이다. 그리고 아래쪽 B 트리들은 아이노드 테이블내의 특정 아이노드에 속하는 파일 혹은 디렉토리 블록들을 인덱싱하는 역할을 담당하고 있다.

우리는 그림 1의 PMFS 구조에서 점선으로 표시된 아이노드 테이블들을 인덱싱하는 위쪽 B 트리를 페이지 캐시 radix 트리, 일반 B+ 트리, CSB+ 트리, 직접 매핑 테이블로 바꿔본다. 파일 시스템에서 임의의 파일에 접근하기 위해선, 해당 파일에 대한 정보를 가지고 있는 아이노드가 속한 아이노드 테이블을 먼저 탐색해야한다. 따라서 임의의 파일로의 접근이 필요한 모든 연산들은 이 부분의 탐색 과정을 필수적으로 거치게 된다. 때문에 워크로드에 상관없이 자료구조의 변화에 따른 파일 시스템 전체 성능의 변화를 관찰하기에 용이하다고 판단하여, 다양한 자료구조들을 적용할 지점으로 아이노드 테이블 인덱싱 구조를 선택하게 되었다. 여기서 자료구조에 입력되는 키값은 아이노드 테이블의 번호가 된다.

3.2 자료구조들의 구현

PMFS에서 아이노드 테이블 인덱싱 관련 함수들은 *pmfs_alloc_blocks()*, *pmfs_get_inode()*가 있다. *pmfs_alloc_blocks*는 인덱싱 자료구조에 새로운 아이노드 테이블을 삽입할 때 호출되는 함수이다. *pmfs_get_inode*는 특정 아이노드 테이블을 찾을 때 호출되는 함수이다. 우리는 두 함수에 새롭게 적용할 자료구조들의 삽입과 탐색 연산들을 구현하였다.

트리 계열의 자료구조들은 노드의 크기를 CPU 캐시 라인의 크기에 가깝게 할수록 CPU 캐시 사용성은 높아지지만, 노드의 크기가 작아지면 트리의 높이가 높아지기 때문에, 트리를 탐색하는 오버헤드가 증가하는 트레이드오프를 가지고 있다 [5]. 따라서 우리는 구현 시에 다양한 연산들이 섞여있는 Filebench의 fileserver 워크로드에서 가장 좋은 성능을 보이는 노드 크기를 선정하여 트리들을 구현했다. 구현된 각 트리 노드의 크기는 PMFS B 트리 4KB, 페이지 캐시 radix 트리 2KB, 일반 B+ 트리 256byte, CSB+ 트리 128byte이다.

직접 매핑 테이블의 경우 전체 뉴메모리 용량을 최대로 사용했을 때 필요한 최대 아이노드 테이블의 개수를 고려하여, 이를 모두 인덱싱하기 위한 매핑 테이블의 크기를 정적으로 할당했다. 이때 매핑 테이블의 각 인덱스 번호와 키값인 아이노드 테이블 번호는 동일하기 때문에 직접 매핑이 가능하다. 따라서 복잡한 탐색 알고리즘 없이, 키값과 일치하는 매핑 테이블의 인덱스에 바로 접근하는 알고리즘만으로 구현할 수 있었다.

표 1 워크로드들의 특성

Workload	Avg. file size	#threads	Read ratio	Write ratio
Createfiles	16KB	16	0%	100%
Fileserver	128KB	50	30%	70%
Webproxy	16KB	100	80%	20%
Webserver	16KB	100	90%	10%

4. 실 험

4.1 실험 환경 및 구성

실험은 Intel Xeon E5-2620 2.4GHz CPU, 15MB CPU LLC(Last Level Cache), 64byte CPU 캐시 라인 크기, 128GB DRAM, 64bit CentOS 6.6 (리눅스 커널 버전 3.11.0)에서 수행된다. 뉴메모리 저장 장치 환경을 시뮬레이션하기 위해 DRAM의 100GB를 PMFS 영역으로 마운트한다. 성능 비교를 위해 표 1에 나타낸 Filebench에서 제공하는 워크로드들을 이용한다.

PMFS에 적용한 자료구조들을 radix 트리 계열 (PMFS B트리, 페이지 캐시 radix 트리), B+ 트리 계열 (일반 B+ 트리, CSB+ 트리), 직접 매핑 테이블의 세 가지로 크게 분류하고 실험 분석을 진행한다. PMFS의 저자인 S. R. Dulloor는 PMFS에서 아이노드 테이블을 관리 하고 있는 구조가 B 트리로 구현되었다고 언급하고 있다 [2]. 하지만 실제 코드에서의 분석 결과, 키값을 구성하고 있는 비트들이 나타내는 값을 통해 자식 노드를 가리키는 포인터의 위치를 결정하는 기존의 페이지 캐시의 radix 트리와 유사한 방법으로 트리가 구현되어 있었다. 따라서 우리는 PMFS B 트리를 radix 트리 계열로 분류한다.

4.2 실험 분석

그림 2 (a), (b), (c)와 표 1을 살펴보면, 파일 개수가 많은 환경과 write 비율이 높을수록 즉, 자료구조로의 삽입 연산이 많을수록, 자료구조들 간의 성능 차이가 커진다는 것을 알 수 있다. 이들 중 비교적 높은 성능을 보인 직접 매핑 테이블과 PMFS B 트리는 파일 개수 256만개의 createfiles와 64만개의 fileserver에서 각각 9%, 5%의 성능 차이를 보였다.

B+ 트리 계열의 자료구조는 키값이 삽입 될 때 발생하는 노드 내에서의 키값들의 정렬이나 트리의 균형을 맞추기 위한 rebalancing으로 인한 알고리즘적 오버헤드 뿐 아니라, 트리 노드가 가지고 있는 데이터와 트리 구조의 잦은 변화가 캐시 미스를 많이 발생시켜 성능 하락의 원인이 된다. 하지만 radix 트리 계열의 자료구조들의 경우 B+ 트리에서 발생하는 키값의 정렬과 rebalancing이 발생하지 않는다. 따라서 이로 인한 알고리즘적 오버헤드가 없고 새로운 노드가 생성될 때를 제외하면 트리 구조에 변화가 발생하지 않는다. 그러므로 B+ 트리에 비해 캐시 미스를 덜 발생시키게 되어 그림 2 (a), (b), (c)에서 더 좋은 성능을 나타낸다.

직접 매핑 테이블의 경우 키값으로 들어오는 아이노드 테이블의 번호와 매핑 테이블의 인덱스 번호가 같다. 따라서 트리 자료구조에서와 같이 키값을 탐색하기 위한 알고리즘적 오버헤드

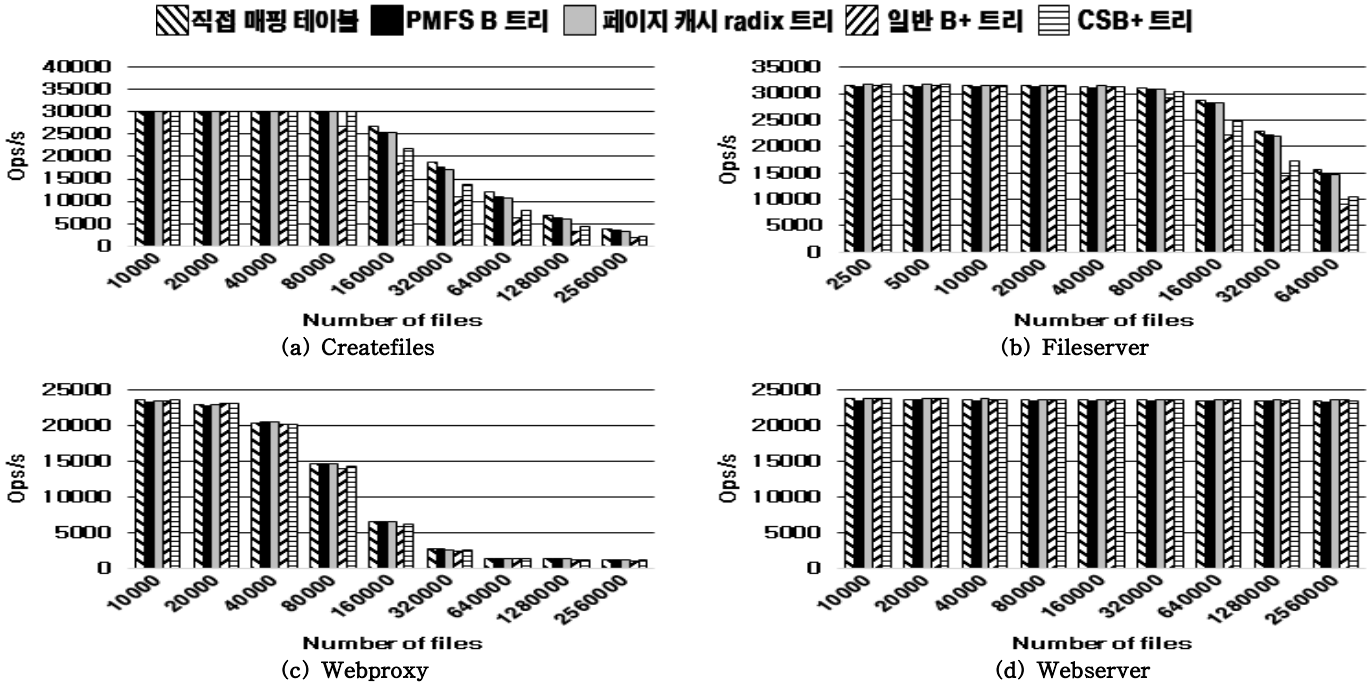

그림 2 Filebench의 워크로드들을 이용한 자료구조들 간의 성능 비교

가 발생하지 않는다. 또한 선형적으로 할당되어 있기 때문에 좋은 캐시 효율성을 가지게 된다. 때문에 그림 2 (a), (b), (c)의 워크로드들에서 평균적으로 가장 좋은 성능을 보이고 있다.

그림 2 (d)의 webserver 결과를 보면 파일 개수에 상관없이 모든 자료구조의 성능이 비슷하게 나타났다. 다른 세 워크로드들과 다르게 webserver에서의 write는 새로운 파일을 생성한 후 write를 수행하는 것이 아닌 기존에 생성된 파일에 append만 일으키는 연산을 수행한다. 따라서 아이노드 테이블 인덱싱 구조에서 새로운 아이노드를 삽입하는 연산은 발생하지 않고 탐색만 이루어지게 된다. 그러므로 webserver에서는 삽입 연산에 의한 자료구조들 간의 성능 차이는 발생하지 않게 된다.

그림 3은 webproxy와 webserver에서 파일 개수 64만개일 때의 커널에서 발생하는 캐시 미스율을 perf 툴 [6] 을 이용해 측정한 결과이다. Webproxy의 경우 write 비율이 크지 않음에도 불구하고 전체적으로 50%이상의 캐시 미스가 발생했고 webserver의 경우 모든 자료구조가 1% 이하로 캐시 미스가 발생했다. Webserver에서 캐시 미스율이 상당히 낮은 이유는 우리가 실험하고 있는 환경인 15MB CPU LLC에 자료구조들의 데이터 대부분이 캐시 되었기 때문이다. 이를 통해 그림 3의 webserver와 같이 자료구조의 탐색 연산만 발생하는 워크로드에서 자료구조의 탐색에 의한 오버헤드는 CPU 캐시에 의해 상쇄될 수 있음을 알 수 있다.

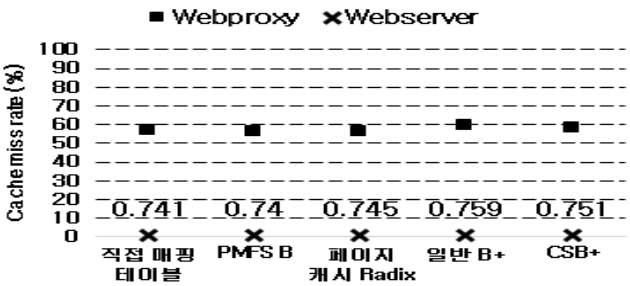

그림 3 Webproxy와 Webserver간의 캐시 미스율 차이

5. 결 론

본 논문에서는 뉴메모리 전용 파일 시스템에서 자료구조의 오버헤드가 파일 시스템 전체 성능에 미치는 영향에 대해 알아보았다. 자료구조에 삽입 연산이 많이 발생하는 워크로드에서 직접 매핑 테이블이 가장 좋은 성능을 보였는데 아이노드 테이블을 인덱싱하는 부분과 같이 하나의 자료구조만을 필요로 하는 영역의 경우 직접 매핑 테이블을 통해 관리하는 것이 가장 좋은 선택이 될 수 있다. 탐색 연산만 발생하는 워크로드에서 자료구조가 뉴메모리에서 차지하는 용량이 CPU LLC의 용량보다 작을 경우 각 자료구조의 탐색 오버헤드가 파일 시스템의 성능에 미치는 영향은 CPU 캐시에 의해 상쇄될 수 있다.

향후 연구에서는 본 논문에서 다루지 않았던 파일 및 디렉토리 블록, 디렉토리 블록내의 디렉토리 엔트리들의 인덱싱 구조에 관한 연구를 진행할 계획이다. 그리고 뉴메모리 전용 파일 시스템의 전체적인 인덱싱 구조에 대해서 논할 계획이다.

참 고 문 헌

[1] K. Suzuki and S. Swanson. “A Survey of Trends in Non-Volatile Memory Technologies: 2000-2014.” In IMW, 2015.

[2] S. R. Dulloor, et al. “System software for persistent memory.” In EuroSys, 2014.

[3] PMFS source code. https://github.com/linux-pmfs/pmfs

[4] J. Rao and K. A. Ross. “Making B+-trees cache conscious in main memory.” In SIGMOD, 2000.

[5] R. A. Hankins and J. M. Patel. “Effect of node size on the performance of cache-conscious B+-trees.“ In SIGMETRICS, 2003.

[6] Perf Wiki. https://perf.wiki.kernel.org