YAFFS 플래시 파일시스템의 성능과 안정성 향상

(Improving the Reliability and Performance of the YAFFS Flash File System)

손 익 준 † 김 유 미 †† 백 승 재 †† 최 종 무 †††
(Ikjoon Son) (Yumi Kim) (Seungjae Baek) (Jongmoo Choi)

요 약 Google의 Android phone이나 Apple의 iPhone과 같은 스마트폰들이 대중화됨에 따라, 플래시 메모리용 고성능 고신뢰 파일시스템에 대한 필요성이 증가되고 있다. 본 논문에서는 YAFFS(Yet Another Flash File System)의 성능 개선 및 신뢰성 향상을 위한 기법을 제안한다. 구체적으로, 파일시스템의 마운트 시간 단축 및 성능 향상을 위해 메타데이터와 유저 데이터의 분리 할당 기법을 도입하였으며, 유저 데이터의 인덱싱 정보를 메타데이터에 추가하였다. 또한 신뢰도 향상을 위해 메타 데이터 블록과 유저 데이터 블록에 대한 마모도 평준화 기법을 도입하였다. 제안된 기법은 1GB의 NAND 플래시 메모리를 가지는 시스템에서 실제 구현되었다. 실험을 통해 제안된 기법이 기존 YAFFS에 비해 6배의 마운트 시간 감소와 약 4배의 벤치마크 성능 향상 그리고, 평균 14%의 삭제 횟수 감소 및 마모도 평준화의 효과가 있음을 보인다.

키워드 : 플래시 메모리, 파일시스템, 임베디드 시스템

Abstract Popularity of smartphones such as Google Android phones and Apple iphones, is increasing the demand on more reliable high performance file system for flash memory. In this paper, we propose two techniques to improve the performance of YAFFS (Yet Another Flash File System), while enhancing the reliability of the system. Specifically, we first propose to manage metadata and user data separately on segregated blocks and indexing information piggy-back technique for reducing mount time and also enhancing performance. Second, we tailor the wear-leveling to the segregated metadata and user data blocks. Performance evaluation results based on real hardware system with 1GB NAND flash memory show that the YAFFS with our proposed techniques realized outperforms the original YAFFS by six times in terms of mount speed and five times in terms of benchmark performance, while reducing the average erase count of blocks by 14%.

Key words : Flash Memory, Filesystem, Embedded System

· 이 연구는 2009년도 단국대학교 대학원 연구보조장학금의 지원으로 이루어진 것임

† 학생회원 : 단국대학교 정보컴퓨터
ikjoon@dankook.ac.kr
†† 정 회 원 : 단국대학교 정보컴퓨터
raspberi@dankook.ac.kr
baeksj@dankook.ac.kr
††† 종신회원 : 단국대학교 정보컴퓨터 교수
choijm@dankook.ac.kr

논문접수 : 2010년 4월 26일
심사완료 : 2010년 6월 22일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제9호(2010.9)

1. 서 론

플래시 메모리는 하드디스크에 비해 소형, 경량, 저전력 소모, 고성능 및 내구성 등의 장점을 가지고 있기 때문에, 스마트폰 등의 임베디드 시스템에서 저장 장치로 주로 사용되고 있다. Apple사의 iPhone의 경우 현재 4GB에서 32GB의 NAND 플래시 메모리를 사용하고 있으며, 멀티미디어 데이터와 웹 컨텐츠 등의 데이터 증가에 따라 앞으로도 사용량이 계속 늘어날 것으로 예상하고 있다.

그러나 덮어쓰기가 불가능하며, 각 블록의 삭제 연산 횟수에 제한이 있는 등 기존 저장장치와 다른 플래시 메모리의 특성으로 인해, 이를 극복하기 위한 기법들이

많이 연구되어왔다[1]. 이들 기법은 크게 두 가지로 나뉜다. 첫 번째는 FTL(Flash Translation Layer)이라 불리는 소프트웨어를 도입하여, FAT이나 Ext3와 같은 기존의 파일 시스템들이 플래시 메모리를 디스크처럼 읽고 쓸 수 있도록 하는 것이다[2-4]. 두 번째는 플래시 메모리를 위한 전용 파일 시스템을 사용하는 것으로, YAFFS[5]나 JFFS[6] 등이 대표적이다.

NAND 플래시 메모리를 위해 사용되는 YAFFS는 Google의 Android 플랫폼을 통해 Motorola, Philips, 삼성, LG, HTC 등에서 제조되는 휴대폰 및 소형 전자 제품에 광범위하게 사용되고 있다[7].

본 논문에서는 YAFFS의 고성능 고 신뢰성 제공을 위한 기법을 제안한다. 구체적으로 YAFFS의 마운트 시간의 감소를 위해 유저 데이터(user data)와 메타데이터(metadata)를 각기 다른 블록에 나누어 저장하고, RAM 상에 구축되어 있는 메타데이터를 해당 데이터 영역 기록 시 함께 저장한다. 이를 통해 마운트 시 검색해야하는 플래시 메모리양을 줄임으로써 전체 마운트 시간을 효과적으로 감소시킨다. 이와 동시에 제안된 기법은 접근 빈도가 다른 메타데이터와 유저 데이터를 별도의 블록에 저장시킴에 따라 기존 기법 대비 높은 성능을 기대할 수 있다[8].

반면, 접근 빈도가 다른 데이터의 저장 블록을 분리함에 따라 각 블록의 삭제 횟수가 편중되는 문제가 발생할 수 있다. 이 문제를 해결하기 위해 본 논문에서는 간단하지만 효과적인 두 가지 마모도 평준화(wear leveling) 기법을 도입한다. 첫째, 삭제 횟수의 치우침이 발생하는 경우 유저데이터 블록과 메타데이터 블록을 서로 교체하여 할당하며, 둘째, 블록 할당할 시 평균보다 높은 삭제 횟수를 가지는 블록에 대한 별도의 고려를 수행한다. 이러한 두 기법은 큰 부하 없이도 특정 블록에 삭제가 치우치는 것을 효과적으로 방지한다.

제안된 기법은 YAFFS 파일시스템에 실제 구현되었으며, 400MHz XScale CPU와 64MB SDRAM, 1GB의 NAND 플래시 메모리 등이 장착된 실제 컴퓨터 시스템 상에서 성능평가를 수행하였다. 기존 YAFFS와의 비교 실험 결과 제안된 기법이 도입된 YAFFS는 기존 기법 대비 6배의 마운트 시간 단축(전원 결함 시 4배), 약 4배의 벤치마크 수행 성능 향상 및 평균 14%의 삭제 횟수 감소 효과를 볼 수 있으며, 효율적으로 각 블록 간 삭제 횟수의 편차의 평준화가 가능함을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 플래시 메모리와 YAFFS의 내부 구조에 대해 소개한다. 3장에서는 YAFFS의 성능과 신뢰성을 향상시키기 위한 기법에 대해 설명하고, 4장에서는 성능 측정 결과를 보인다. 마지막으로, 5장에서는 결론 및 향후 연구계획에 대해 기술한다.

2. 플래시 메모리와 YAFFS의 구조

그림 1은 본 논문에서 다루는 NAND 플래시 메모리의 일반적인 구조를 보이고 있다. 낸드 플래시 메모리는 여러 개의 블록으로 구성되며, 각 블록은 다시 여러 개의 페이지로 구성된다. 각 페이지의 크기는 0.5~4KB 수준이며, 각각의 페이지는 ECC나 다른 부가적인 용도를 위하여 16~64B 크기의 스페어 영역을 가진다.

플래시 메모리의 연산은 크게 읽기, 쓰기, 삭제로 나뉜다. 이때, 읽기/쓰기 연산은 페이지 단위로 수행되는 데 반해, 삭제 연산은 블록 단위로 수행되는 특징을 가진다. 플래시 메모리의 또 다른 특성은 덮어쓰기의 제한이다. 이는 EEPROM에서 파생된 플래시 메모리의 특성상 새로운 데이터를 덮어쓰기 위해서는 반드시 삭제 연산이 선행되어야 하기 때문이다. 또한 삭제 횟수가 제한

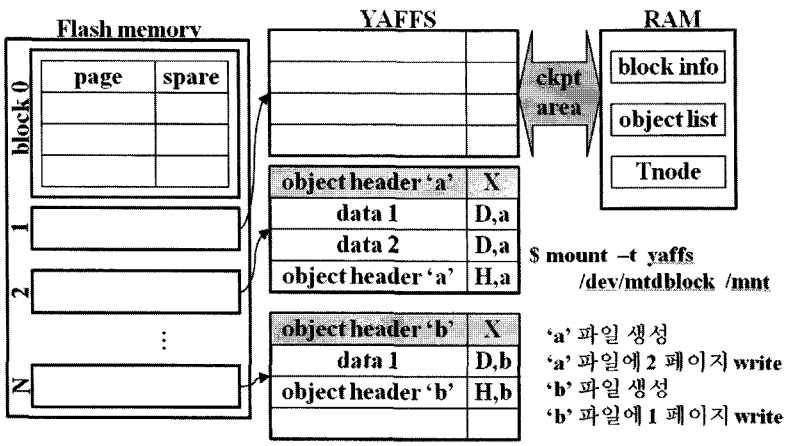


그림 1 플래시 메모리와 YAFFS의 구조

되어 있는데, 일반적으로 MLC(Multi-Level Cell)의 경우 10,000회, SLC(Single-Level Cell)의 경우 100,000회 정도의 제한을 가지고 있다.

이러한 플래시 메모리의 제약 사항을 해결하기 위해 YAFFS는 LFS(Log-structured File System)[9]에서 제안된 out-of-place 업데이트 기법을 도입하였다. 구체적으로, 업데이트 요청이 발생했을 때, 새로운 페이지를 할당받아 업데이트된 데이터를 기록하고, 기존의 페이지는 무효화(invalidate)시킨 뒤, 매핑 정보를 갱신한다. 무효화된 페이지는 추후 가비지 컬렉션(garbage collection) 작업을 통해 다시 할당 가능한 상태로 바뀐다.

파일 생성 및 데이터 기록 시 YAFFS의 데이터 저장 구조를 그림 1에 나타내었다. YAFFS는 파일 이름이나 크기와 같은 메타데이터의 저장을 위해 *yaffs_ObjectHeader* 라는 이름의 자료구조를 파일 당 관리하며, 이는 유저 데이터를 저장하기 위한 페이지와 별도의 페이지에 기록된다. 플래시 메모리에 기록된 모든 페이지는 *tag* 라 불리는 추가적인 정보를 스페어 영역에 기록하게 된다. *tag* 에는 *object id, chunk id, sequence number* 등이 포함된다. *object id* 는 해당하는 페이지를 소유하는 파일을 식별하기 위해, *chunk id* 는 파일에서 현재 페이지의 위치를 기록하기 위해 사용된다. 마지막으로 *sequence number* 는 *object id*와 *chunk id* 가 동일한 페이지가 복수 개 있는 경우 유효한 페이지를 확인하기 위해 사용된다.

마운트 작업이 수행되는 경우 YAFFS가 플래시 메모리에 저장되어 있는 정보를 바탕으로 RAM에 구축하는 정보는 크게 *yaffs_Object, yaffs_Tnode* 및 *block info* 등으로 나뉜다. *yaffs_Object* 는 메타데이터인 *yaffs_ObjectHeader* 의 내용을 읽어 들여 구축되며, 디렉토리 구조에 따라 계층 형식으로 관리된다. 각각의 파일 데이터는 *yaffs_Tnode* 구조를 이용하여 인덱스 되는데, 이는 각 데이터 페이지의 스페어 영역(*tag*)을 읽음으로써 구축된다. 또한 모든 플래시 메모리 영역을 스캔하면서 얻게 된 각 블록과 페이지의 상태 정보를 바탕으로 *block info* 구조를 구축한다. YAFFS는 마운트 연산 수행 시마다 이러한 자료구조를 구축해야 하기 때문에 상당한 시간이 소요되는 문제점이 존재한다. 이러한 오버헤드를 줄이기 위해 체크포인트 기술을 도입함으로써 마운트 시에 스캔 과정을 생략한 새로운 YAFFS2가 소개되었다[5]. 그러나 갑작스러운 전원 결함(power failure)이 발생하는 경우에는 플래시 메모리 전체 영역을 스캔해야 하는 문제점이 여전히 존재하며, 이는 불시의 전원 결함이 상시 발생 가능한 소형 단말기에서 취약점을 가진다.

3. 고성능 · 고신뢰 플래시 파일시스템 설계

3.1 유저 데이터와 메타데이터 분리

YAFFS의 마운트 시간단축과 성능 향상을 위해 본 논문에서 제안된 기법을 그림 2에 보였다. 구체적인 기법은 크게 세 가지로 구분된다. 첫째, 메타데이터와 유저 데이터가 저장되는 블록을 분리하였다. 이를 통해 전체 플래시 메모리를 스캔하는 대신 한정된 개수의 메타데이터 블록만을 읽음으로써 yaffs_object 구조를 RAM 상에 구축할 수 있게 된다.

이러한 할당 블록의 분리는 성능 향상의 장점을 가져온다. 이는 메타데이터와 유저 데이터의 갱신 빈도가 다르다는 특성에 기인한다. 구체적으로, 메타데이터는 파일의 크기나 접근 시간과 같은 다양한 정보들을 가지고 있으므로, 유저 데이터의 일부만이 변경되더라도 항상 새로 기록되어야 한다. 따라서 빈번하게 변경되는 메타데이터와 그렇지 않은 유저 데이터를 분리함으로써 가비지 컬렉션 성능의 향상이 가능하며, 이는 파일 시스템 자체의 성능향상을 가져온다[8].

둘째, *yaffs_Tnode* 데이터를 구축하기 위해 본 논문에서는 *yaffs_ObjectHeader* 데이터를 플래시 메모리에

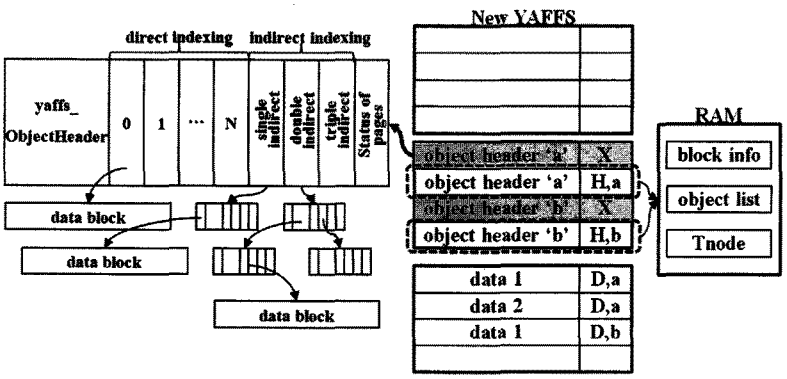


그림 2 새로운 형태의 YAFFS 구조

기록할 때, 유저 데이터에 대한 인덱스 정보를 함께 기록하였다. 구체적인 기록 방법은 다음과 같다. *yaffs_ObjectHeader*는 하위 호환성을 위해 0.5KB 페이지를 기준으로 작성되어 있다. 따라서 최근 사용되는 2KB 혹은 4KB 페이지 구조의 플래시 메모리에 YAFFS가 운용되는 경우 1.5KB 이상의 공간이 사용되지 않고 남아있게 됨을 의미한다. 따라서 본 연구에서는 비 사용되는 공간에 inode 구조를 응용하여 다수의 direct index pointer와 3개의 indirect index pointer를 삽입하여 인덱스 정보를 저장했다. 이를 통해 모든 유저 데이터 페이지의 스페어 영역에 접근하지 않고도 *yaffs_Tnode* 구조를 구축할 수 있도록 해준다.

셋째, 전체 플래시 메모리의 스캔 없이 *block info*를 구축하는 기법을 도입하였다. 이는 두 가지 구현이 가능하다. 첫째는 *block info* 구조를 별도의 플래시 메모리 공간에 기록하는 것이고, 둘째는 모든 블록의 첫 번째 페이지만을 읽음으로써 블록의 상태를 확인할 수 있도록 데이터를 저장하는 것이다. 현재 버전에서는 두 기법을 모두 구현함으로써 블록 정보를 빠르게 구축함과 동시에 갑작스러운 전원 결함 시에도 정확한 블록 정보를 구축할 수 있도록 하였다.

3.2 마모도 평준화 기법

메타데이터와 유저 데이터를 각기 다른 블록에 나누어 저장하는 경우 각 데이터의 업데이트 빈도가 다르기 때문에 특정 블록에 삭제 연산이 편중되는 문제를 야기할 수 있다. 본 논문에서는 이러한 문제를 해결하기 위해 간단하지만 효과적인 마모도 평준화 기술을 도입하였다. 도입된 기법은 두 단계로 이루어진다. 첫째, 새로운 블록 할당이 요청되었을 때, 블록이 어떤 데이터의 저장을 위해 사용될 것인지를 판단한다. 메타데이터를 저장하기위해 사용된다면 유저 데이터 저장에 사용되었던 블록 중에서 할당 후보 블록을 선택하며, 반대의 경우에는 메타데이터의 저장을 위해 사용되었던 블록 중에서 할당 후보 블록을 선택한다. 이는 메타데이터와 유저 데이터 저장을 위해 사용되는 블록을 맞바꾸는 효과를 가진다.

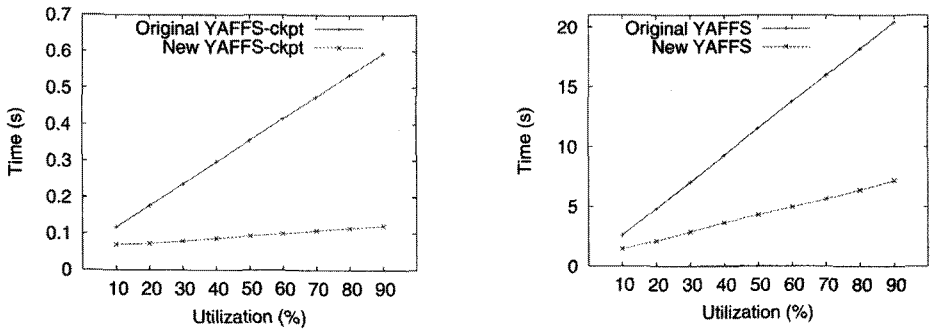
둘째, 할당 후보 블록들의 삭제 횟수를 확인한다. 만약 삭제 횟수가 전체 블록의 평균 삭제 횟수보다 많다면, 다음 번 후보 블록의 삭제 횟수를 확인하여 둘 중 삭제 횟수가 적은 블록을 할당한다. 이는 파일시스템 수행 성능에 영향을 주지 않으면서, 특정 블록에 삭제 연산이 지나치게 편중되는 현상을 줄여 주며 파일시스템의 워크로드에 따라 메타데이터와 유저 데이터에 사용될 블록을 동적으로 재분배할 수 있다는 장점을 가진다.

4. 실험 결과 및 분석

제안된 기법은 YAFFS에 실제 구현되었으며, 400MHz XScale CPU와 64MB SDRAM, 1GB의 낸드 플래시 메모리가 장착된 컴퓨터 시스템에서 실제 실험되었다. 이를 위해 해당 시스템에 리눅스 커널 2.6.28 버전을 포팅하였다. 실험을 통해 기존 YAFFS와 본 논문에서 제안된 기법이 적용된 YAFFS 간의 마운트 시간 측정 및 가비지 컬렉션 부하 비교, 각 블록들의 삭제 횟수 분산을 측정하였다.

4.1 성능 측정

그림 3은 다양한 이용률(utilization) 상에서 측정된 기존 YAFFS와 새로운 YAFFS간의 마운트 시간 비교 결과를 보여준다. 이용률은 Postmark 벤치마크[10]를 실행함으로써 생성되었다. 이 벤치마크는 랜덤한 크기의 많은 파일들을 생성한 후, 해당 파일들에 대해 읽기/쓰기/삭제/덧붙이기 트랜잭션을 수행한다. 실험에서는 10%의 이용률을 채우기 위해 평균 파일 크기 512KB, 초기 파일 개수 100개, 그리고 800회의 트랜잭션을 수행하도록 설정하였다. 그림 3은 전원 결함으로 인해 체크포인트 기술이 적용되지 않는 경우와, 체크포인트가 적용 가능한 경우의 결과를 각각 보여주고 있다. 전원 결함이



(a) 전원 결함이 없는 경우 (체크포인트 사용) (b) 전원 결함이 발생한 경우 (체크포인트 사용 불가)

그림 3 이용률별 마운트 시간

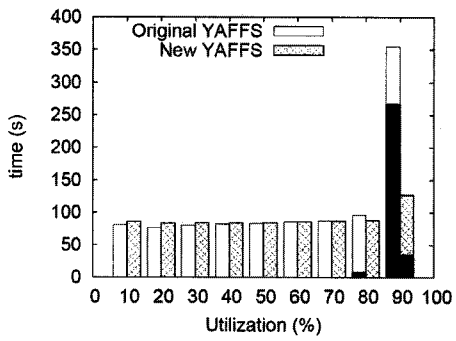


그림 4 이용률별 성능 측정

발생한 경우(그림 3의 (b)), 새로운 YAFFS는 전체 페이지를 스캔하지 않기 때문에 기존 YAFFS에 비해 약 4배 가량 단축된 마운트 시간을 보이고 있다. 체크포인트가 사용되는 일반 적인 경우에도(그림 3의 (b)), 새로운 YAFFS는 효과적인 마운트 시간의 감소가 가능함을 확인할 수 있다.

또한 Postmark 벤치마크의 반응 시간을 측정하여 그림 4에 나타내었다. 반응 시간에는 단순히 Postmark의 실행 시간뿐만 아니라 실행 중의 가비지 컬렉션에 소요된 시간도 포함되므로 이들 각각의 시간을 그래프에 진한 색상으로 표시하였다. 이용률이 90%를 넘게 되는 경우, 할당 가능한 블록의 부족으로 인해 YAFFS는 보다 적극적으로 가비지 컬렉션을 수행하는 aggressive 모드로 동작하게 된다. 이 경우 새로운 YAFFS는 기존의 YAFFS에 비해 약 4배의 성능 차이를 보임을 알 수 있다. 이는 제안된 기법이 갱신 빈도가 다른 메타데이터와 유저데이터를 분리하여 저장함으로써 가비지 컬렉션 오버헤드를 효과적으로 줄였음을 의미한다.

4.2 마모도 평준화 확인

그림 4에 보인 Postmark 벤치마크 수행 중 각 블록의 삭제 연산 횟수를 측정하여 평균 삭제 횟수, 최대/최소 삭제 횟수 및 각 블록별 삭제 횟수의 분포를 그림 5에 나타내었다. 새로운 YAFFS에서, 평균 삭제 횟수는 5.54회 이고 표준 편차는 3.41인 반면, 기존 YAFFS의 평균 삭제 횟수는 6.35, 표준 편차는 4.09로 측정되었다.

실험 결과를 통해 다음과 같은 사실을 확인할 수 있다. 첫째, 본 논문에서 제안한 기법이 적용된 YAFFS는 효과적인 데이터 배치를 통해 가비지 컬렉션의 횟수 감소 및 이를 통한 평균 삭제 횟수의 감소가 가능하다. 둘째, 제안된 마모도 평준화 기법에 의해 블록간의 삭제 횟수의 차이가 감소하였으며, 이는 제안된 기술이 플래시 메모리의 마모도 평준화 및 이를 통한 신뢰성 향상에 효과적임을 의미한다.

5. 결론 및 향후 연구

본 논문에서는, 플래시 메모리 파일시스템의 성능 향상 및 마운트 시간 감소 그리고 마모도 평준화 기법을 제안하였다. 실제 플래시 메모리 상에서 진행된 실험 결과 제안된 기법이 마운트 속도의 감소 및 가비지 컬렉션 수행 시간의 향상을 가져왔으며, 블록 간의 삭제 횟수 차이를 감소시켜 신뢰성을 향상시킴을 확인 할 수 있었다.

향후 본 연구는 두 가지 방향의 확장을 고려하고 있다. 첫째, 새로운 데이터 분류 기법의 도입이다. 본 논문에서는 블록을 분리 할당함에 있어서 데이터가 메타데이터인지 여부만을 고려하였다. 그러나 파일의 확장자나 갱신 빈도 등 다른 속성들을 활용한다면 가비지 컬렉션 성능을 보다 향상 시킬 수 있을 것이다. 둘째, 최근 TLC(Triple-Level Cell)나 QLC(Quadruple-Level Cell)와 같은 대용량 플래시 메모리의 경우 1,000번 이하의 삭제연산만이 가능하다. 또한 본 논문에서 제안한 할당 기법은 할당 후보 블록들의 삭제 횟수가 평균보다 높은 경우에 대비한 데이터의 동적 이주(migration)기법이 필수적이다. 따라서 이를 위한 더욱 세밀한 마모도 평준화 기술을 설계하는 것이다.

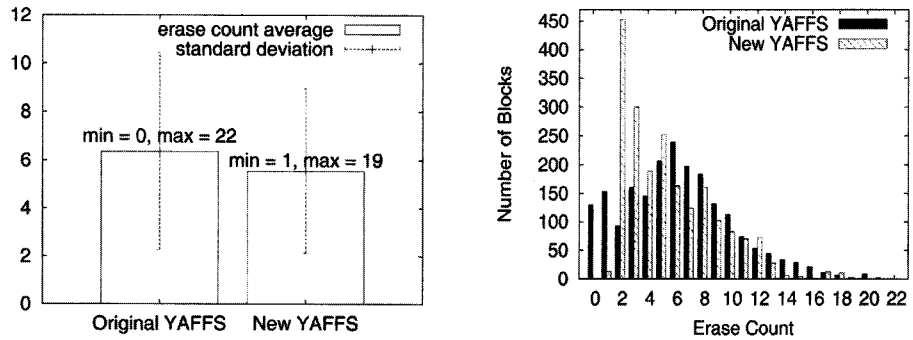


그림 5 마모도 평준화 측정 결과

참 고 문 헌

[1] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," *ACM Comput. Surv.*, 37(2):138-163, 2005.

[2] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pp.229-240, New York, NY, USA, 2009. ACM.

[3] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, "A Space-efficient Flash Translation Layer for Compactflash Systems," *Consumer Electronics, IEEE Transactions on*, 48(2):366-375, May 2002.

[4] M. S. Kwon, S. H. Bae, S. S. Jung, D. Y. Seo, and C. K. Kim, "KFAT: Log-based Transactional FAT File System for Embedded Mobile Systems," *In in 2005 US-Korea Conference*, pages ICTS-142, 2005.

[5] Aleph One, YAFFS: Yet another flash file system, "http://www.yaffs.net".

[6] D. Woodhouse, "JFFS: The Journaling Flash File System," In Ottawa Linux Symposium, 2001.

[7] eCosCentric, "http://www.ecoscentric.com/middlewar r/yaffs.shtml".

[8] S. Baek, S. Anh, J. Choi, D. Lee and S. H. Noh, "Uniformity Improving Page Allocation for Flash Memory File Systems," In *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software*, pp.154-163, 2007.

[9] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-structured File System," *ACM Trans. Comput. Syst.*, 10(1):26-52, 1992.

[10] J. Katcher, "Postmark: A New File System Benchmark," Techinical Report TR3022, Network Appliance Inc., 1997.

손 익 준

2007년~현재 단국대학교 정보·컴퓨터학부 학사과정. 관심분야는 운영체제, 임베디드 시스템, 차세대 저장장치 등

김 유 미

2008년 단국대학교 전기전자컴퓨터공학부 졸업(공학사). 2009년 단국대학교 대학원 컴퓨터학과(공학석사). 관심분야는 운영체제, 임베디드 시스템

백 승 재

2005년 단국대학교 컴퓨터공학과 졸업(공학사). 2004년~현재 비트 컴퓨터 강사. 2007년 단국대학교 대학원 정보컴퓨터 과학과(이학석사). 2010년 단국대학교 대학원 컴퓨터학과(공학박사). 관심분야는 운영체제, 임베디드 시스템, 차세대 저장장치 등

최 종 무

1993년 서울대학교 해양학과 졸업(이학사). 1995년 서울대학교 대학원 컴퓨터공학과(공학석사). 2001년 서울대학교 대학원 컴퓨터 공학과(공학박사). 2001년~2003년 유비쿼스 주식회사 책임 연구원 2003년~현재 단국대학교 공과대학 컴퓨터학부 컴퓨터공학 전공 부교수. 2005년~2006년 UC Santa Cruz 방문 교수. 관심분야는 운영체제, 임베디드 시스템, 차세대 저장장치 등