Android 플랫폼 기반 플래시 메모리 관리 기법에 대한 성능 평가

김윤아, 오기환, 김강년, 강운학, 이상원
성균관대학교 컴퓨터공학과
e-mail : kya121247@gmail.com,

Performance Evaluation of Flash Memory Management Schemes on Android Platform

Yun-A Kim, Gi-Hwan Oh, Kang-Nyeon Kim, Woon-Hak Kang
Sang-Won Lee
Dept of Computer Engineering, Sung Kyun Kwan University

요 약

스마트폰에서 낸드 플래시 메모리가 저장 장치로 사용됨에 따라 다양한 플래시 파일 시스템과 플래시 변환 계층들이 제시되었다. 플래시 메모리는 덮어 쓰기가 불가능하기 때문에 이들 기법들은 기본적으로 로그 기반 구조를 취하고 있지만 가비지 수집, 데이터 배치 정책의 설계에 따라 성능과 수명 관리 측면에서 많은 차이를 보인다. 본 논문은 플래시 메모리 관리 기법들의 다양한 설계가 성능에 미치는 영향을 알아보고 종합적으로 비교 해보기 위해 대표적인 스마트폰 플랫폼인 안드로이드상에서 시뮬레이션 기반의 성능 평가를 수행 한다. 또한 각 기법들의 설계가 성능에 미치는 영향을 분석 한다.

1. 서론

최근 스마트폰 시장 규모가 커지면서 관련 기술들에 대한 연구가 활발해지고 있다. 기존 휴대폰이 전화, 단문 메시지 중심의 제한적인 서비스를 제공한 반면, 스마트폰은 사용자가 원하는 어플리케이션을 자유롭게 설치 할 수 있는 환경을 제공 한다. 최근 스마트폰을 통해 대용량 멀티미디어 파일 재생이나 3D 게임처럼 과거의 휴대폰에서 지원하기 어려웠던 어플리케이션을 사용하는 사용자가 늘어나면서, 스마트폰 환경에서의 데이터 크기 또한 꾸준히 증가하고 있는 추세이며 스마트폰 벤더들은 이러한 사용자 데이터를 수용하기 위해 낸드 플래시 메모리를 사용하고 있다.

플래시 메모리는 성능, 소음, 전력 소비, 충격에 대한 내구성 측면에서 모두 기존의 디스크보다 우수한 특성을 보인다. 2000년대 이후부터 디지털 카메라와 USB 플래시 드라이브 등에 본격적으로 탑재되기 시작했고 최근에는 스마트폰을 비롯한 각종 모바일 기기 시장에서 디스크를 거의 대체했다. 또한 플래시 SSD의 출시로 랩톱과 서버 환경에서도 차세대 저장 장치로 주목 받고 있다.

하지만 낸드 플래시 메모리는 덮어 쓰기가 불가능하고 제한적인 수명을 갖는 등 디스크와 특성이 다르기 때문에 기존 디스크 기반 저장 장치 관리 기법을 적용하는 것이 비효율적이다. 이러한 문제를 해결하기 위해 플래시 파일 시스템과 플래시 변환 계층(FTL)과 같은 기법들이 다양하게 제시 되었고 이들은 설계 방식에 따라 성능, 수명 관리 등에 있어 각기 다른 특성을 보이고 있다.

본 논문은 대표적인 플래시 메모리 관리 기법들이 취하고 있는 다양한 설계가 성능에 미치는 영향을 분석하기 위해 시뮬레이션 기반의 성능 평가를 수행 한다. 이를 위해 안드로이드 플랫폼 상에서 IOZone 벤치마크를 통해 워크로드를 생성하고, 이를 각 파일 시스템 및 FTL에서 처리하는 과정을 프로파일링 하여 수행 시간과 가비지 수집 부담을 정량적으로 측정 한 후, 결과를 비교 분석 한다.

2. 관련 연구

2.1 NAND Flash Memory

플래시 메모리는 EEPROM의 일종으로 비휘발성 메모리다. 구현 방식에 따라 NOR 플래시와 NAND 플래시 메모리로 구분 할 수 있으며, NOR 플래시 메모리는 바이트 단위 접근, Execution In-Place (XIP)등의 특성으로 인해 코드 저장 및 실행 용도로 사용 된다. NAND 플래시 메모리는 집적도가 높기 때문에 저장 장치에 사용 된다.

낸드 플래시 메모리 칩은 다수의 블록으로 구성되며, 하나의 블록 내에는 여러 페이지가 들어 있다. 페이지는 데이터 영역과 메타 정보 기록을 위한 Spare 영역으로 구분된다. Spare 영역은 메타 정보를 저장하기 위해 사용 된다. 낸드 플래시 메모리는 덮어 쓰기가 불가능하고 데이터는 반드시 삭제된 블록에 쓰여야 하는 제약이 있다. 삭제는 블록 단위로 수행 되고, 데이터의 읽기와 쓰기는 페이지 단위로 수행 된다.

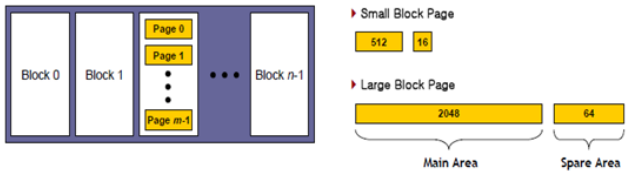

그림 1. NAND 플래시 메모리 구조

낸드 플래시 메모리는 덮어 쓰기가 불가능한 특성 때문에 디스크 기반의 소프트웨어와 호환이 불가능하다. 이를 해결하기 위한 접근 방법으로 플래시 파일 시스템을 이용하거나 플래시 변환 계층을 사용하는 방법 등이 있다. 일반적으로 전자는 기기에 내장된 플래시 메모리 칩을 관리하기 위해 사용되고, 후자는 MMC카드나 플래시 SSD와 같이 플래시 메모리 기반 저장 장치의 컨트롤러에 내장되는 경우가 많다. 또 다른 중요한 특성은 제한적인 수명이다. 최대 삭제 회수 이상으로 블록을 삭제 하면, 해당 블록에 대한 신뢰성이 떨어져 사용할 수 없다. 일반적으로 SLC 플래시 칩의 경우, 블록 당 100만 회, 집적도가 높은 MLC 칩은 블록 당 10만 회 수준으로 플래시 기반 저장 장치의 수명 확장을 위해 모든 블록을 가급적 고르게 사용하는 웨어레벨링 기능이 필수적이다.

2.2 Flash File System

플래시 파일 시스템은 상위 계층에 덮어 쓰기가 불가능한 플래시 메모리의 특성을 감추고 어플리케이션에 디스크 파일 시스템과 동일한 파일 인터페이스를 제공 한다. 지금까지 제안된 YAFFS2(Yet Another Flash File System2), JFFS가 대표적이다. 플래시 파일 시스템은 MTD 인터페이스로 플래시 메모리를 관리 한다. YAFFS2는 플래시 메모리의 각 페이지를 Object Header 페이지(Inode)와 데이터 페이지로 구분하고, 이를 Spare 영역(Out-Of-Bound,OOB)에 기록 한다.[1] 파일은 여러 개의 Chunk로 구성 되며 각 Chunk들의 위치는 메모리에 상주하는 Tnode-Tree를 통해 찾는다. Tnode-Tree는 마운트 시점에 생성되고 각 Tnode는 파일 Chunk 위치를 가리킨다. 파일의 특정 위치에 덮어 쓰기가 발생하면 기존 데이터는 무효 처리하고 빈 공간에 데이터를 쓰고 Tnode의 주소 정보를 갱신 한다. 무효 데이터들은 가비지 수집 연산을 통해 삭제 한다.

2.3 Flash Translation Layer

파일 인터페이스의 호환성을 유지하는 플래시 파일 시스템과 달리 플래시 변환 계층의 목적은 플래시 메모리를 디스크처럼 덮어 쓰기 가능한 블록 장치로 에뮬레이션 하는 것이다. 따라서 플래시 변환 계층을 내장한 MMC카드, SD카드, 플래시 SSD와 같은 플래시 저장 장치들은 운영체제 입장에서는 디스크와 동일한 저장 장치로 보이고 블록 인터페이스로 제어 할 수 있으며 EXT나 FAT과 같은 디스크 파일 시스템을 사용할 수 있다.

덮어 쓰기가 불가능한 특성을 감추기 위해 플래시 변환 계층은 내부적으로 주소 매핑 기법을 사용 한다. 상위 계층에서 특정 LBA에 데이터를 덮어 쓰면, FTL은 내부적으로 빈 페이지에 새로운 데이터를 쓰고 매핑 정보를 새 데이터의 위치로 갱신함으로써 데이터가 덮어 쓰여진 것처럼 처리 한다. 덮어 쓰기 이전의 페이지는 무효 처리되고 가용 공간이 부족해지면 가비지 수집을 통해 삭제된다. 가비지 수집 과정에서 가급적 블록을 균등하게 삭제하기 위해 웨어레벨링 정책을 적용하여 플래시 메모리의 수명을 연장한다.

주소 매핑 단위에 따라 FTL은 크게 블록 단위 매핑, 페이지 단위 매핑, 하이브리드 매핑 기법으로 분류 할 수 있다. 블록 단위 매핑은 주소 매핑 정보를 블록 단위로 관리하기 때문에 매핑 정보의 크기를 줄어 메모리 사용량이 적은 것이 장점이지만, Small Random Write에 취약한 문제를 보인다. 페이지 레벨 매핑은 비교적 공간 활용이 유연하기 때문에 랜덤 쓰기에 취약한 문제가 해결되지만, 주소 매핑 정보의 크기가 커진다. 하이브리드 매핑 기법은 중간 형태로, 블록 레벨 매핑을 기본 구조로 취하지만, Write Buffer 용도의 Over Provisioning 영역에 한해 페이지나 섹터 단위의 매핑 기법을 적용 한다. 대표적인 연구로 BAST[2]와 FAST[3], 등을 들 수 있다.

BAST는 플래시 메모리 공간을 데이터 영역, 쓰기 버퍼 목적의 로그 영역, 삭제된 블록 집합인 Free 영역으로 구분 하고, 데이터 영역에 대해 블록 레벨 매핑, 로그 영역에 대해 페이지 레벨 매핑을 적용 한다. 로그 블록은 데이터 블록에 대한 쓰기 버퍼로서, 데이터 블록에 대한 모든 쓰기 요청은 관련된 로그 블록에 순차적으로 기록 된다. 로그 블록 내의 매핑 정보는 별도의 매핑 테이블로 관리 된다. 덮어 쓰기가 발생하면 이전 페이지는 무효 처리되고 로그 블록에 Append 된 후, 매핑 정보를 갱신 한다. 로그 블록이 꽉 차면 원본 데이터와 병합한다. BAST는 Compact Flash 처럼 하드웨어 자원이 충분치 않은 임베디드 저장 장치용으로 설계 되었는데, 이러한 BAST의 설계는 Small Random Write 워크로드에 대해 Log Block Thrashing 문제를 야기 한다. Log Block Thrashing은 가용 로그 블록이 없을 때, 새로운 쓰기 요청이 오면, 활용률이 떨어지는 Victim 로그 블록이 빈번하게 병합되는 현상을 지칭 한다.

FAST는 OLTP처럼 Small Random Write가 많은 워크로드 처리 시 Log Block Thrashing을 해결하기 위해 데이터 블록과 로그 블록의 1:1관계를 모든 데이터 블록이 로그 영역 전체를 공유하도록 변경 했다. 모든 Write는 로그 영역에 Append 되며, 로그 영역 전체에 대해 Page Level Mapping을 적용 한다. 따라서 Random Write로 인한 Log Block Thrashing 문제는 완화 될 수 있다. 하지만 FAST는 로그 영역이 꽉 찼을 때, 병합 연산에 관여하는

블록 개수가 많아지기 때문에 병합 연산 비용이 크게 증가 하고 이때의 실시간 응답성이 떨어지는 문제를 내재하고 있다. 또한 읽기 요청 때마다, 로그 영역 전체에서 유효 페이지를 찾아야 하는 오버헤드도 존재 한다. FASTer는 FAST의 성능을 개선한 것으로 로그 영역의 활용도를 높이기 위해 Second Chance 정책을 적용한다.[4]

2.4 Android Platform

스마트폰 시장의 주도권을 차지하기 위해 마이크로소프트, 애플, 구글과 같은 IT기업들이 자사의 OS를 채택한 스마트폰을 출시하고 있다. 구글은 Android라는 오픈 소스 플랫폼을 공개하고 휴대폰 제조업체와의 협력을 통해 스마트폰 시장 점유율을 높여가고 있다. Android는 리눅스 커널과 Dalvik 미들웨어, 어플리케이션 지원을 위한 각종 라이브러리, 핵심 어플리케이션과 같은 휴대폰 구동에 필요한 요소들을 대부분 포함하고 있으며 현재 스마트폰용으로는 Android 2.3 Ginger Bread 버전이 공개된 상태이다[5]. 안드로이드 프로젝트는 스마트폰 구동에 필요한 소프트웨어 스택뿐만 아니라, 개발 도구인 SDK를 함께 배포 한다. 또한 QEMU를 통해 모바일 기기가 없어도 안드로이드 가상 머신상에서 어플리케이션 개발을 할 수 있도록 지원한다.

3. 본론

본 장에서는 성능 평가를 위한 시뮬레이션 방법을 설명하고 실험 결과를 분석 한다. 실험을 위해 안드로이드 에뮬레이터 상에서 IOZone[6]을통해 1) 순차 읽기 2) 순차 쓰기 3) 랜덤 읽기/쓰기를 워크로드를 생성 하고 이를 처리하는 과정에서 발생하는 플래시 칩 레벨의 읽기, 쓰기, 삭제 및 가비지 수집 정보를 프로파일링을 통해 얻는다. 이러한 정보를 기반으로 아래 표의 수식에 따라 Elapsed Time를 도출 한다.

Elapsed Time =
(Cr*Tr) + (Cw*Tw) + (Gc*Tw) + (Ce * Te)

Cr : 페이지 읽기 회수
Cw : 페이지 쓰기 회수
Ce : 블록 삭제 회수
Gc : Copyback 회수
Tr : 페이지 읽기 Latency (20us)
Tw : 페이지 쓰기 Latency(250us)
Te : 블록 삭제 Latency(1.5ms)

성능 평가 대상은 플래시 파일 시스템인 YAFFS2와 FAST, FASTer, BAST, Page Level Mapping FTL 이다. YAFFS2의 프로파일링 정보는 관련 구조체에서 구할 수 있다. FTL은 가상 SD카드에 대한 I/O 요청 내용을 모두 수집하고 이를 FTL 시뮬레이터에 전달하면, 시뮬레이터가 각 FTL의 동작 특성에 따라 표 3.의 Elapsed Time을 산출 한다.

그림 2. 순차 읽기 소요 시간

IOZone의 100MB 순차 읽기에 소요된 시간을 나타낸다 (PLM : Page Level Mapping). 전반적으로 FTL은 편차가 없었으며 YAFFS2가 근소하게 빠른 것으로 나타났다. 읽기 성능은 플래시 메모리 칩의 특성 자체가 빠르기 때문에 큰 성능 차이가 나지 않는 것으로 파악 된다.

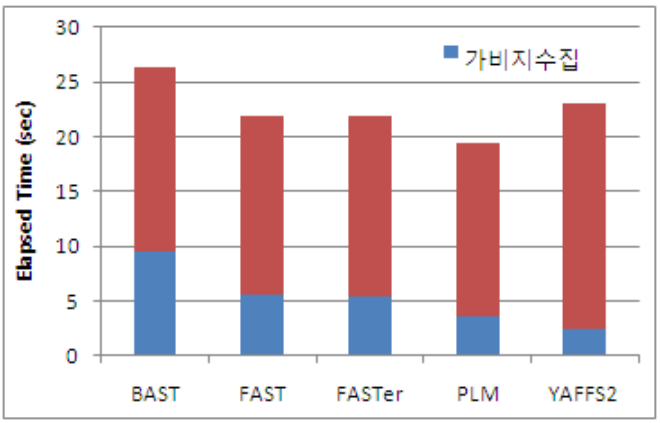

그림 3. 순차 쓰기 및 가비지 수집 소요 시간

100MB 순차 쓰기에 소요된 시간이다. 순차 쓰기 실험 전에는 플래시 메모리 전 영역에 대해 랜덤 쓰기를 미리 해두어(Aging) 정확한 Runtime 성능을 보고자 했다. 순차 쓰기 또한 각 기법이 큰 차이를 보이지 않는다. 페이지 레벨 매핑 FTL이 가장 빠르지만 FAST와의 차이는 10% 이내로 미미하다. 대부분의 기법들이 순차 쓰기는 가비지 수집의 큰 부담 없이 처리 할 수 있지만 그림 3의 결과는 이전 Aging의 영향이 반영 된 것이다. 로그 영역 활용에 있어 유연성이 떨어지는 BAST가 가장 느린 것으로 나타났고 상대적으로 로그 영역 활용율이 높은 FAST와 FASTer가 BAST보다 가비지 수집 부담이 절반으로 줄었

다. BAST와 FAST의 전체 성능 차이는 가비지 수집에 소요된 성능 차이 때문으로 관찰 되었다.

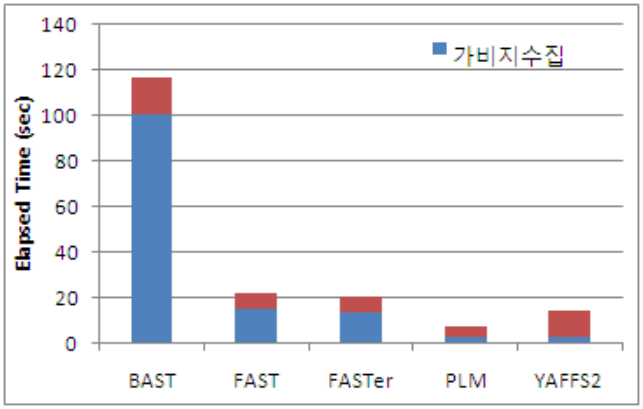

그림 4. 랜덤 읽기/쓰기 및 가비지 수집 소요 시간

순차 쓰기의 가비지 수집 부담은 평균적으로 23%였지만 랜덤 쓰기 시에 가비지 수집 부담은 평균 56%로 급상승한다. 그림 4에 의하면 플래시 메모리 관리 기법의 성능 차이가 가장 크게 나타나는 패턴이 랜덤 쓰기 인 것으로 확인되었다. 이는 플래시 메모리 관리 기법의 설계가 가장 큰 영향을 주는 것이 결국 랜덤 쓰기 성능 인 것으로 해석 된다. 특히 가비지 수집 성능에서 큰 차이를 보이는데 이 결과를 통해 랜덤 쓰기 성능 향상을 위해 가비지 수집 알고리즘의 설계가 가장 중요한 점을 알 수 있다.

가비지 수집 부담은 BAST의 경우 86%로 랜덤 쓰기에 취약한 문제점이 노출 되었다. FAST와 FASTer는 하이브리드 매핑임에도 불구하고 로그 영역 활용도가 높아 전체 수행 시간이 BAST의 수행 시간의 17%에 불과했다. FASTer는 지역성이 있는 랜덤 워크로드에서 보다 뛰어난 성능을 보이는 것으로 알려져 있지만 IOZone의 랜덤 쓰기 분포가 균일했기 때문에 페이지 레벨 매핑보다 3배 느린 것으로 나타났다.

4. 결론

본 논문은 다양한 낸드 플래시 메모리 관리 기법들의 성능 평가를 수행하여 I/O 처리 시간과 가비지 수집 오버헤드를 계산하고 비교 분석 했다. 이를 통해 각 기법들이 취하고 있는 가비지 수집, 웨어 레벨링, 데이터 배치등의 설계가 성능에 주는 영향을 분석 했다. 실험 결과 플래시 메모리 관리 기법의 설계가 큰 영향을 미치는 것은 랜덤 쓰기 인 것으로 나타났으며 특히 가비지 수집 성능을 크게 좌우하는 것을 확인했다. 또한 하이브리드 매핑 FTL은 로그 영역의 활용도가 가비지 수집 뿐만 아니라 전체 성능에 주는 영향이 큰 것으로 나타났다. 향후 과제로 DFTL과 UBIFS와 같은 최신 플래시 메모리 관리 기술들을 평가에 포함하는 한편, 보다 정확한 평가를 위해 에뮬레이터 환경 대신 실제 안드로이드 머신 상에서 실험을 수행할 것이다.

참고문헌

1. YAFFS. http://yaffs.net
2. 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 Transactions on Consumer Electronics Vol.48, No.2, pp.366-375, 2002.
3. S.W.Lee, D.J.Park, T.S.Chung, D.H.Lee, S.Park, and H.J.Song, "FAST: A Log-Buffer Based FTL Scheme with Fully Associative Sector Translation", ACM Transactions on Embedded Computing Systems, Vol.6, No3, pp.18, 2007.
4. S.-P. Lim, S.-W. Lee, and B. Moon. FASTer FTL for enterprise-class Flash memory SSDs. In Proc. of SNAPI,2010
5. Android Project, http://www.android.com/
6. IOZone Filesystem Benchmark, http://www.iozone.org