# 안드로이드 플랫폼 상에서 NAND 플래시 메모리 파일시스템 성능 분석

김상훈*, 이재강*, 안광선*
*경북대학교 전자전기컴퓨터학부
e-mail:ksh3000@knu.ac.kr

# A Performance Analysis of NAND Flash Memory File System on the Android Platform

Sang Hoon Kim*, Jae Kang Lee*, Kwang Seon Ahn*
*School of Electrical Engineering and Computer Science, Kyungpook National University

요 약

본 논문에서는 NAND 플래시 전용 파일 시스템인 YAFFS2와 UBIFS에 대해 성능 분석을 한다. 각 파일 시스템에 따라 mount 시간 측정, read/write의 속도 측정, 메모리 소모량에 대하여 비교 분석하였다. 그 결과 clean mount와 unclean mount에서는 UBIFS가 YAFFS2보다 약 49% 이상 향상되었다. 또한 read/write성능 분석에서 read성능은 큰 차이가 없었지만, write성능은 UBIFS가 압도적으로 향상 되었다는 것이 증명되었다. 반면, UBIFS가 YAFFS2에 비해 메모리 소모량은 약 26% 이상 높았다.

## 1. 서론

정보통신의 발달로 다양한 디바이스 장치들이 개발되어지고 있다. 그 중 스마트 폰이 대중화되고 그 종류 또한 늘어나면서 플래시 메모리가 기본 저장 매체로 많이 사용되고 있다. 플래시 메모리는 기존의 저장매체와는 달리 높은 안정성과 고용량을 저장할 수 있다는 장점으로 인해 많은 임베디드 시스템의 대표적인 저장장치로 자리매김하고 있다. 플래시 메모리는 데이터를 쓴 페이지의 블록을 지우기 전까지 다시 사용할 수 없다는 단점이 있다. 또한 읽기 속도는 매우 빠르지만, 쓰기 속도와 삭제 속도가 다른 저장매체와 비교해 상대적으로 느리며, 한 번에 지울 수 있는 크기가 일정하고 삭제 연산의 횟수가 제한되어 있다. 플래시 메모리는 NOR와 NAND로 나눌 수 있다. NOR의 경우에는 NAND에 비해 빠른 읽기속도를 가지는 것이 특징이다. 그러나 가격이 높아 대용량 저장매체로 사용되기 어렵다. 이러한 상황에서 현재 NAND 플래시 메모리 전용 파일시스템들이 어느 정도의 성능을 나타내는지 파악해야할 필요성이 있다. 본 논문에서는 안드로이드 플랫폼 상에서의 NAND 플래시 전용 파일 시스템인 YAFFS2와 UBIFS에 대한 성능을 비교 분석한다.

## 2. 관련 연구

### 2.1 Android Platform

안드로이드(Android)는 휴대 장치를 위한 운영체제와 미들웨어, 사용자 인터페이스 그리고 표준 응용 프로그램을 포함하고 있는 스택이다. 안드로이드는 리눅스 커널 위에서 동작하며, 다양한 안드로이드 시스템 구성요소에 사용되는 C/C++ 라이브러리들을 포함하고 있다. 또한, Dalvik 가상머신을 통해 자바로 작성되어진 응용 application을 별도의 프로세스에서 실행하는 구조로 되어있다. 안드로이드 플랫폼의 아키텍처는 크게 application, framework, library, android Run-time, 리눅스 커널로 구성되어져 있다. 안드로이드는 기존의 리눅스 커널을 기반으로 Dalvik 가상머신과 자바 core 라이브러리를 추가했다는 점에서 차이점이 있다. 이는 리눅스 커널에서 확장된 형태라고 볼 수 있다.

### 2.2 NAND Flash 메모리

플래시 메모리에서 데이터를 읽고 쓰는 기본적인 단위는 페이지 단위이며, 한 페이지의 크기는 약 512 ~ 2,048 Byte이다. 각각의 페이지는 Bad 블록 등의 에러를 보정하고, 관련 메타 데이터를 보관할 수 있는 여분의 공간(Out-Of-Band)을 가지고 있다. 읽기와 쓰기는 페이지 단위이며, 지우기는 블록 단위이다. <표 1>은 플래시 메모리의 기본 연산 수행 시간을 보여준다[1].

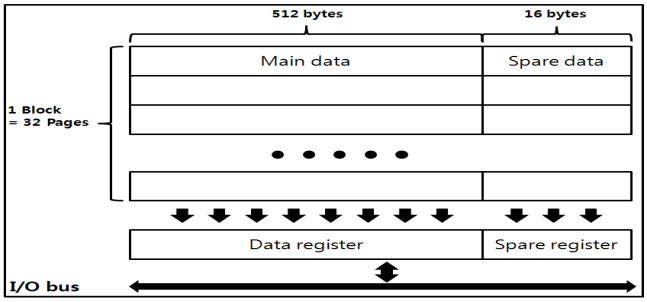
플래시 메모리의 각 블록들은 페이지로 구성 된다. 블록은 작은 블록에서 큰 블록으로 나눌 수 있다. 작은 블록인 경우 32개의 페이지로 구성되고, 하나의 페이지는 512

Byte이다. 그리고 큰 블록인 경우 64개의 페이지로 구성되며, 각 페이지는 2048 Byte의 크기를 가진다[2]. (그림 1)은 NAND 플래시 메모리 구조를 보여준다.

| 종류 | 읽기 | 쓰기 | 지우기 |
|---|---|---|---|
| DRAM | 60ns(2B)<br>2.56μs(512B) | 60ns<br>2.56μs(512B) | - |
| NOR<br>Flash | 150ns(1B)<br>14.4μs(512B) | 211μs(2B)<br>3.53ms(512B) | 1.2s<br>(128KB) |
| NAND<br>Flash | 10.2μs(1B)<br>35.9μs(512B) | 201μs(1B)<br>226μs(512B) | 2ms<br>(16KB) |
| Disk | 12.4ms(512B)<br>(Average) | 12.4ms(512B)<br>(Average) | - |

<표 1> 플래시 메모리의 기본 연산 수행 시간

NAND 플래시 메모리는 유효하지 않은 페이지들이 부족할 경우, 블록 안에 무효화 되어진 페이지들을 정리하여 자유 공간을 할당하는 가비지 컬렉션(Garbage-Collection) 작업이 이루어진다.



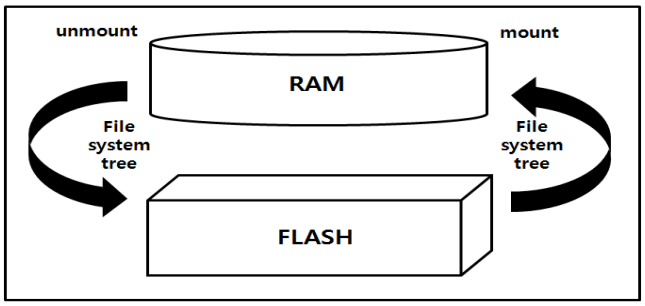

(그림 1) NAND 플래시 메모리 구조

## 2.3 File System

파일 시스템은 데이터를 쉽게 발견 및 접근할 수 있도록 관리하는 체제이다. 윈도우 계열 파일 시스템은 FAT16, FAT32, NTFS 등이 있고, 리눅스 계열 파일 시스템은 EXT4, JFFS2, YAFFS2, UBIFS 등이 있다. 파일 시스템에서 중요한 요소 중 하나는 저장 매체에 대한 안정성과 효율성을 고려해야 한다. YAFFS2와 UBIFS는 안드로이드 플랫폼 상에 탑재되고 있는 파일 시스템이다.

### 2.3.1 YAFFS2(Yet Another Flash File System 2)

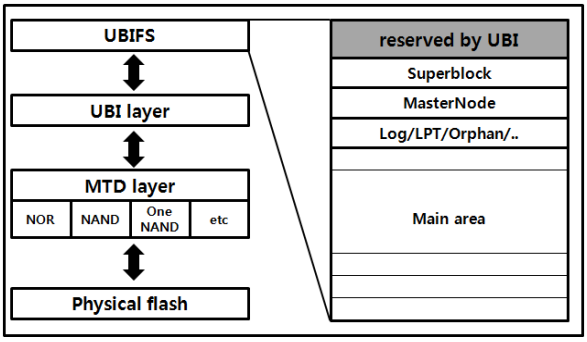
Alepha on 사에서 개발된 NAND 플래시 전용 파일 시스템인 YAFFS2는 업데이트 연산을 추가 연산으로 변형한 Out-Place-Update 기법을 사용한다. 이 기법은 플래시 메모리의 In-Place-Update가 되지 않는 문제점을 해결할 수 있다는 것이 장점이다[5]. YAFFS2는 체크 포인트의 기능을 도입하였고, YAFFS의 초기화 지연에 대한 문제점을 극복했다. 체크 포인트란 로그내의 정보를 반영하기 위하여 데이터 파일을 갱신하는 트랜잭션 로그의 일정한 시점이다. 체크 포인트 기능은 정상적인 mount 해제가 이루어지지 않을 시에는 체크 포인트는 무시되며, NAND 플래시 전체의 영역을 다시 검색하여 읽어오게 되어 초기화 시점에서 지연이 발생하게 된다. (그림 2)은 체크 포인트를 통한 YAFFS2 mount 시점 최적화를 보여준다.




(그림 2) 체크포인트를 통한 YAFFS2 mount시점 최적화

### 2.3.2 UBIFS(Unsorted Block Image File System)

UBIFS는 노키아에서 Szeged 대학과 함께 개발한 파일 시스템이다. UBIFS는 리눅스 커널 2.6.27 버전부터 지원하고 있으며, JFFS2(Journaling Flash File System2)의 후기 버전의 파일 시스템이다. (그림 3)는 UBIFS의 스택 구조를 보여준다[4].




(그림 3) UBIFS의 스택구조

UBIFS는 동적 압축, 전원 차단에 대한 안정성을 높이고, mount 속도 개선, 대용량 파일 처리, 쓰기 성능과 같은 효율성을 목표로 개발 되었다. UBIFS는 MTD(Memory Technology Device Layer)나 FTL(Flash Translation Layer)상에서 직접적으로 동작하지 못하고, UBI(Unsorted Block Image) 볼륨 상에서 동작되어지도록 설계가 되어져 있다. UBI는 리눅스 커널 2.6.22 버전부터 지원을 하고 있으며, 플래시 메모리 관리에 대한 bad block management, wear-leveling, 논리적 볼륨과 물리적 볼륨의 Mapping 작업, 볼륨의 정보저장 등의 기능을 수행한다[6]. UBI에 의한 물리적인 플래시 메모리를 관리해주기 때문에, UBIFS는 논리적 주소만을 가지고 동작이 가능하다.

## 3. 실험 환경 및 결과 분석

### 3.1 실험 환경

실험에 사용된 target board의 명세는 <표 2>와 같다. 성능 분석을 위해 사용되어진 툴은 iozone으로써 현재 일

반적으로 많이 쓰이는 파일시스템 벤치마킹 툴이다[7]. iozone은 read, write, re-read, re-write, read backwards, read strided, fread, fwrite 등의 다양한 테스트 환경을 제공해 준다. 이 실험에서는 가장 최신의 iozone 3.397 버전으로 성능 테스트를 하였으며, 오차를 최소화 하기위해 각 실험 당 10회 테스트가 수행되었다. 실험상의 환경은 <표 3>과 같다.

<표 2> target board 명세

| 구분 | 모델 |
|---|---|
| CPU | Samsung S5PV210 ARM Cortex A8 Processor |
| Memory | DDR2 512MB (16bit X 4 : 128MB) |
| Flash | 256 NAND Flash Memory(SLC) |
| Serial | RS232-2ch(1 port D-sub, 1 port connector) |
| SD/MMC | SD/MMC - 2ch |
| Ethernet | SMSC LAN 9220 100Mbps Ethernet Controller |

<표 3> 실험 환경

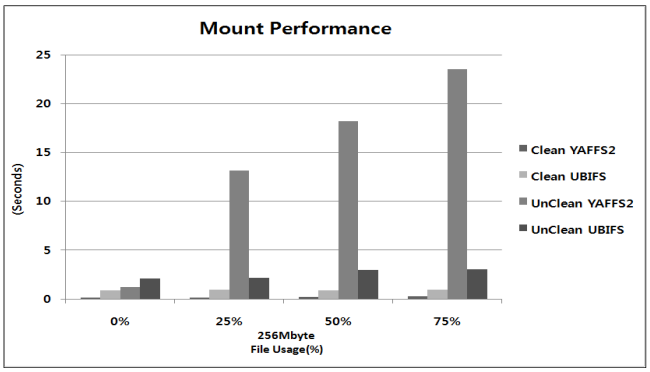
| 파일 시스템 | 실험 환경 | | |
|---|---|---|---|
| | mount 시간 | read/write 성능 | memory 소모량 |
| YAFFS2 | YAFFS2 mount 시간 | 1M,8M,16M, 32M | memfree+buffers+ cached |
| UBIFS | attaching UBI 시간 + UBIFS mount 시간 | 1M,8M,16M, 32M | memfree+buffers+ cached |

### 3.2 실험 결과 분석

본 절에서는 YAFFS2와 UBIFS에 대한 성능을 비교한다. 실험은 안드로이드 버전 2.3, 리눅스 커널 버전 2.6.35 상에서 실험을 하였다. 또한 256Mbyte의 NAND 플래시 메모리상에서 40Mbyte에 대한 가용메모리를 이용하여 유저 환경과 동일한 상태에서 실험을 진행하였다. 성능 측정 방식은 정상·비정상 종료에 따른 mount 시간 측정, read/write 대한 속도 측정을 하였다. 그리고 각 파일 시스템의 dummy 파일의 개수에 따른 메모리 소모량을 분석하여, 각 파일시스템의 성능 분석을 하였다.

#### 3.2.1 Mount 시간 분석

(그림 4)는 256Mbyte 메모리의 정상적인 종료(clean) 후 mount 시간과 비정상적인 종료(unclean) 후 mount 시간에 따른 분석을 보여준다. 비정상적인 mount 시간 측정은 메모리상에 1Mbyte를 write할 때, 임의적으로 target board의 전원을 off시켜 테스트 하였다. 또한 파일 사용량은 0%, 25%, 50%, 75%로 용량을 저장한 후 성능 분석을 하였다. 분석 결과 YAFFS2는 파일 사용량이 높아지고 비정상종료가 일어났을 때, 속도가 느리게 나타났다. 이는 YAFFS2의 체크 포인트가 비정상종료에 대한 대비를 하지 못하기 때문이다. 따라서 YAFFS2의 mount 시간은 모든 chunks를 검색하는 시간에 비례한다는 것을 알 수 있다. 그에 반해 UBIFS는 정상적인 mount 시간과 비정상적인 mount 시간의 차이가 평균 2.55초 매우 안정적임을 보여준다. 이는 UBIFS가 mount를 위한 전체적인 검색을 하지 않고, index에 미리 저장해둔 정보를 읽어오기 때문이다. 전체적인 파일 사용량에 따른 mount 속도 역시, 큰 변화가 없는 것으로 보아 다양한 파일 사용량 조건하에서도 UBIFS의 mount 성능에는 큰 영향을 주지 않는다고 할 수 있다.




(그림 4) YAFFS2와 UBIFS의 clean 과 unclean mount 시간 비교

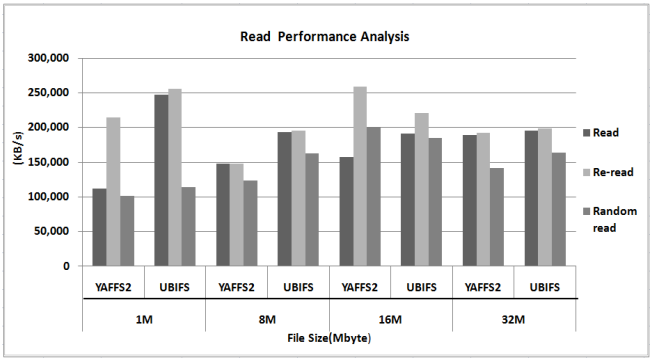
#### 3.3.2 read / write 성능 분석

(그림 5)는 YAFFS2와 UBIFS에 대한 read, re-read, random-read에 대한 3가지 유형에 따른 분석 결과이다.

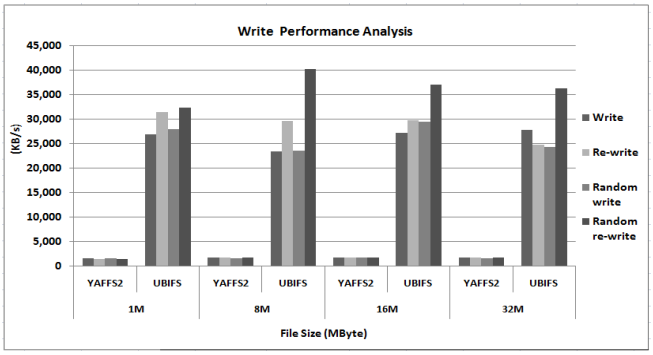
YAFFS2와 UBIFS의 전체적인 읽기 성능은 큰 차이점을 볼 수 없었다. 하지만 작은 용량을 read 할 때는 UBIFS가 빠른 속도를 보이지만, 16Mbyte에서는 YAFFS2의 re-read 성능은 점차적으로 빨라지는 것을 볼 수 있다. YAFFS2는 풀 덤프 방식(Full-Dump Method)을 사용하여 체크 포인트 정보를 메타데이터 저장하게 된다. 따라서 모든 정보를 저장하고 있는 메타데이터를 읽어 오게 되므로 re-read의 성능은 빨라지게 된다. 하지만 메타데이터를 한번에 저장해야 되므로 비교적 긴 시간 소모 생긴다는 것이 단점이다.

YAFFS2와 UBIFS의 write성능은 현격한 차이가 난다. (그림 6)은 YAFFS2와 UBIFS의 write, re-write, random-write, random re-write에 대한 4가지 유형에 대한 실험분석 결과이다. 그래프에서 보여주는 것과 같이 UBIFS의 write 성능이 압도적으로 빠르다는 것을 알 수 있다. UBIFS의 경우 다른 파일시스템과는 차별적으로 write-back cache를 지원하여 write 성능을 향상시켰다. write-back cache란 캐시 내에 일시적으로 정보가 저장된 후에 블록단위로 유휴 머신 주기(idle machine cycle) 동안에 저장 되는 방식이다. 즉, cache에 데이터를 저장했다가 해당 블록이 교체될 때에만 메모리에 저장되어지기 때문에, traffic이 적어지게 되므로 write성능이 향상되어진다. 그에 반해 YAFFS2의 체크 포인트 관리자는 주기에 따라 파일시스템 관리 정보를 플래시 메모리에 저장하게

된다. 쓰기 요청이 갑작스럽게 증가하고 일정 시간이 지속되면 체크포인트의 주기가 길어지게 되므로 성능의 효율성은 떨어지게 된다. 쓰기 요청이 모두 끝난 직후 체크포인팅이 이루어진다면 데이터 손실 없이 정보저장이 가능하다. 그러나 쓰기 요청에 대한 빈도가 갑작스럽게 늘어나게 되면, 데이터에 대한 무결성을 보장하지 못하는 단점이 있다. 이는 앞서 분석한 mount 속도와도 밀접한 관계가 있다.



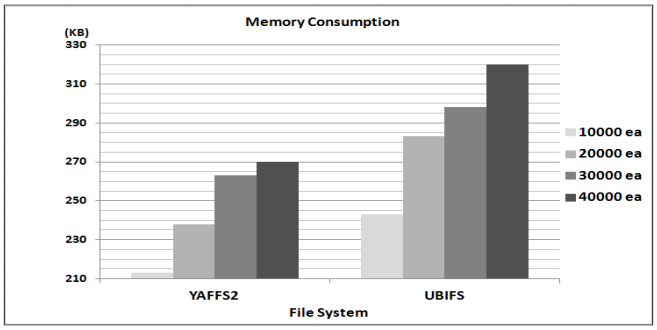

(그림 5) YAFFS2와 UBIFS의 read 성능 분석




(그림 6) YAFFS2와 UBIFS의 write 성능 분석

### 3.3.3 메모리 소모량 분석

그래프의 메모리 소모량에 대한 수치는 /proc/meminfo 정보에 있는 "memFree, buffers, cached"에 대한 합산 값이다. YAFFS2와 UBIFS의 메모리 소모량은 (그림 7)과 같다.




(그림 7) YAFFS2와 UBIFS의 메모리 소모량 분석

메모리 성능의 정확한 측정을 위해 1Kbyte dummy 파일의 수를 10,000 ~ 40,000개까지 증가시키면서 실험하였다. dummy 파일의 생성이유는 dummy 파일에 access할 때 임의적인 오버헤드를 주기 위함이다. 따라서 실 사용자들의 사용 환경과 유사한 실험 환경을 제공할 수 있다. 메모리 소모량의 성능 측정결과 YAFFS2보다 UBIFS의 메모리 소모량이 높은 것을 볼 수 있다. 이는 UBIFS가 write의 성능을 향상시키기 위해 write-back cache 정책을 따르기 때문이다. write-back cache 정책은 cache에 데이터를 써 두고 물리적 저장매체의 데이터와 캐시의 데이터를 동기화 시켜 블록의 교체 시 마다 메모리에 데이터를 할당하는 방식이다. 따라서 cache 영역의 일부를 데이터화하기 위해서는 메모리의 소모량이 YAFFS2보다 상대적으로 많아지게 된다.

## 4. 결론

본 논문에서는 안드로이드 플랫폼 상에서 NAND 전용 파일시스템인 YAFFS2와 UBIFS의 정상 종료 시 mount 속도와 비정상 종료 시 mount 속도, read/write의 성능 분석과 메모리 소모량에 대하여 분석하였다. 정상 종료 시 mount와 비정상 종료 시 mount에서는 UBIFS가 YAFFS2보다 약 49% 이상 향상되었다. 이는 UBIFS가 사용자 부주의나 예측 불가능한 상황 발생 시, 데이터의 무결성을 대하여 보장해주는 것을 의미한다. 두 파일상의 성능 분석에서 read성능은 큰 차이가 없었지만, write성능은 UBIFS가 압도적으로 향상 되었다는 것이 증명되었다. 이는 앞서 언급한 UBIFS의 write-back cache에 의하여 블록단위로 유휴 머신 주기 동안에 저장되기 때문이다. 또한, cache 영역을 access 하기 때문에 write 속도의 성능은 훨씬 향상되어지는 것이다. 그러나 write-back cache 과정에서 메모리 소모량이 증가한다는 단점이 있다. 실험결과 UBIFS가 YAFFS2에 비해 메모리 소모량은 약 26% 이상 높았다. 따라서 향후 UBIFS의 메모리 소모량의 축소화 방안에 대한 연구가 필요하다.

## 참고문헌

[1] J. Kim, J. Kim, S. Noh, S. Min, Y. Cho, "A Space-efficient Flash Translation Layer for CompactFlash System," IEEE Translations on Consumer Electrics, May 2002.

[2] 오용석, 박찬익, "안드로이드 기반 모바일 플랫폼을 위한 새로운 NAND 플래시 메모리 파일 시스템," 한국정보과학회, Vol 36. No. 2(B), pp. 388-393, 2009.

[3] 손익준, 김유미, 백승재, 최종무, "YAFFS 플래시 파일 시스템의 성능과 안정성 향상," 정보과학회 논문지, Vol.16, No.9, pp.898-903, 2010.

[4] A. Bityutskiy, F. Havasi, G. Loki, Z. Sogor. Design of UBIFS. "http://osl.sed..hu/~havasi/ubifs".

[5] YAFFS2, "http://www.yaffs.net".

[6] MTD(Memory Technology Devices), "http://www.linux-mtd.infradead.org".

[7] IOZONE, "http://www.iozone.org".