Design and Evaluation of the Compressed Flash Translation Layer for High-Speed and Large-Scale Flash Memory Storages

임근수†, 반효경‡, 김지홍†, 고 건†
†서울대학교 컴퓨터공학부, ‡이화여자대학교 컴퓨터학과
†서울특별시 관악구 신림9동 산 56-1번지, ‡서울특별시 서대문구 대현동 11-1번지
전화: +82-2-880-1844, 팩스: +82-2-875-7021
Email: ksyim@oslab.snu.ac.kr, bahn@ewha.ac.kr, jihong@davinci.snu.ac.kr, kernkoh@oslab.snu.ac.kr

Abstract — *Recently flash memory is widely deployed as a storage medium for mobile and embedded systems. Since flash memory has mainly two drawbacks of slow write speed and a high cost per byte ratio, several techniques were developed to address these problems. Specifically, the flash compression layer (FCL) expands the storage capacity and the write bandwidth by storing and transferring data in compression form, and the flash translation layer (FTL) logically removes erase operations of flash memory from a host system by redirecting logical addresses to physical addresses. Both the FCL and the FTL were individually developed, thus they use redundant hardware components and are not efficient in some cases. Therefore, this paper presents an efficient middleware that unifies the FCL and the FTL for high-speed and large-scale flash memory based storage systems. The simulation results show that the proposed compressed flash translation layer (CFTL) expands the storage capacity by over 40% and also reduces the write latencies by 18-19% that are mainly due to its integrated architecture.*[1]

I. 서론

최근 컴퓨팅 패러다임이 언제 어디서나 원하는 정보와 컴퓨팅 파워를 활용할 수 있는 유비쿼터스 컴퓨팅으로 전환되어 감에 따라 휴대폰, PDA, 디지털카메라와 같은 휴대용 정보기기의 사용이 급증하고 있다. 이와 같은 휴대기기에서는 일반적으로 하드디스크 대신 플래시메모리를 보조기억장치로 사용한다. 왜냐하면 플래시메모리는 가볍고 물리적인 충격에 강해 휴대가 용이할 뿐만 아니라 저전력으로 동작해 배터리 소모량을 줄이기 때문이다[1].

표 1인 플래시메모리의 특성을 다른 메모리 소자와 비교한 것으로, 플래시메모리는 비교적 낮은 단가로 빠른 읽기속도를 제공하는 영속성 있는 저장장치임을 알 수 있다. 플래시메모리는 크게 바이트단위 I/O를 지원하는 NOR형과 페이지단위 I/O만을 지원하는 NAND형이 있다. NOR형은 읽기속도가 빠른데 반하여 쓰기속도가 느려 주로 코드용 메모리로 사용하며[7], NAND형은 이와는 대조적으로 쓰기속도가 빠르고 단위 공간당 단가가 낮아 주로 대용량 데이터 저장장치로 사용한다[6].

[1] 본 연구는 정보통신연구진흥원에서 지원하는 기초기술연구사업과 교육인적자원부에서 지원하는 BK21사업의 일환으로 수행되었음.

표 1. 다양한 메모리 소자의 특성 비교.

	Read	Write	Erase	Cost/MB
DRAM	60ns (2B) 2.6μs (512B)	60ns (2B) 2.6μs (512B)	-	30~40
NOR-type Flash	150ns (1B) 15μs (512B)	211μs (1B) 3.5ms (512B)	1.2s (128KB)	20~30
NAND-type Flash	10μs (1B) 36μs (512B)	226μs (1B) 266μs (512B)	2ms (16KB)	10~20
Magnetic Disk	12.4ms (512B)	12.4ms (512B)	-	1

EEPROM인 플래시메모리에 데이터를 쓰기 위해서는 삭제연산을 선행해야 하는데, 삭제 단위는 쓰기 단위보다 크며 수행시간이 오래 걸린다. 이러한 특성은 플래시메모리를 주 메모리로 사용하는 것을 어렵게 한다. 그리고 보조기억장치로 사용하는 경우에도 일반 하드디스크용 파일시스템을 그대로 활용하는 것을 저해한다.

그래서 논리적으로 삭제연산을 감출 수 있는 플래시 변환계층이 파일시스템과 플래시메모리 사이의 미들웨어 형태로 제안되어 사용되고 있다[2, 3, 8]. 플래시 변환계층은 쓰기연산 시에 파일시스템이 생성한 논리주소를 플래시메모리상의 이미 삭제연산을 수행한 영역에 대한 물리주소로 변환하는 역할을 수행한다. 빠른 주소변환을 위해 주소변환 테이블은 SRAM을 사용해 구성한다. 플래시 변환계층을 사용하면 호스트 시스템에서는 FAT와 같은 일반 자기디스크용 파일시스템을 사용해서도 플래시메모리를 효율적으로 제어할 수 있다.

하지만 플래시 변환계층을 사용하여 삭제연산을 감춰도 플래시메모리의 쓰기속도는 다른 메모리 소자와 비교해 상대적으로 느려 여전히 쓰기 성능을 개선해야 할 필요가 있다. 또한 하드디스크와 비교해 단위 공간당 단가가 수십 배 가량 높기 때문에 한정된 기억공간을 효율적으로 활용해야 할 필요성이 있다.

이러한 두 가지 문제점을 효율적으로 개선할 수 있는 방법에는 데이터 압축기법이 있다[4, 5]. 압축기법을 활용해 압축된 데이터를 전송하고 기록함으로써 플래시메모리의

쓰기 대역폭과 기억공간을 실질적으로 확장할 수 있다. 플래시 압축계층은 이러한 기능을 호스트 독립적으로 수행하는 미들웨어로 하드웨어 압축기와 쓰기버퍼로 구성한다[13].

플래시 변환계층과 플래시 압축계층은 서로 독립적으로 설계되었기 때문에 마이크로프로세서와 주 메모리 등과 같은 하드웨어를 중복해서 사용하는 문제가 있다. 뿐만 아니라 플래시 압축계층에서 가진 블록내부의 유효한 페이지의 개수와 같은 정보를 플래시 변환계층에서 삭제블록을 선택하는 과정에 사용한다면 보다 삭제정책의 효율성을 높일 수 있다.

따라서 본 논문에서는 플래시 변환계층과 플래시 압축계층을 통합해 압축 플래시 변환계층을 설계해 중복된 하드웨어 비용을 제거하고 시스템의 성능을 최적화한다. 제안하는 압축 플래시 변환계층의 성능은 공인된 벤치마크를 활용해 시뮬레이션을 통해 평가한다. 시뮬레이션 결과 플래시메모리의 기억공간을 40%이상 확장하고 쓰기연산 속도를 18-19%가량 개선함을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 플래시 변환계층의 원리와 다양한 세부종류에 대해 알아보고, 3장에서는 플래시 압축계층의 고안 배경과 동작원리에 대해 살펴본다. 4장에서는 이 두 가지 시스템을 통합해 설계한 압축 플래시 변환계층에 대해 기술하고, 5장에서는 시뮬레이션을 통해 성능을 평가한다. 그리고 6장에서 결론을 맺는다.

II. 플래시 변환계층

플래시 변환계층(Flash Translation Layer, FTL)은 플래시메모리의 삭제연산을 감추기 위한 미들웨어로 호스트 시스템의 파일시스템과 플래시메모리 사이에 위치한다. 삭제연산은 쓰기연산 시에 파일시스템이 생성한 논리주소를 플래시메모리상의 이미 삭제연산을 수행한 영역에 대한 물리주소로 변환함으로써 감춰진다. 비교적 수행시간이 오래 걸리는 삭제연산을 감추고 I/O를 하나의 단위(atomic operation)로 처리해 하드디스크와 같은 단일 저장장치를 구성함으로써, 상단에서 일반 파일시스템을 사용해서 플래시메모리를 효율적으로 제어할 수 있다. 이는 현대의 운영체제의 기본 철학처럼 정책(policy)과 기능(mechanism)을 각각 파일시스템과 저장장치로 구분한 것으로 볼 수 있다. 기능에 해당하는 FTL은 크게 호스트시스템에서 독립된 하드웨어 형태[2, 3, 8]와 호스트시스템 내부의 디바이스드라이버 형태[14]로 구현할 수 있다.

FTL은 주소변환 단위에 따라 크게 페이지(쓰기)단위 주소변환과 블록(삭제)단위 주소변환으로 나뉜다. 페이지단위 주소변환은 그림 1(a)와 같이 정교하게 주소를 변환하기 때문에 성능은 좋은데 반하여 주소변환 테이블의 크기가 커져 제작비용이 높아지는 단점이 있다. 반대로 블록단위 주소변환은 그림 1(b)처럼 비교적 비정교하게 주소를 변환해 주소변환 테이블의 크기는 작지만, 내부의 한 페이지에 대한 수정연산이 발생해도 전체 블록을 삭제하고 갱신해야 하는 추가비용이 있다. 뿐만 아니라 이 과정을 수행하는 도중에 폴트가 발생하면 데이터의 일관성을 깨뜨릴 수 있다.

이러한 블록단위 주소변환 방식의 단점을 개선한 기법은 교체블록 기법은 그림 1(c)와 같이 블록 내부의 페이지에 대한 갱신요청이 발생하면 교체블록을 할당해 쓰기를 수행하고 이를 연결리스트로 구성해 향후 읽기연산 시에 이 리스트를 역순으로 검색해 가장 늦게 수정된 데이터를 제공하는 방법이다.

마지막으로 최근에 제안된 로그블록 기법은 그림 1(d)처럼 페이지단위 주소변환과 블록단위 주소변환 기법을 병합한 형태로 비교적 큰 단위로 요청되는 순차 입출력은 블록단위로 처리하고, 작은 단위로 요청되는 임의 입출력은 페이지 단위로 로그구조 파일시스템[15, 16]과 유사하게 로그형태로 저장하는 방식이다[3]. 이를 통해 주소변환 테이블의 크기를 줄이고도 상대적으로 높은 성능을 발휘할 수 있다.

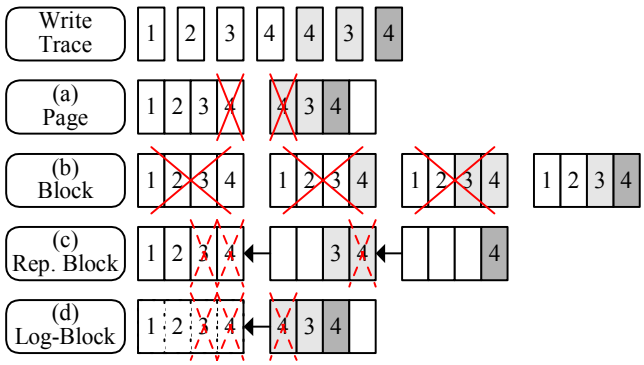

그림 1. 플래시 변환계층의 설계기법에 따른 동작예제.

이처럼 FTL은 주소변환 테이블을 사용해 삭제연산을 감출 수 있어 플래시메모리의 쓰기 성능을 개선하고 일반 파일시스템을 사용해 제어 가능하게 할 뿐만 아니라 실시간 데이터베이스 시스템 등에서 비휘발성 주 메모리로도 활용할 수 있게 한다.

하지만 FTL을 사용해 삭제연산을 논리적으로 감춰도 플래시메모리의 쓰기속도는 다른 메모리 소자와 비교해 상대적으로 느리기 때문에 여전히 쓰기 성능을 개선해야 할 필요가 있다. 뿐만 아니라 하드 디스크와 비교해 수십 배 가량 단위 공간당 단가가 높기 때문에 기억공간을 효율적으로 활용해야 할 필요가 있다. 이러한 문제점을 극복하기 위해서 플래시 압축계층이 제안되었다.

III. 플래시 압축계층

플래시 압축계층(Flash Compression Layer, FCL)은 데이터를 압축해 전송하고 저장함으로써 플래시메모리의 쓰기 대역폭과 기억공간을 실질적으로 확장한다. 데이터 압축의 일반적인 특성은 동일한 크기의 페이지를 압축해도 압축률에 따라 압축 페이지의 크기가 다양해 진다는 것이다. NOR형 플래시메모리는 바이트 I/O를 지원하기 때문에 다양한 크기의 압축페이지를 관리하는데 효율적이다.

하지만 페이지 I/O만의 지원하는 NAND형 플래시메모리는 압축페이지 관리가 용이하지 않다. 이러한 특성으로 인해서 FCL은 플래시메모리의 종류에 따라 서로 다른 형태로 설계되었다.

과거 NAND형 플래시메모리가 개발되기 이전에 NOR형 플래시메모리를 데이터 저장장치로 널리 사용하던 시대에는 NOR형 플래시메모리를 위한 압축기법이 몇 가지 제안된 바 있다[4, 5]. 최근에 NAND형 플래시메모리가 출시 이후에 NOR형 플래시메모리는 주로 코드용 메모리로 사용하고 있는데, 코드 메모리에 압축기법을 적용하는 것은 적합하지 않기 때문에 NOR형 플래시메모리 압축에 대한 연구는 주목을 받지 못하고 있다. 왜냐하면 코드 메모리는 쓰기연산의 비율이 적고 코드를 압축하게 되면 코드의 수행속도를 느리게 할 수 있기 때문이다.

그래서 현대의 플래시메모리 압축에 대한 연구는 주로 NAND형 플래시메모리를 대상으로 하고 있다. 본 논문에서도 대용량 저장장치로 널리 사용되는 NAND형 플래시메모리를 사용한 플래시 압축계층에 대해 주로 살펴본다[13].

NAND형 플래시메모리에 압축페이지를 저장하면 그림 2와 같이 내부파편이 발생해 압축효율을 크게 저하시킬 수 있다. 이때 내부 파편이 발생한 영역에 이보다 작은 크기의 데이터를 저장하는 것은 실질적으로 거의 불가능하다. 왜냐하면 이를 위해서는 인접한 페이지로 구성된 블록에 대해 비교적 긴 수행시간을 갖는 삭제연산을 선행해야 하기 때문이다. 따라서 NAND형 플래시메모리에서 페이지 크기와 입출력 단위가 같으면 일반적으로 압축기법을 사용하여도 공간확장과 같은 이득을 얻을 수 없다.

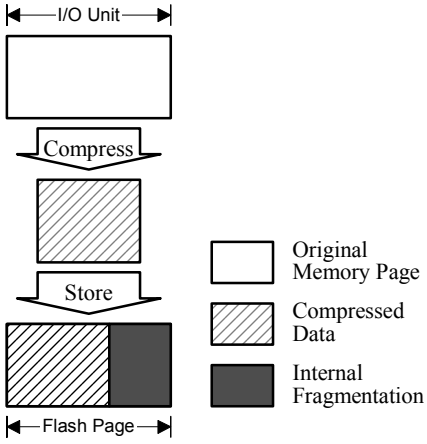

그림 2. 내부파편 문제.

일반적으로 내부파편은 큰 입출력 단위를 사용하거나 작은 플래시 페이지 크기를 사용함으로써 줄일 수 있다. 하지만 간단하게 입출력 단위를 키우거나 플래시 페이지 크기를 키우는 형태로 설계를 변경하면 이 보다 심각한 문제를 야기할 수 있다. 큰 입출력 단위는 임의접근 시에 입출력 연산의 속도를 지연시키고 인접한 페이지에 대한 중복된 입출력 연산의 수행을 야기한다. 그리고 작은 플래시 페이지 크기는 플래시 메모리의 CMOS 회로설계를 복잡하게 해 공정의 효용을 낮추고 FTL에서 사용하는 주소변환 테이블의 크기를 키우게 된다.

그래서 위의 설계요소를 변경하지 않으면서 내부파편을 효율적으로 줄일 수 있는 기법으로 페이지 그룹화 기법이 제안되었다[13]. 페이지 그룹화 기법을 사용한 FCL의 동작원리는 그림 3과 같다. 각각의 메모리 페이지는 개별적으로 압축되어 쓰기버퍼에 임시적으로 저장된다. 이때 쓰기버퍼는 바이트단위 I/O를 지원해야 하며 예외상황에 대응하기 위해서는 비휘발성이어야 한다. NOR형 플래시메모리와 배터리 부착형 RAM이 이 두 조건을 만족한다. 쓰기버퍼에 압축 페이지를 저장할 때는 내부파편을 최소화할 수 있도록 그룹화하는데 크게 3가지 서로 다른 그룹화 기법이 있다[12]. 이후에 쓰기버퍼의 사용비율이 임계값을 넘으면 쓰기버퍼의 그룹 중 가장 내부파편을 적게 가진 그룹을 선택해 저장장치인 NAND형 플래시메모리에 실제로 저장한다.

그리고 FCL에서는 압축과 복원에 따른 지연시간을 감출 수 있도록 입출력 버스보다 압축 및 복원 속도가 빠른 X-RL과 같은 고속 하드웨어 압축기를 사용한다[9, 10]. FCL에서 X-RL을 사용한 또 다른 이유는 X-RL이 플래시 페이지 크기와 같은 작은 크기의 페이지에 대해서도 압축효율이 좋고 하드웨어 구조가 비교적 간단하기 때문이다.

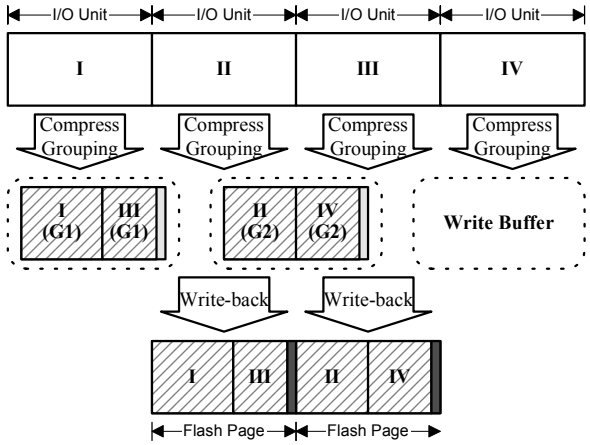

그림 3. 플래시 압축계층의 동작 예제.

이상에서 살펴본 현재까지 고안된 FTL과 FCL은 개별적으로 설계되었기 때문에 마이크로프로세서와 주 메모리와 같은 하드웨어를 중복해 사용하는 문제를 갖는다. 또한 FCL이 관리하는 블록내부의 유효한 페이지의 개수와 같은 정보를 FTL이 삭제블록을 선택 단계에서 활용할 수 있다면 보다 삭제 정책의 효율성을 높일 수 있을 것이다. 따라서 본 논문에서는 FTL와 FCL을 통합한 형태의 압축 플래시 변환계층을 설계한다.

IV. 압축 플래시 변환계층

제안하는 압축 플래시 변환계층(Compressed Flash Translation Layer, CFTL)의 블록 다이어그램은 그림 4와 같다. CFTL의 구조는 크게 FTL에 해당하는 주소 변환테이블이 위치한 좌측부분과 FCL에 해당하는 압축기와 복원기 그리고 쓰기버퍼로 구성된 우측부분으로 구분할 수 있다. 이때 쓰기버퍼의 크기는 선행 연구의 결과를 바탕으로 160KB로 설정한다[14]. 다이어그램에 제시된 하드웨어 이외에도 마이크로프로세서와 주 메모리용 DRAM이 기본적으로 사용된다. 다이어그램상에 CFTL로 표시된 부분은 SOC(System on Chip)형태로 구현하면 보다 제안하는 시스템의 실용성을 높일 수 있을 것이다.

이와 같은 CFTL의 구조는 몇 가지 경우의 입출력 연산의 속도를 개선할 수 있다. 쓰기연산의 경우 임계 값보다 압축률이 좋은 경우에 쓰기 버퍼에 기록해 그룹화하게 되는데 이 경우 데이터를 압축해 크기가 줄어들었고 쓰기버퍼의 속도가 저장장치보다 빠르기 때문에 쓰기 성능을 개선할 수 있다. 또한 입력의 경우에도 요청 페이지가 쓰기버퍼에서 적중하면 읽기속도가 빠른 쓰기버퍼에 대한 선인출 효과를 얻게 된다. 그리고 순차적으로 쓰기와 읽기가 요청되는 경우 각각의 연산이 쓰기버퍼와 저장장치에 분산돼 병렬적으로 처리되기 때문에 입출력 속도를 개선할 수 있다.

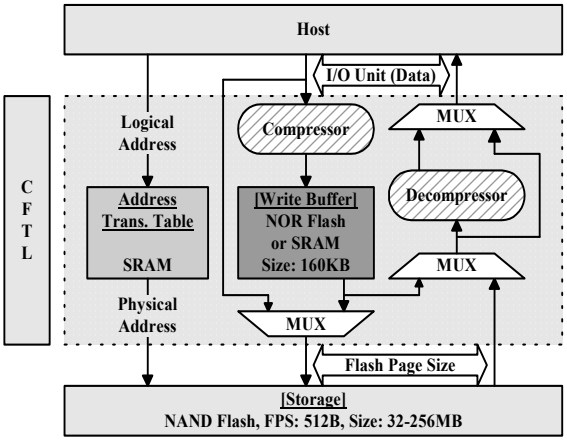

그림 4. 압축 플래시 변환계층의 블록 다이어그램.

다음으로 FTL과 상호작동을 위해서 쓰기버퍼에서 압축 페이지 그룹을 저장장치의 플래시 페이지에 실제로 기록하는 경우에, FTL의 주소변환 과정에서 사용되는 메타데이터를 함께 저장한다. 메타데이터는 NAND형 플래시메모리의 모든 플래시 페이지에 존재하는 16B 또는 64B 크기의 여분의 공간(spare area)을 활용해 저장한다. 메타데이터는 각각의 압축 페이지의 논리주소와 크기 그리고 전체 페이지에 대한 에러 검사코드이다. 여분의 공간의 크기가 제한되어 있어서 하나의 플래시 페이지에 저장할 수 있는 최대 압축 페이지의 개수(N) 는 수식 1과 같은 제한된다.

$$N = \left\lfloor \frac{S_{spare} - S_{ECC}}{\lg \frac{S_{storage}}{S_{page}} + \lg S_{page}} \right\rfloor = \left\lfloor \frac{S_{spare} - S_{ECC}}{\lg S_{storage}} \right\rfloor \quad (1)$$

여기서 S_{spare}는 여분 공간의 크기를 S_{ECC}는 에러 검사코드의 길이를 S_{page}는 플래시 페이지 크기를 그리고 S_{storage}는 플래시 메모리의 총 용량을 의미한다. 일반적으로 여분의 공간의 크기가 16B인 경우에는 4이고 64B인 경우에는 19로 충분히 많은 압축 페이지를 한 플래시 페이지에 저장할 수 있다.

그 외의 CFTL의 동작원리는 로그블록 기법의 FTL과 유사하다[3]. 즉 시스템이 시작하는 경우에 메타데이터를 읽어 주소변환 테이블을 구성하고 이를 바탕으로 읽기와 쓰기 연산을 지원한다. 그리고 플래시메모리의 사용률이 일정비율 이상이 되면 가장 유효한 페이지의 숫자가 적은 블록을 선택해 해당 블록의 총 삭제횟수를 고려해 삭제연산을 수행한다.

V. 성능 평가

이 장에서는 CFTL의 성능을 시뮬레이션을 통해 평가한 과정과 결과를 제시한다. 다양한 설계 환경에 따른 성능분석을 원활하게 수행하기 위해서 시뮬레이션 기법을 사용하였다. 이때 벤치마크는 무손실 압축 알고리즘 평가에 널리 사용하는 Canterbury Corpus 벤치마크를 사용하였다[11]. 공인된 벤치마크를 사용했기 때문에 본 논문의 실험결과를 관련 연구들의 결과와 비교적 용이하게 비교할 수 있다는 장점이 있다.

성능향상 비율 계산을 수월하게 하기 위하여 몇 가지 성능지표를 새롭게 정의한다. 먼저 압축률(CR)을 압축 페이지의 크기를 소스 페이지의 크기로 나눈 값으로 정의한다. 즉 압축률이 낮을수록 높은 압축효율을 의미한다. 내부파편 비율(IFR)은 내부 파편의 크기를 소스 페이지의 크기로 나눈 값으로 정의한다. 그리고 실질 압축률(ECR)을 NAND형 플래시메모리에 저장한 압축 페이지들의 평균 압축률과 평균 내부파편 비율의 합으로 정의한다. 그러면 실질 압축률은 총 사용된 플래시 페이지의 개수 분에 총 저장한 메모리 페이지의 개수가 된다.

앞서 언급한 것과 같이 CFTL의 설계 목표는 쓰기 대역폭과 기억 공간을 효율적으로 확장하고 쓰기연산 시간을 단축하는 것이다. 여기서 쓰기 대역폭의 확장 비율(E_{WB})은 $1/CR$-1과 같이 평균 압축률을 사용해 계산할 수 있으며, 기억공간의 확장비율(E_{SC})은 $1/(CR+IFR)$-1=$1/ECR$-1과 같이 실질 압축률로부터 유도할 수 있다.

그림 5는 벤치마크 프로그램의 압축단위에 따른 압축률을 보인다. CFTL은 입출력 연산을 지연시키지 않기 위해서 압축단위를 저장장치의 플래시 페이지 크기와 동일하게 설계하였다. 즉 플래시 페이지 크기가 제품에 따라 512B와 2KB인 경우, X-RL 알고리즘을 사용한 CFTL의 해

당 벤치마크에 대한 평균 압축률은 각각 약 59.3%와 57.3%이다.

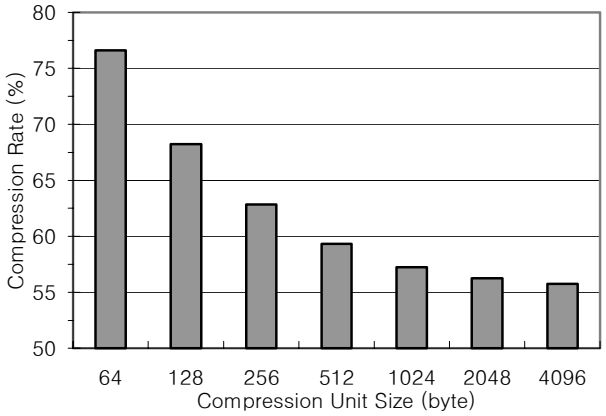

그림 5. 압축단위에 따른 압축률.

그림 6은 벤치마크 프로그램 별 내부파편 비율이다. 평균 내부파편 비율은 플래시 페이지 크기가 512B인 타입 I의 경우 약 11.7%이고 페이지 크기가 2KB인 타입 II의 경우에는 약 24.3%이다. 작은 플래시 페이지 크기를 사용하면 내부파편을 더 많이 줄임을 알 수 있다. 그리고 플래시 페이지 크기와 무관하게 벤치마크 fax의 경우 내부파편을 거의 제거하는데 이는 해당 벤치마크의 평균 압축률이 약 16%로 매우 뛰어나기 때문이다. 즉 제안하는 CFTL에서 내부파편 비율은 압축률의 평균과 분포에 의해 결정된다.

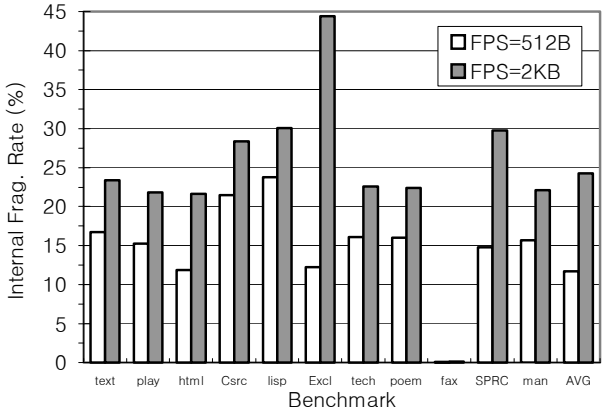

그림 6. 플래시 페이지 크기(FPS)에 따른 벤치마크 별 내부파편 비율.

표 2는 저장장치로 NAND형 플래시메모리 타입 I을 사용한 경우에 제안하는 CFTL의 성능을 요약한 것이다. 저장공간의 확장비율은 40%이상이며, 쓰기 대역폭 확장비율은 약 49%임을 알 수 있다. 그리고 이러한 성능향상은 쓰기버퍼의 종류와는 무관한 것임을 알 수 있다. 여기에서 쓰기 대역폭 확장비율은 쓰기 연산시의 입출력 버스 사용량의 감소비율을 의미하는 것으로, 이로 인해 소모전력을 줄일 수는 있지만, 이 결과가 쓰기연산 속도 자체의 단축을 의미하는 것은 아니다.

표 2. 제안하는 압축 플래시 변환계층의 성능 요약.

성능지표 / 쓰기버퍼	저장공간 확장비율	쓰기 대역폭 확장비율	쓰기연산 시간 단축비율
NOR Flash	40.8%	49.1%	18.2%
DRAM	40.8%	49.1%	19.0%

실제 쓰기연산 속도를 구하기 위해서 CFTL에 기반한 NAND형 플래시메모리 저장장치에 벤치마크를 순차적으로 저장하는 실험을 수행하였다. 쓰기버퍼로는 읽기연산의 속도는 빠른데 반하여 비교적 쓰기연산의 속도가 느린 NOR형 플래시메모리와 읽기 및 쓰기연산의 속도가 모두 빠른 배터리 부착형 DRAM을 사용하였다. 실험결과 표 2와 같이 평균적으로 18-19%가량 쓰기연산 시간을 단축하였다. 이는 대부분의 쓰기연산이 저장장치인 NAND형 플래시메모리에 페이지 단위로 쓰여지지 않고, 압축된 형태로 쓰기버퍼에 쓰여져 그룹화되어 여러 압축 페이지가 한번에 쓰여졌기 때문이다. 쓰기버퍼에서 저장장치로 실제로 데이터가 쓰여질 때 쓰기버퍼에서 한 페이지를 읽는 동안의 시간이 일반 시스템과 비교한 추가비용이 되지만, 쓰기버퍼로 사용한 NOR형 플래시메모리와 DRAM 모두 읽기연산의 속도가 빨라 이로 인한 성능손실을 전체 성능에 거의 영향을 끼치지 않았다.

그리고 쓰기버퍼로 비교적 단가가 높고 실행시간에 전력을 많이 소모하는 배터리 부착형 DRAM을 사용한 경우와 이와는 대비되는 특성을 지닌 NOR형 플래시메모리를 사용한 경우에 성능상에 큰 차이가 없었다. 따라서 CFTL의 쓰기버퍼는 NOR형 플래시메모리를 사용하는 것이 동일한 성능을 발휘하면서도 전원소모를 줄이고 생산단가를 낮출 수 있다.

VI. 결론

본 논문에서는 초고속 대용량 플래시메모리 기반 저장장치 설계를 위해 호스트시스템과 플래시메모리 사이의 미들웨어 형태로 압축 플래시 변환계층을 설계하고 성능을 분석하였다. 압축 플래시 변환계층은 하드웨어 비용을 낮추고 성능향상을 최적화하기 위해 기존에 개발된 플래시 변환계층과 플래시 압축계층을 통합해 설계하였다. 시뮬레이션 결과 제안하는 시스템은 플래시메모리의 저장공간을 실질적으로 40%이상 확장하고 쓰기연산 시간을 평균적으로 18-19%가량 단축시킴을 확인하였다. 추가적으로 쓰기 대역폭을 약 49%가량 확장함을 보였는데 이는 제한하는 시스템이 전력 소모로 줄일 수 있음을 의미한다. 본 논문에서는 데이터 압축기법을 활용하여 데이터를 압축해 전송하고 저장함으로써 플래시메모리의 대표적인

문제점인 느린 쓰기연산 문제와 단위 공간당 높은 단가 문제를 크게 개선할 수 있음을 보였다.

참고 문헌

[1] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber, "Storage Alternatives for Mobile Computers," *In Proceedings of the 1st Symposium on Operating System Design and Implementation*, pp. 25-37, 1994.

[2] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," *In Proceedings of the 6th Symposium on Architectural Support for Programming Languages and Operating Systems*, pp. 86-97, 1994.

[3] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for CompactFlash Systems," *IEEE Transactions on Consumer Electronics*, Vol. 48, No. 2, pp.366-375, 2002.

[4] M. Kjelso and S. Jones, "Memory Management in Flash-Memory Disks with Data Compression," *Lecture Notes in Computer Science*, Vol. 986, Springer Verlag, pp. 399-413, 1995.

[5] S. Wells and D. Clay, "Flash Solid-State Drive with 6MB/s Read/Write Channel and Data Compression," *In Proceedings of the 40th International Conference on Solid-State Circuits*, pp. 52-53, 1993.

[6] Samsung Electronics, "256M x 8 Bit / 128M x 16 Bit NAND Flash Memory," http://www.samsungelectronics.com/.

[7] Intel Corporation, "3 Volt Synchronous Intel StrataFlash Memory," http://www.intel.com/.

[8] Intel Corporation, "Understanding the Flash Translation Layer (FTL) Specification," http://developer.intel.com

[9] S. Bunton and G. Borriello, "Practical Dictionary Management for Hardware Data Compression," *Communications of the ACM*, Vol. 35, No. 1, pp. 95-104, 1992.

[10] M. Kjelso, M. Gooch, and S. Jones, "Design and Performance of a Main Memory Hardware Data Compressor," *In Proceedings the 22nd Euromicro Conference*, IEEE Computer Society Press, pp. 422-430, 1996.

[11] R. Arnold and T. Bell, "A Corpus for the Evaluation of Lossless Compression Algorithms," *In Proceedings of the 7th IEEE Data Compression Conference*, pp. 201-210, 1997.

[12] A. Silberschatz, P.B. Galvin, and G. Gagne, *Operating System Concepts*, 6th Ed., pp. 285-287, John Wiley & Sons Inc., 2003.

[13] K.S. Yim, H. Bahn, and K. Koh, "A Compressed Page Management Scheme for NAND-type Flash Memory," *In Proceedings of the International Conference on VLSI*, pp. 266-271, 2003.

[14] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash Memory Based File System," *In Proceedings of the USENIX 1995 Winter Technical Conference*, pp. 155-164, 1995.

[15] M. Rosenblum and J.K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems*, Vol. 10, No. 1, pp. 26-52, 1992.

[16] M. Seltzer, K. Bostic, M.K. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX," *In Proceedings of the 1993 Winter USENIX Technical Conference*, pp. 307-326, 1993.