논문 2008-45SD-3-12

휴대용 저장장치 시스템을 위한 Clustered Flash Translation Layer

(A Clustered Flash Translation Layer for Mobile Storage Systems)

박 광 희*, 김 덕 환**

(Kwanghee Park and Deok-Hwan Kim)

요 약

컴팩트 플래시 메모리와 같은 휴대용 저장장치 표준에서는 플래시 메모리 시스템 소프트웨어인 FTL(Flash Translation Layer)이 필요하다. 본 논문에서는 논리 주소를 물리 주소로 빠르게 변환하기 위해 Clustered Hash Table과 2단계 소프트웨어 캐시 기법을 사용하여 FTL을 설계하였다. 실험 결과 본 논문에서 제안한 CFTL이 잘 알려진 NFTL과 AFTL보다 각각 13%, 8% 이상 주소 변환 성능이 빠르고 AFTL보다 메모리 사용량을 75% 이상 감소시켰다.

Abstract

It is necessary to develop the flash memory system software FTL(Flash Translation Layer) which is used in mobile storage like CompactFlash memory. In this paper, we design the FTL using clustered hash table and two phase software caching method to translate logical address into physical address fastly. The experimental results show that the address translation performance of CFTL is 13.3% higher than that of NFTL and 8% higher than that of AFTL, and the memory usage of CFTL is 75% smaller than that of AFTL.

Keywords: FTL, NAND Flash Memory, Clustered Hash Table, Software TLB

I. 서 론

NAND 플래시 메모리는 현재 임베디드 시스템에서 가장 선호하는 저장장치로 각광 받고 있다. 하지만 NAND 플래시 메모리를 저장장치로 사용하기 위해서는 극복해야 하는 NAND 플래시 메모리의 특징들이 있다[4]. 하드디스크와 달리 제자리 쓰기(In-place update)가 불가능하고, 쓰기 전 삭제 연산(Erase before write)을 먼저 수행해야 하며 각 블록 별로 쓰기 횟수(Wear level count)에 한계가 있다. 그렇기 때문에 일반 운영체제의 파일 시스템과 같이 자기 디스크의 속성에 적합하게 설계되어 있을 경우 NAND 플래시 메모리의 특성에 따른 연산을 할 수 있는 FTL(Flash Translation Layer)이라는 미들웨어가 필요하다.

현재 NAND 플래시 메모리의 용량이 급격히 증가함에 따라 FTL의 성능이 플래시 메모리의 데이터의 처리 속도 및 수명에 많은 영향을 끼친다. 현재 몇 가지 방식의 뛰어난 성능의 FTL이 존재하는데, 그 중 Intel에서 설계한 FTL과 M-Systems의 NFTL(NAND FTL), 그리고 대만 국립 대학교에서 FTL과 NFTL의 주소 변환 기법을 혼용하는 AFTL(Adaptive FTL)이 가장 널리 알려져 있다.

FTL은 페이지 단위의 최소 단위 주소 변환 기법을 사용하여, 주소 변환 테이블을 선형 테이블로 설계 했

* 학생회원-제1저자, ** 정회원-교신저자, 인하대학교 전자공학과
(Dept. of Electronic Engineering, Inha University)
※ 본 논문은 정보통신부 출연금으로 ETRI, SoC산업 진흥센터에서 수행한 IT SoC 핵심설계인력양성사업의 연구결과입니다.
접수일자: 2007년9월10일, 수정완료일: 2008년2월27일

고, 512Byte 페이지 단위의 주소들의 하드웨어 주소 정보를 포함하고 있으므로 주소 변환 속도가 빠르지만 주소 변환 테이블의 크기가 크다는 단점이 있다[5~6].

NFTL은 대용량 장치의 NAND 플래시 메모리를 지원하기 위해 페이지의 집합인 16KB의 블록 단위의 주소 변환 기법을 사용하며, 이에 따라 주소 변환 테이블의 크기가 FTL보다 상대적으로 작지만 블록 단위의 주소 변환 테이블 검색과 함께 블록 내의 페이지 단위 검색이라는 오버헤드가 존재한다.

AFTL은 FTL과 NFTL에서 빠른 주소 변환과 작은 주소 변환 테이블을 장점으로 살리기 위해 자주 쓰이는 블록들을 Fine-Grained Hash Table이라는 곳에 따로 저장하여 페이지 단위 주소 변환 기법을 사용하고, 나머지 블록들은 Coarse-Grained Hash Table을 통해 블록 단위의 주소 변환을 한다. 또한 LRU(Least Recently Used) 기법에 따라 Fine-Grained Hash Table과 Coarse-Grained Hash Table 간의 블록 정보를 교체하는 정책을 제시했다. 하지만 AFTL의 주소 변환 성능이 NFTL과의 실험결과에서 별다른 차이를 보이지 않았다. 또한 기존의 해시 테이블 기법을 사용함으로 인해 200% 정도의 메모리 오버헤드를 가지고 있으며, 대용량의 플래시 메모리에서 사용 시에 부적합하다. 또한 LRU는 구현 시에 오버헤드가 존재하기 때문에 FTL 성능에 영향을 줄 수 있다[1].

본 논문에서는 운영체제에서 메모리 주소 변환에서 사용하는 해시 테이블이 메모리를 과도하게 사용하고 주소가 증가 시에 충돌이 일어나는 버켓들이 많아지므로 검색 성능에 영향을 주는 것을 방지하기 위해 Clustered Hash Table 기법을 사용하였다[2]. 그리고 최근 LRU 기법 중 오버헤드가 적고 좋은 성능을 나타내는 NUR(Not Used Recently) 기법을 구현하여 Coarse-Grained Hash Table과 Fine-grained clustered hash table 간의 주소 정보 교체 시에 탁월한 성능을 제공한다. 또한 Fine-grained clustered hash table의 적중률 증가를 위해 Miss Rate를 줄이기 위한 멀티 레벨 Fine-grained clustered hash table을 설계 및 구현하였다.

본 논문의 순서는 먼저 Ⅱ장에서 관련 연구에 대해 소개하고 Ⅲ장에서 제안하는 CFTL(Clustered FTL)의 설계 및 구현에 대해 설명하고 Ⅳ장에서는 각 FTL과 CFTL의 성능을 비교한다. 마지막으로 Ⅴ장에서 결론을 맺는다.

Ⅱ. 관 련 연 구

본 장에서는 현재 Intel에서 설계한 FTL과 M-Systems에서 설계한 NFTL 그리고 FTL과 NFTL을 혼합하여 설계한 대만 국립 대학교의 AFTL에 관해 설명한다.

1. FTL(Flash Translation Layer)

FTL은 Intel에서 초기 EEPROM 형태의 플래시 메모리에 데이터를 쓰기 위해서 설계되었다.

NAND 플래시 메모리는 쓰기 전 삭제 연산을 선행해야 하는데, 삭제 단위는 블록 단위로 쓰기 단위인 페이지 단위보다 크고 수행시간이 오래 걸린다. 그러므로 제자리 쓰기를 위해서 동일한 영역에 쓰는 방법은 많은 오버헤드를 가지고 있다. 또한 각 블록은 쓰기 횟수가 한계가 있으므로 쓰기 횟수 평준화가 필요하다. 이러한 단점을 극복하기 위해 주소 변환 테이블을 메인 메모리나 SRAM에 적재하고 Out-place update 시에 물리적 주소만을 변경하는 기법을 제시하였다. 또한 논리적 메모리에서 바로 페이지 단위의 주소 변환 기법을 사용하기 때문에 속도가 빠르다. 하지만 각 페이지 별로 엔트리가 존재하므로 주소 변환 테이블의 크기가 매우 크다는 단점이 있다.

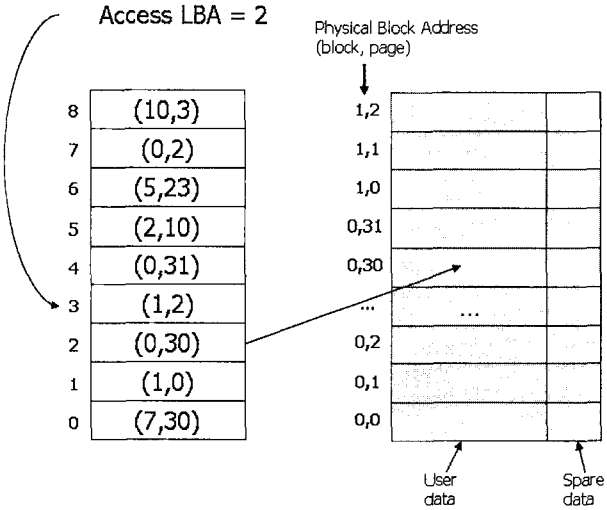

그림 1. Intel FTL의 주소 변환 방법
Fig. 1. Address translation scheme of Intel FTL.

2. NFTL(NAND Flash Translation Layer)

NFTL은 M-Systems사에서 설계한 NAND 플래시 메모리 전용 FTL로 위에서 설명한 FTL의 단점인 주소 변환 테이블의 크기를 줄이기 위해 블록 단위의 주소

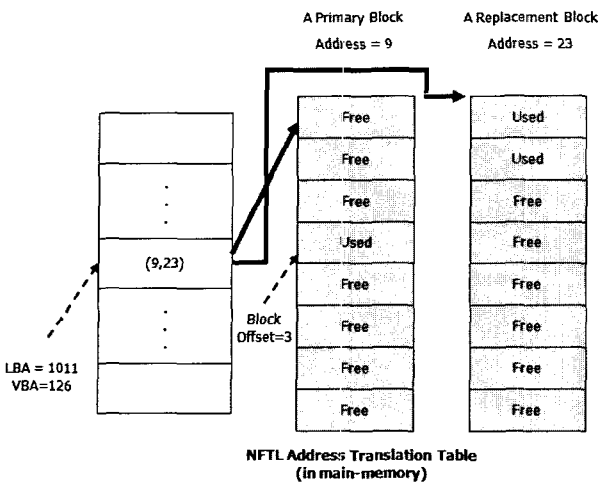

그림 2. NFTL의 주소 변환 방법
Fig. 2. Address translation scheme of NFTL.

변환 방식을 사용한다. 그림 2와 같이 LBA(Logical Block Address)의 값을 블록 당 페이지 개수로 나누어 몫은 VBA(Virtual Block Address)로 사용하고 나머지 값은 Offset을 인덱스로 사용한다. 그리고 주소 변환 테이블로 접근 시 VBA의 값을 통해 Primary 블록과 Replacement 블록의 주소를 가지고 있는 엔트리에 접근한다. 그 후에 주소 변환 테이블에 따라 Primary 블록에 접근 시 Offset으로 페이지 단위 접근을 시도한다. 하지만 Primary 블록의 데이터가 유효하지 않을 시에는 Replacement 블록의 주소로 다시 접근하여 각 페이지를 순차적으로 검색하는 기법을 사용한다. 이를 일컬어 블록 단위 주소 변환 기법이라고 한다. 블록 단위 주소 변환 기법은 주소 변환 테이블의 크기가 작은 대신 Replacement 블록에 데이터가 존재 할 때 순차적으로 주소를 다시 검색해야 하는 오버헤드가 존재한다.

3. AFTL(Adaptive Flash Translation Layer)

AFTL은 대만 국립 대학교에서 설계한 FTL로써 Coarse-Grained, Fine-Grained Hash Table 이렇게 두 개의 주소 변환 테이블로 이루어져 있다. Coarse-Grained Hash Table에서는 NFTL과 같이 블록 단위의 주소 변환 기법을 사용하고, Fine-Grained Hash Table에서는 FTL과 같은 페이지 단위의 주소 변환 기법을 사용한다. 페이지 단위의 주소 변환 기법을 사용하는 경우는 Replacement 블록의 데이터가 Valid 상태일 경우 이 데이터들이 앞으로 자주 쓰일 수 있다고 가정하여 Fine-Grained Hash Table에 저장하여 페이지 단위로 주소를 변환한다. Primary Block일 경우는 Offset을 이용하여 바로 찾기 때문에 Coarse-Grained Hash

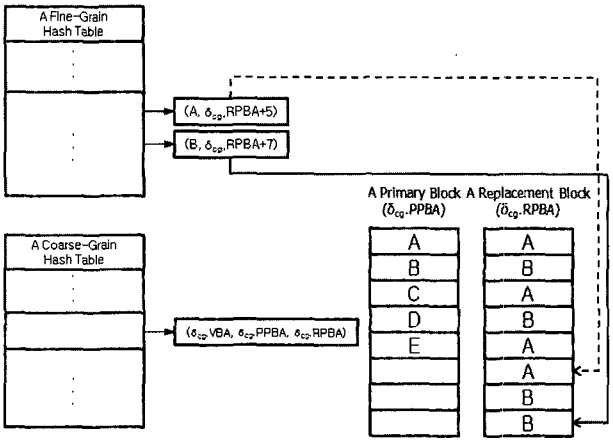

그림 3. AFTL의 주소 변환 방법
Fig. 3. Address translation scheme of Intel AFTL.

Table을 사용한다. Fine-Grained Hash Table의 크기의 제한이 있으므로 Fine-to-Coarse 간의 교체 기법이 필요한데 이때 사용하는 기법이 LRU를 적용하여 일정 기간 동안 사용하지 않는 엔트리들을 Coarse-Grained Hash Table로 이동하는 기법을 사용한다. AFTL은 Replacement 블록에 있는 Valid 데이터들을 페이지 단위 주소 변환 기법을 사용하고 참조 되는 페이지들로 인해 Garbage Collection 작업 시 블록 수거 작업을 임의적으로 지연시켜 오버헤드를 줄인다. 하지만 실험 결과를 살펴보면 페이지 단위의 주소 변환 기법과 블록 단위 주소 변환을 혼용했음에도 불구하고 주소 변환 성능이 NFTL과 별다른 차이가 없다는 것이 단점이다.

III. CFTL의 설계 및 구현

본 논문에서 제안하는 Clustered FTL(CFTL: Clustered Flash Translation Layer)은 일반 해시 테이블 보다 메모리 사용량이 적고 대용량 주소 변환 시에 성능이 뛰어난 Clustered Hash Table을 적용한다. 그리고 AFTL과 같이 블록, 페이지 단위의 주소 변환 테이블 기법을 혼용하면서 Fine-grained clustered hash table을 2단계로 설계하여 캐시의 적중률을 증가 시킨다. 또한 적중률의 성능을 높이기 위해 메모리 페이지 교체 기법 중 성능이 좋고 오버헤드가 적은 것으로 알려진 NUR 알고리즘을 사용하여 새로운 Fine-to-Coarse 교체 기법을 제안한다.

1. Overview

그림 4는 CFTL의 주소 변환 방법의 전체 구조를 설명한다. 일반적으로 FTL의 주소변환 방법은 주어진

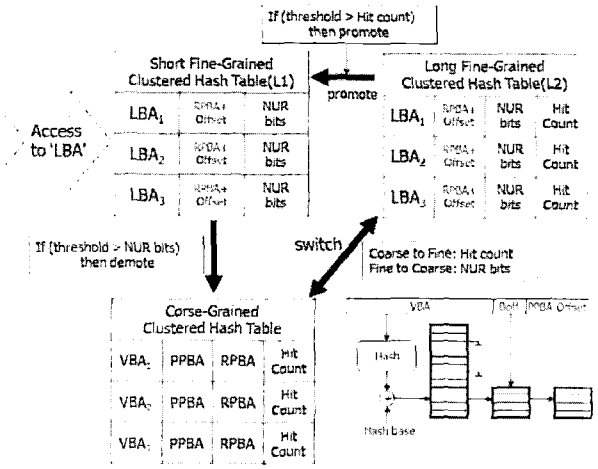

그림 4. CFTL의 주소 변환 기법
Fig. 4. Address translation scheme of CFTL.

LBA(Logical Block Address)를 가지고 PBA(Physical Block Address)로 변환하여 데이터의 물리적인 플래시 메모리 주소를 제공하는데 특정 LBA를 요청 시 가장 먼저 Short fine-grained clustered hash table에서 검색을 하고 LBA에 대한 데이터가 존재하지 않을 시 그 다음 Long fine-grained clustered hash table을 검색한다. 하지만 Long fine-grained clustered hash table에도 존재하지 않을 시에는 Coarse-grained clustered hash table에서 최종적으로 호스트 시스템이 요구하는 LBA에 의한 주소 변환 정보를 찾는다.

2단계 Fine-grained clustered hash table에서 검색할 시에는 페이지 단위 주소 변환 기법을 이용 LBA를 바로 PBA로 변환이 가능하지만 Coarse-grained clustered hash table에서 검색할 시에는 NFTL과 마찬가지로 블록 단위 주소 변환 기법을 위해 VBA(Virtual Block Address)와 Block offset으로 변환하고, Coarse-grained clustered hash table에서 Primary Block에서 해당하는 VBA를 검색하게 되는데 해당하는 VBA가 없을 시에는 Replacement Block에서 Block offset을 이용하여 요구하는 PBA를 검색한다. Boff은 연결된 버켓들의 순서 값을 나타내고 offset은 버켓 안의 Subblock의 순서를 나타낸다. 이를 통해 검색된 PPBA(Primary Physical Block Address)와 RPBA (Replacement Physical Block Address)를 찾아 주소를 변환하게 된다.

2. Clustered Hash Table

Clustered Hash Table은 과거에 Sun Microsystems에서 설계한 Clustered Page Table 기법을 FTL 상에서

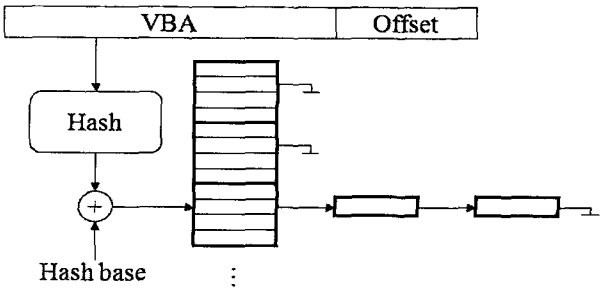

그림 5. 일반 Hash Table
Fig. 5. Traditional Hash Table.

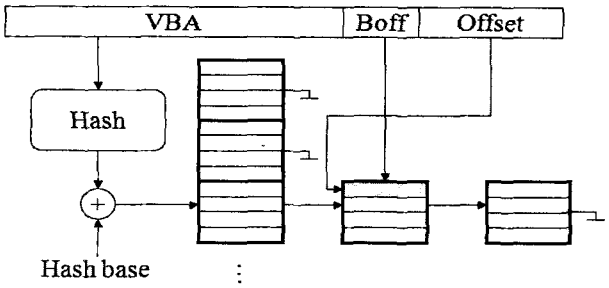

그림 6. Clustered Hash Table
Fig. 6. Clustered Hash Table.

일컫는 단어이다. Clustered Page Table은 64bit 가상 메모리 내의 페이징 기법 중의 하나이며 Hash table에서 Subblocking 기법을 이용한 확장 방식이다.

Clustered Hash Table에서 버켓은 Subblock의 집합을 의미하며 각 Subblock의 개수를 Subblock factor라고 일컫는다. Clustered hash table의 장점으로는 Hash Table은 주소 변환 속도가 항상 고정되어 있어 유연하지 못하며 Tag 및 Next 포인터가 노드별로 추가적으로 필요하므로 메모리 사용량에서 200%의 오버헤드가 발생한다. 하지만 Clustered Hash Table은 그림 6과 같이 버켓 단위로 구성되어 있어 메모리의 사용량이 그림 5의 Hash Table보다 적다. 또한 연속적이면서 동시에 사용하는 데이터들을 Clustered Hash Table의 버켓의 Subblock들에 적재하면 연속적인 주소 접근 시 멀티 레벨의 Fine-grained clustered hash table을 통해 적중률을 증가 시킬 수 있다.

3. 2 단계 Fine-grained clustered hash table

CFTL의 성능을 높이기 위해 하드웨어 캐시 종류 중의 하나인 TLB(Translation Lookaside Buffer)의 개념을 차용하여 그림 7과 같이 2단계 Fine-grained clustered hash table을 구현하였다. 기존의 Software TLB 기법을 차용하기 때문에 접근 속도를 향상 시키지는 못하지만 2단계 fine-grained clustered hash table을

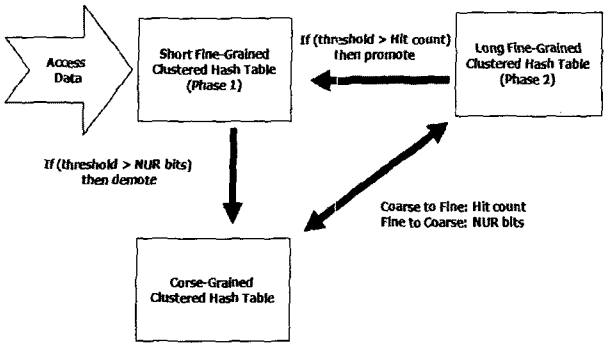

그림 7. Coarse-to-Fine, Long-to-Short 승급 정책, 그리고 Fine-to-Coarse 강등 정책

Fig. 7. Coarse-to-Fine, Long-to-Short promotion policy, and Fine-to-Coarse demotion policy.

이용해 Miss Penalty를 감소시키며 Coarse-Grained Clustered Hash Table의 접근 빈도를 낮출 수 있다. 그러므로 주소 변환 대부분이 Fine-grained clustered hash table에서 이뤄지므로 주소 변환 시간을 줄일 수 있다. 또한 2단계의 주소 변환 테이블을 두므로 이에 따른 Coarse-to-Fine 또는 Fine-to-Coarse간의 이동 정책이 필요하다. coarse-to-fine으로 이동하는 경우를 예를 들어 특정 LBA에 잦은 접근으로 인해 Hit count가 일정횟수만큼 증가하면 처음에는 Long fine-grained clustered hash table로 승급한다. 이와 함께 Hit count는 초기화 된다. Short fine-grained clustered hash table에서도 이러한 정책은 동일하게 적용된다. 그리고 NUR 기법을 통해 일정 시간 동안 접근이 되지 않을 시 Fine-grained clustered hash table에서 Coarse-grained clustered hash table로 강등되는 정책을 적용한다. 이를 Fine-to-Coarse 강등 정책이라고 한다.

4. NUR(Not Used Recently)

NUR은 유사 LRU의 하나로 적은 오버헤드로 LRU 기법의 성능을 낼 수 있는 기법으로 알려져 있다. NUR은 LRU와 달리 참조, 수정 비트라는 비트 벡터를 사용하며 시스템에서 정해진 주기마다 비트에 따라 최근에 안 쓰인 슬롯들을 분별 할 수 있는 방법이다. 두 개의 비트를 통해 참조된 상황과 수정이 되었는지를 판단하여 표 1과 같이 Fine-to-Coarse 교체 기법에 적용 할 수 있다. 특정 주소의 NUR의 참조, 수정 비트가 모두 0인 경우는 그 주소는 사용되지 않는 것으로 보고 fine-grained clustered hash table에서 Coarse-grained Clustered Hash Table로 이동 시키고 Fine-grained clustered hash table의 슬롯을 삭제 한다.

표 1. NUR의 참조, 수정 비트에 따른 교환 정책

Table 1. Replacement policy for reference, modification bit of NUR.

참조 비트	수정 비트	정책
0	0	Coarse로 이동
0	1	유지
1	0	유지
1	1	유지

Ⅳ. 성 능 평 가

성능 평가를 위해 CFTL에서 Short fine-grained clustered hash table, Long fine-grained clustered hash table의 크기 비율을 1:4로 정하고 Clustered Hash Table의 Subblock factor를 8로 설정했다. 표 2는 저장장치 성능 벤치마킹 방식으로 잘 알려진 앤드류 벤치마크를 통해 동일한 환경에서 각 FTL이 소모하는 System Time을 10회 측정한 값을 평균을 냈다[3]. CFTL의 전체적인 성능이 앤드류 벤치마크 테스트에서 NFTL과 AFTL보다 빠른 것을 알 수 있다. 표 3은 MFS(Maximum Fine-Grained Slot)에 따른 CFTL과 AFTL의 메모리 사용량을 비교한 것이다. Clustered Hash Table을 적용한 CFTL이 메모리 사용량을 약 75% 이상 감소 시켰음을 알 수 있다.

위의 실험 결과를 CFTL을 기준으로 결론을 내면 표 4와 같이 CFTL은 AFTL보다 메모리 사용량이 75% 이상 적으면서 NFTL보다는 많은 중간 크기이며

표 2. 앤드류 벤치마크 실험 결과

Table 2. Andrew benchmark test result.

	NFTL	AFTL	CFTL
Real Time	1m 9.47s	1m 14.30s	1m 10.27s
User Time	34.66s	34.95s	35.34s
System Time	14.41s	13.70s	12.68s
System Time (excepted compilation)	6.02s	5.85s	4.83s

표 3. 메모리 사용량

Table 3. Memory Utilization.

MFS(L1 + L2)	NFTL	AFTL	CFTL
2,500 + 10,000	64.0KB	244.1KB	61.0KB
5,000 + 20,000	64.0KB	488.3KB	122.1KB
7,500 + 30,000	64.0KB	732.4KB	183.1KB
10,000 + 40,000	64.0KB	976.6KB	244.1KB
12,500 + 60,000	64.0KB	1220.7KB	305.2KB
15,000 + 70,000	64.0KB	1464.8KB	366.2KB

표 4. 각 FTL 간의 성능 평가 결과
Table 4. Each FTL relative performance results.

	NFTL	AFTL	CFTL
메모리 사용량	작다	크다	중간
주소변환 시간	길다	중간	짧다

주소 변환 시간은 NFTL, AFTL보다 각각 13%, 8% 이상 빠르다.

V. 결 론

NAND 플래시 메모리의 용량이 매년 증가하고 자기 하드 디스크를 대체하게 됨에 따라 NAND 플래시 메모리를 기존의 파일 시스템과 호환이 가능하면서 NAND 플래시 메모리의 빠른 성능을 이끌어 낼 수 있는 FTL의 중요성이 점점 높아지고 있다. 이러한 FTL은 현재 모든 휴대용 저장장치 표준에 적합하게 설계 되어 내장되고 있으므로 반드시 필요한 기술이라고 볼 수 있다. 본 논문에서 제시한 CFTL은 대용량 저장장치에 사용 시 AFTL보다 성능이 뛰어나고 주소 변환 테이블의 크기가 작기 때문에 휴대용 저장장치에 사용하기에 적합하다. 향후 Subblocking 기법과 Superpage를 통한 향상된 소프트웨어 TLB 기법을 적용한다면 작은 크기의 메모리를 사용하면서 빠른 성능을 낼 수 있는 FTL을 설계 할 수 있을 것이다.

참 고 문 헌

[1] C.-H. Wu and T.-W. Kuo, "An Adaptive Two-Level Management for the Flash Translation Layer in Embedded Systems," International Conference on Computer Aided Design, San Jose, CA, Nov 2006.
[2] M. Talluri, M. D. Hill and Y. A. Khalidi, "A new page table for 64-bit address spaces," In Proc. of Symposium of Operating System Principles, Dec 1995.
[3] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and Performance in a Distributed File System," ACM Transactions on Computer Systems, Vol. 6, pp. 51-81, Feb 1988.
[4] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compact-Flash Systems," IEEE Transaction on Consumer Electronics, Vol. 48, No. 2, May 2002.
[5] Intel Corporation, "Understanding the Flash Translation Layer Specification".
[6] Intel Corporation, "Software Concerns of Implementing a Resident Flash Disk".

저 자 소 개

박 광 희(학생회원)

2005년 용인대학교 컴퓨터정보처리학과 학사 졸업.
2008년 인하대학교 전자공학과 석사 졸업
2008년 인하대학교 박사 1년차
2004년 인프니스 기술연구소 연구원
2004년~2005년 인테그라 정보통신 부설연구소 연구원

<주관심분야 : 임베디드 시스템, 스토리지 시스템, 방송통신서비스>

김 덕 환(정회원)

1987년 서울대학교 계산통계학과 학사 졸업
1995년 한국과학기술원 정보통신공학과 석사 졸업
2003년 한국과학기술원 정보통신공학과 박사 졸업
1987년~1997년 LG전자 통신기기연구소 선임연구원
1997년~2006년 동양공업전문대학 전산경영기술공학부 부교수
2004년 University of Arizona, Tucson 박사후 연구원
2006년~현재 인하대학교 전자전기공학부 부교수

<주관심분야 : 임베디드 시스템, 시스템소프트웨어, 멀티미디어, 패턴인식, 데이터마이닝>