논문 2015-10-21

스마트 기기의 효율적인 I/O를 위한 플래시 파일 시스템 성능 분석

(Performance Analysis of Flash File System for the Efficient I/O on Smart Device)

정 경 호, 김 용 환, 김 상 진, 정 영 석, 김 성 수*
(Kyung-Ho Chung, Yong-Hwan Kim, Sang-Jin Kim, Young-Seok Jung, Sung-Soo Kim)

Abstract : Recently NAND flash memory has been found to be the primary cause of low performance in the smart device. NAND flash memory is different from each other the execution time of I/O operations that flash file system is required. Therefore, it is necessary to compare and analyze the flash file system I/O performance for the efficient I/O on smart device. In this paper, it was tested and analyzing the I/O performance of the YAFFS2, JFFS2, UBIFS. Experimental results most read I/O performance is good, but the writing I/O performance is not good. For UBIFS, showed a more good I/O performance compared to other flash file system.

Keywords : Flash file system, Performance, JFFS, YAFFS, UBIFS

Ⅰ. 서 론

최근 임베디드 시스템은 다양한 분야에서 활용하고 있으며, 특히 스마트폰, MP3 플레이어, 디지털 카메라 등과 같은 휴대용 스마트 기기로 크게 확산되고 있다. 스마트 기기는 기존 임베디드 시스템과는 달리 고성능 운영체제를 사용하면서, 기존의 고유한 목적 외에 센서, 네트워크, 멀티미디어 기능을 통한 다양한 형태의 콘텐츠들을 제공한다. 하지만 전원 공급, 부팅속도, 어플리케이션 종류에 따라 서로 다른 성능을 가지는 문제가 있다. 특히 다양한 어플리케이션 종류와 특징에 따라 I/O 성능 면에서 큰 차이가 나타나는데, 일반적으로 그 원인을 프로세서의 성능과 무선 네트워크의 연결 문제로 본다.

최근 스마트 기기에서 사용하는 NAND 플래시 메모리가 스마트 기기 어플리케이션에 심각한 성능 저하를 가져오는 주요 원인으로 밝혀졌다. NAND 플래시 메모리는 작고 가벼우면서 대용량의 데이터를 저장할 수 있으므로 운영체제나 데이터베이스와 같은 시스템 소프트웨어의 설치와 사용이 가능하다. 하지만 NAND 플래시 메모리는 특성상 I/O 연산 동작과 수행 시간이 다르므로 좀 더 복잡한 I/O 인터페이스를 처리하기 위한 플래시 파일 시스템이 요구된다. 따라서 스마트 기기의 효율적인 I/O를 위해서는 다양한 플래시 파일 시스템의 I/O 성능을 비교하고 분석할 필요가 있다.

본 논문에서는 스마트 기기의 I/O 연산과 성능의 관계를 살펴보고, 임베디드 리눅스 기반에서 주로 사용하는 JFFS, YAFFS, UBIFS 플래시 파일 시스템의 I/O 성능을 실험하여 비교 분석한다. 또한 이를 통해 I/O 동작이 스마트 기기의 어플리케이션 로딩과 실행에 어떠한 성능을 보이는지 비교 분석한다.

Ⅱ. 관련 연구

1. 스마트 기기의 I/O 성능

NAND 플래시 메모리를 사용하는 스마트 기기의 많은 어플리케이션들은 종류에 따라 I/O 성능 차이를 보인다. I/O 성능이란 사용자가 어플리케이션을 사용하는데 있어 웹 사이트가 로딩 되는 시간과 어플리케이션이 실행되는 시간으로 판단할 수

*Corresponding Author(ninny@ikw.ac.kr)
Received: 15 Nov. 2014, Revised: 18 Dec. 2014, Accepted: 26 Jan. 2015.
K.-H. Chung, Y.-H. Kim, S.-J. Kim, Y.-S. Jung, S.-S. Kim: Kyungwoon University

 2015 June 10(3) 171-178
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2015.10.3.171

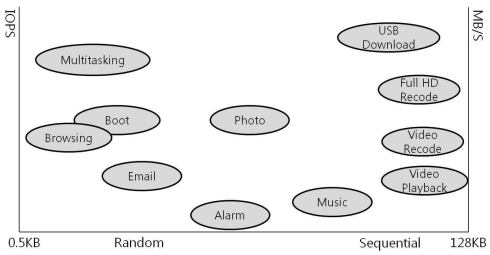

그림 1. 스마트 기기의 주요 어플리케이션 I/O
Fig. 1 Application I/O of Smart Device

있다. 미국 NEC 연구소에 따르면, 스마트 기기에서 사용되는 플래시 메모리가 스마트 기기의 어플리케이션에 심각한 성능저하를 가져오는 주요 원인으로 밝혀졌다 [1]. 이것은 플래시 메모리의 특성에서 찾을 수 있으며, 플래시 메모리의 I/O 연산 수행 시간이 서로 다르기 때문이라 할 수 있다. 예를 들어 게임이나 멀티미디어 어플리케이션의 경우 플래시 메모리의 I/O가 네트워크 송수신량보다 크다. 이미지나 동영상 뷰어와 같은 콘텐츠는 쓰기 연산 보다는 읽기 연산이 더 많이 발생하며, 디지털 카메라나 동영상 촬영과 같은 어플리케이션들은 쓰기 연산이 더 많이 발생한다 [2]. 그림 1은 스마트 기기의 각 어플리케이션을 순차 및 랜덤 I/O 접근에 따라 분류한 내용이다.

데이터베이스의 경우 연락처, 통화기록, 메일 목록, 웹 히스토리, 시스템 설정과 같은 수많은 어플리케이션에서 사용하며, NAND 플래시 메모리의 쓰기 연산이 자주 발생한다. 특히 순차 I/O 동작에 비해 랜덤 I/O 동작을 더 많이 수행한다. 플래시 파일 시스템에서 순차 I/O보다 랜덤 I/O가 더 많이 발생할 경우 I/O 성능은 더욱 낮아진다. 이것은 플래시 메모리의 특성상 데이터가 순차적으로 기록되기 때문에 랜덤 I/O의 연산 속도가 더욱 늦다. 따라서 랜덤 I/O 동작이 많이 발생하면 스마트 기기의 I/O 성능에 많은 영향을 미칠 수 있다 [3].

2. NAND 플래시 메모리

플래시 메모리는 스마트폰, 디지털 카메라 등과 같은 휴대용 단말기의 저장 공간으로 많이 사용된다. 플래시 메모리는 메모리 셀을 구성하는 게이트의 종류에 따라 NOR 플래시 메모리와 NAND 플래시 메모리로 구분하며, 저장 용도가 서로 다르다. NOR 플래시 메모리는 바이트 단위로 주소 지정을 수행하며, 집적도가 낮고 가격이 비싸다. 따라서 주로 실행 코드를 저장하는 용도로 사용한다. NAND

표 1. 주요 SD 카드별 I/O 성능
Table 1. I/O Performance of SD Cards

SD Card	Speed Class	I/O Performance(MB/s)			
		Seq Write	Seq Read	Rand Write	Rand Read
Transcend	2	4.35	13.52	1.38	2.92
RiData	2	5.86	11.51	0.03	2.76
Sandisk	4	4.93	8.44	0.67	0.73
Kingston	4	4.56	9.84	0.01	1.94
Wintec	6	9.91	13.38	0.01	3.82
A-Data	6	8.93	13.49	0.01	3.64
Patriot	10	8.83	13.38	0.01	3.72
PNY	10	10.28	14.02	0.01	3.95

플래시 메모리는 페이지 또는 블록 단위로 주소 지정이 이루어지며, NOR 플래시 메모리에 비해 I/O 성능이 떨어지지만 가격이 저렴하고 데이터 저장의 집적도가 높아서 대용량 데이터의 저장 매체로 많이 사용한다 [4]. 하지만 NAND 플래시 메모리는 다음과 같은 특성으로 인해 복잡한 I/O 연산을 처리하기 위한 전용 플래시 파일 시스템이 요구된다. 표 1은 주요 SD 카드별 I/O 성능을 나타낸다 [1].

첫째, 플래시 메모리의 읽기, 쓰기, 지우기 연산은 일반적인 블록 단위의 장치와는 달리 연산 시간이 각각 서로 다르다. 플래시 메모리는 특성상 페이지 단위로 분할된 데이터가 메모리 전역에 걸쳐서 분산 저장되며, 데이터 연산 단위인 페이지와 다수의 페이지가 모인 블록들로 구성된다. 따라서 읽기와 쓰기 연산은 페이지 단위로 수행하며, 지우기 연산은 페이지보다 큰 단위인 블록 단위로 수행한다. 따라서 읽기와 쓰기 연산에 비해 지우기 연산의 수행 시간이 길다.

둘째, 덮어쓰기 연산이 불가능하다. 플래시 메모리의 데이터를 수정하기 위해서는 반드시 해당 페이지가 포함된 블록에 대해 먼저 전기적으로 지우기 연산을 수행한 후 쓰기 연산을 수행해야 한다. 따라서 초기화 연산을 먼저 수행하고 저장 공간을 미리 확보하여, 삭제 대상이 아닌 데이터를 초기화된 공간에 복사한 후 원본 블록을 삭제해야 한다.

셋째, 각 블록의 지우기 횟수는 제한되어 있다. 배드 블록(Bad Block)은 더 이상 쓸 수 없게 된 블록을 말하며, 이러한 배드 블록이 증가할 경우 사용 가능한 공간이 감소하게 된다. 따라서 웨어 레벨링(Wear-Leveling)을 통해 최대한 배드 블록이 늦게 나타나도록 충분한 수명을 확보해야 한다.

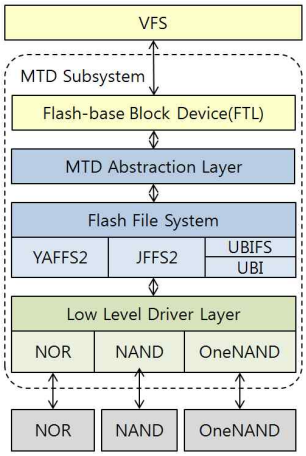

그림 2. MTD 기반의 플래시 파일 시스템
Fig. 2 Flash File System based MTD

3. 플래시 파일 시스템

플래시 파일 시스템은 구조에 따라 FTL(Flash Translation Layer) 기반의 플래시 파일 시스템과 MTD(Memory Technology Device) 기반의 플래시 파일 시스템으로 구분할 수 있다. FTL 디바이스는 SSD(Solid State Disk)나 eMMC(embedded Multi Media Card)와 같은 메모리 장치를 말한다. 이들은 내부에 플래시 메모리를 가지고 있지만, 외부에서는 블록 장치로 취급된다. MTD는 저 수준의 메모리 드라이버와 상위 플래시 파일 시스템 사이의 하위 서브시스템으로서 플래시 메모리에 대해 일관되고 공통된 소프트웨어 인터페이스를 제공한다. 안드로이드 스마트 기기는 리눅스 커널을 통해 MTD 서브시스템 인터페이스를 지원한다. 대표적인 MTD 기반의 플래시 파일 시스템으로는 JFFS, YAFFS, UBIFS 등이 있다. 그림 2는 MTD 기반의 플래시 파일 시스템 구성을 나타낸다.

3.1 FTL(Flash Translation Layer)

FTL 기반의 파일 시스템은 플래시 메모리 드라이버의 상단에 FTL라는 플래시 변환 계층을 두어 플래시 메모리의 논리적 주소와 물리적 주소를 따로 유지함으로서 같은 논리적 주소에 수정이 일어나는 문제를 해결한다. FTL은 웨어 레벨링(Wear-Leveling), 가비지 수집(Garbage Collection)을 포함한 플래시 장치 관리 기능도 제공하여 주기적으로 안정적인 자료를 새로운 블록에 이동시킬 수 있다. FTL에서 매핑 알고리즘은 입출력 요청의 패턴에 따라 서로 다른 수의 쓰기 및 지우기 연산을 발생시키는데, 여기서 결정하는 FTL 알고리즘과 버퍼 페이지 교체 기법에 따라 전체적인 I/O 성능을 향상시킬 수 있다. 하지만 파일 관련 연산에서 쓰기 연산을 복잡하게 하여, 기존 파일 시스템 계층과 FTL 계층에서 유사한 작업을 반복적으로 처리할 수 있기 때문에 효율성이 떨어진다 [5].

3.2 JFFS(Journaling Flash File System)

JFFS는 1999년 스웨덴의 Axis사에서 개발한 초기의 리눅스 플래시 파일 시스템으로 NAND 플래시 메모리 지원을 위해 JFFS2가 레드햇에서 설계되었다. JFFS는 플래시 메모리 공간을 순차적으로 저장하는 LFS(Log-Structured File System) 형태의 파일 시스템 구조를 가진다. LFS는 파일의 수정 연산이 발생했을 때, 새로운 페이지를 할당받아 수정된 데이터를 기록하고 기존의 페이지는 무효화시켜 매핑 정보를 갱신한다. 따라서 파일 시스템 장애 시에 로그 정보를 이용하여 빠른 복구와 온라인 상태에서 유지 보수 기능을 제공한다. 또한 가비지 수집을 통해 각 메모리 블록의 데이터 삭제 횟수를 균등하게 유지하고, 가용 메모리 용량을 효율적으로 관리한다. JFFS2는 웨어 레벨링과 가비지 수집을 통해 플래시 메모리에 대한 갱신 연산을 추가 연산으로 변형하여 처리하고, 플래시 메모리의 덮어쓰기가 허용되지 않는 문제를 해결하였다. 하지만 플래시 메모리의 이용률이 커지면서 데이터 삭제 횟수를 균등하게 관리하기 위해 각 블록의 데이터를 이동하기 위한 지우기 동작이 증가하여 쓰기 속도가 저하된다. 또한 마운트를 수행할 때 플래시 메모리를 스캔하고, 파일의 메타 데이터를 램에 유지하기 때문에 마운트 시간이 오래 걸리고 메인 메모리를 많이 사용하는 단점이 있다 [6].

3.3 YAFFS(Yet Another Flash File System)

YAFFS는 2002년 Aleph One사에서 개발한 NAND 플래시 메모리 파일 시스템으로 마운트 시점에서 빠른 성능을 가지며, JFFS2에 비해 입출력 속도가 뛰어나다. 따라서 안드로이드 플랫폼을 가진 다양한 스마트 기기에서 많이 사용한다. YAFFS는 마운트를 수행할 때 플래시 메모리 공간을 순차적으로 저장하므로 플래시 메모리 전체를 스캔해야 한다. 따라서 플래시 메모리가 클수록 마운팅 시간이 길어지며, 파일에 대한 수정이 발생하면 메모리 낭비를 가져온다. 또한 파일 쓰기 연산을 수행할 때 반드시 초기

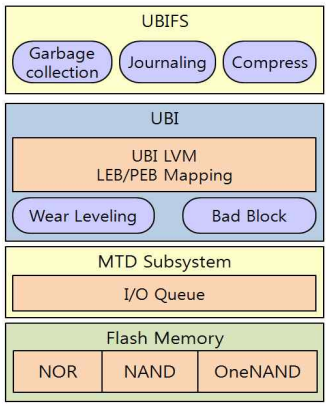

그림 3. UBIFS 플래시 파일 시스템의 구조
Fig. 3 Structure of UBIFS

화를 실행 한 후, 해당파일 전체를 블록 단위로 플래시 메모리에 다시 써야하므로 메모리 공간을 확보해야 한다. 이와 같이 마운트 연산을 수행할 때마다 발생하는 오버헤드를 줄이기 위해 스캔 과정을 생략한 YAFFS2가 소개되었다 [7].

3.4 UBIFS(Unsorted Block Image File System)

UBIFS는 2007년 노키아에서 Szeged 대학의 도움을 받아 개발한 차세대 플래시 파일 시스템이다. 주로 오픈 소스로 개발되어 리눅스 커널 2.6.27 이후에 탑재되어 노키아 및 구글의 스마트 기기에서 많이 사용한다. UBIFS는 하위의 UBI와 MTD 서브시스템 상에서 동작하며, UBI 서브시스템은 MTD에 의해 구성된 물리적 플래시 메모리 정보를 상위 계층의 논리적 메모리 정보와 매핑하여 관리한다. 다시 말해서 MTD는 가장 하위의 플래시 메모리 드라이버를 관리하고, UBI는 MTD로부터 다양한 플래시 메모리에 접근하기 위한 I/O 인터페이스를 제공받아 상위의 UBIFS에게 논리적인 블록을 제공한다. 따라서 UBIFS는 하위의 물리적인 메모리 관리에 신경 쓸 필요가 없으며, UBI 서브시스템이 배드 블록을 관리하고 웨어 레벨링을 통해 I/O 로드를 전체 플래시 메모리로 분산시켜주는 특징을 가진다. UBIFS는 메타 데이터가 플래시 메모리에 저장되고 유지되기 때문에 플래시 메모리의 스캔이 불필요하고 정전일 때만 저널 스캔이 발생한다. 이러한 요소들로 인해 마운트 시간을 빠르게 하고 쓰기 성능을 향상시켜 대용량 파일을 처리할 수 있다. 그림 3은 UBIFS의 구조를 나타낸다 [8].

Ⅲ. 플래시 파일 시스템의 성능 분석

1. 성능 분석의 필요성

현재 플래시 메모리의 I/O 성능을 향상시키기 위한 다양한 연구가 이루어져 왔다 [9]. 기존 연구들은 대부분 NAND 플래시 메모리의 I/O에 초점을 두고, 크게 플래시 메모리 자체 성능 연구와 플래시 전용 파일 시스템의 I/O 성능 연구로 나누어 수행하고 있다. 플래시 파일 시스템의 경우 복잡한 I/O 인터페이스 분석을 통한 효율적인 I/O 처리 방안을 제시하고 있다. 하지만 기존 연구들은 다양한 기능과 성능에도 불구하고 대용량 플래시 메모리와 빈번한 I/O에 대해 빠르게 지원하고 있지 못하다.

예를 들어 플래시 파일 시스템의 전체 성능은 어떤 버퍼 페이지 교체 기법을 사용하는지에 따라 달라질 수 있다. 버퍼는 어플리케이션 실행에서 빈번하게 사용되는 페이지를 메인 메모리에 보관하여 디스크 접근에 필요한 I/O 비용을 줄인다. 만약 버퍼에 여분의 저장 공간이 없다면, 기존 버퍼에 보관된 페이지 중 교체 대상 페이지를 선정해야 한다. 하지만 이러한 기법들은 기존 범용 리눅스 커널에 최적화되어 있으므로 플래시 메모리의 I/O 성능을 제대로 활용하지 못한다. 따라서 임베디드 리눅스 기반에서 기존 플래시 파일 시스템의 구조와 특성을 비교하고, 플래시 메모리의 특성과 버퍼 페이지의 교체 예상 비용을 고려한 효율적인 버퍼 페이지 교체 기법이 필요하다 [10].

본 논문에서는 YAFFS2, JFFS2, UBIFS와 같은 플래시 파일 시스템의 I/O 성능을 측정하고 분석한다. 최근 임베디드 리눅스 기반의 파일 시스템 성능 비교를 위한 연구가 진행되었다. 하지만 FAT, EXT와 같은 범용 파일 시스템의 성능을 비교하고 있으며, 이 분석은 NAND 플래시 메모리의 특성을 잘 고려하고 있지 못하다 [11]. 따라서 효율적인 I/O 성능 분석을 위해 최신의 UBIFS를 포함한 대표적 플래시 파일 시스템의 I/O 성능 분석이 필요하다. 또한 이러한 정보를 바탕으로 스마트 기기 어플리케이션 I/O 성능의 문제가 되는 요소들을 찾고, 어플리케이션 특성에 따른 적합한 플래시 파일 시스템을 선택할 수 있다.

2. 실험 환경 및 방법

2.1 실험 환경

본 논문의 실험에 사용된 임베디드 리눅스 시스템의 정보는 표 2와 같다.

표 2. 실험 대상 하드웨어 정보
Table 2. Hardware Specification

Spec	Description
OS	Embedded Linux Kernel 2.6.32.9
CPU	S5PV210 ARM Cortex A8 1GHz AP
Memory	Mobile DDR2 512Mbyte
Flash	SLC NAND Flash 256Mbyte
기타	7" Display, Audio, USB2.0, UART Ethernet Controller

실험에 사용하는 NAND 플래시 메모리의 파티션은 Bootloader, Kernel, Rootfs 3개로 나누고, UBIFS 파일 시스템에 사용할 Rootfs에는 mtd2에 할당하였다. 표 3은 NAND 플래시 메모리의 파티션에 할당한 정보를 나타낸다.

표 3. NAND 플래시 메모리 파티션
Table 3. Partition of NAND Flash Memory

Partition	Area	Size	Memory Address
mtd0	bootloader	0.5MB	00000000~00080000
mtd1	Kernel	3.5MB	00080001~00400000
mtd2	Rootfs	252MB	00400001~10000000

2.2 실험 방법

본 논문은 YAFFS2, JFFS2, UBIFS 플래시 파일 시스템 별로 읽기 쓰기 I/O로 분류하고, 표 4와 같이 페이지 크기는 4K부터 128K까지 6개로 나누어 페이지 크기에 따라 1M부터 64M까지 7개의 파일 크기로 나누어 실험하였다.

표 4. 실험 항목
Table 4. Experiment Data

Item	Experiment Data
Read I/O	Sequential Read, Random Read
Write I/O	Seq Write, Seq Retry Write, Rand Write
Page	4K, 8K, 16K, 32K, 64K, 128K
File Size	1M, 2M, 4M, 8M, 16M, 32M, 64M

실험은 웨어 레벨링과 가비지 수집 기능을 최소화하기 위해 매회 플래시 메모리 전체 지움을 실시한다. 메모리 사용량은 표 5와 같이 리눅스 커널을 모듈로 컴파일하여 파일 시스템을 마운트 할 때 사용되는 메모리 사용량을 마운트 전과 비교한다. I/O 성능은 sysbench-0.40.12 벤치마크 툴 [12]을 사용하여 측정하였으며, 표 6에서 메모리 사용량을 보여준다.

표 5. 플래시 파일 시스템별 모듈 크기
Table 5. Module Size by Flash File System

File System	Module	Module Size
YAFFS2	yaffs.ko	1,136K
JFFS2	jffs2.ko	1,928K
UBIFS+ UBI	ubifs.ko, ubi.ko	2,628K+ 1,252K

표 6. 플래시 파일 시스템별 메모리 사용량
Table 6. Memory Usage by Flash File System

File System	Using	Remaining	Cache
Before Mount	10,620K	299,120K	4,520K
YAFFS2	12,336K	297,404K	5,680K
JFFS2	13,516K	296,224K	6,472K
UBIFS	17,262K	292,448K	9,124K

3. 성능 분석

본 논문의 실험에 대한 성능 분석은 읽기, 쓰기 I/O를 순차와 랜덤으로 구분하여 결과를 분석한다.

3.1 순차 읽기 I/O 성능

순차 읽기에서는 YAFFS2, JFFS2, UBIFS 모두 파일 크기가 클수록 읽기 I/O가 증가하며, 특히 YAFFS2에서의 모든 크기의 파일이 JFFS2, UBIFS보다 높은 I/O 전송률을 가진다. 순차 읽기에 대한 I/O 평균값으로 볼 때 YAFFS2 파일 시스템이 675.818M로 가장 높은 I/O 성능으로 나타났으며, 다음으로 UBIFS, JFFS2 파일 시스템 순으로 나타났다. 그림 4는 플래시 파일 시스템의 순차 읽기 I/O 성능 결과이다.

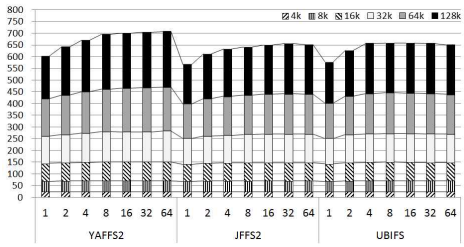

그림 4. 순차 읽기 I/O 성능
Fig. 4 Performance of Sequential Read I/O

3.2 랜덤 읽기 I/O 성능

랜덤 읽기에서는 YAFFS2와 UBIFS가 저용량 파일에서 높은 성능을 나타내며, 파일 크기가 커질수록 I/O 성능은 낮아진다. 이와는 달리 JFFS2는 저용량 파일인 경우 낮은 성능을 보여주지만, 파일 크기가 클수록 I/O 성능이 증가한다. I/O 평균값에서 YAFFS2는 674.636M로서

UBIFS보다 36.038M, JFFS2보다 76.854M의 더 높은 I/O 성능으로 나타났다. 그림 5는 플래시 파일 시스템의 랜덤 읽기 I/O 성능 결과이다.

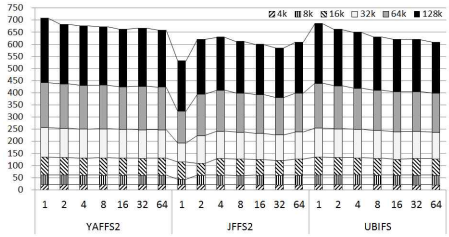

그림 5. 랜덤 읽기 I/O 성능
Fig. 5 Performance of Random Read I/O

3.3 순차 쓰기 I/O 성능

순차 쓰기에서는 UBIFS가 저용량 파일에서부터 대용량까지 아주 높은 I/O 성능을 나타냈다. 읽기 I/O에서 높은 성능을 보였던 YAFFS2는 의외로 쓰기 I/O에서 가장 낮은 I/O 성능을 가진다. 또한 JFFS2는 1M~64M까지 높고 낮은 전송률이 반복되어 다소 불안정한 I/O 성능으로 나타났다. I/O 평균값에서 UBIFS는 126.633M으로 JFFS2에 비해 두 배정도 빠르며 YAFFS2과는 무려 10배 이상의 빠른 I/O 성능을 나타냈다. 그림 6은 플래시 파일 시스템의 순차 쓰기 I/O 성능 결과이다.

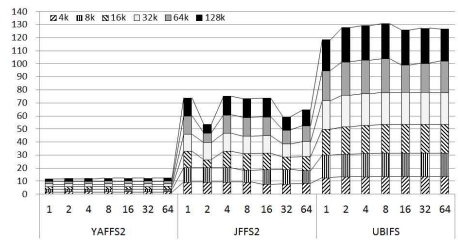

그림 6. 순차 쓰기 I/O 성능
Fig. 6 Performance of Sequential Write I/O

3.4 랜덤 쓰기 I/O 성능

랜덤 쓰기에서는 YAFFS2, JFFS2 모두 낮은 전송률을 보이지만 파일 크기에 따라 안정적으로 일정한 전송률을 나타냈다. UBIFS는 저용량일 때 아주 높은 I/O 성능을 보이며 대용량으로 갈수록 I/O 전송률이 낮아지는 것을 알 수 있었다. I/O 평균값에서는 UBIFS가 YAFFS2보다 17배 정도 높은 I/O 성능을 나타냈으며, JFFS2와는 3배 정도의 차이를 나타냈다. 그림 7은 플래시 파일 시스템의 랜덤 쓰기 I/O 성능 결과이다.

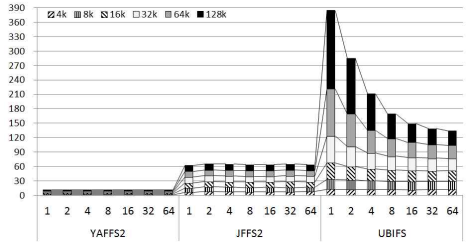

그림 7. 랜덤 쓰기 I/O 성능
Fig. 7 Performance of Random Write I/O

4. 성능 평가

4.1 플래시 파일 시스템별 전체 I/O 성능 평가

YAFFS2, JFFS2, UBIFS와 같은 플래시 파일 시스템에 대한 전체 성능에서 읽기 I/O에서는 순차 읽기, 랜덤 읽기 모두 YAFFS2가 높은 I/O 성능을 보였으며, 쓰기 I/O에서는 UBIFS가 상당히 높은 I/O 성능을 보였다. 표 7은 플래시 파일 시스템의 전체 I/O 평균을 나타낸다.

표 7. 파일 시스템의 전체 I/O 평균 (단위:MB)
Table 7. Average of Total File System I/O

File System	Seq Read	Rand Read	Seq Wrte	Rand Write	Average
YAFFS2	675.82	674.63	12.20	12.30	277.47
JFFS2	630.35	597.78	67.67	64.88	286.54
UBIFS	640.96	638.60	126.63	211.05	350.47

표 7에서와 같이 실험 대상의 I/O 전체 평균을 살펴보면 UBIFS가 350M로 가장 높았으며, 다음으로 JFFS2가 286M, 마지막으로 YFFS2가 277M로 가장 낮은 I/O 성능을 보였다. 전체적으로 3가지 플래시 파일 시스템 모두 읽기 I/O에서는 성능이 좋은 사실을 확인할 수 있다. 쓰기 I/O는 전반적으로 읽기 I/O에 비해 성능이 좋지 못하며, 특히 JFFS2에 비해 YAFFS2는 성능이 비교적 낮다. 하지만 UBIFS의 경우 다른 플래시 파일 시스템에 비해 전체적으로 좋은 I/O 성능을 나타냈다. 이러한 결과는 UBIFS의 특징에서 이유를 찾을 수 있다. JFFS의 경우 마운트를 수행할 때 플래시 메모리를 스캔하고, 파일의 메타 데이터를 램에 저장하므로 플래시

메모리 크기에 대비하여 마운트 시간이 선형적으로 증가하는 구조를 가진다. 이에 비해 UBIFS는 메타 데이터가 플래시 메모리에 저장되고 유지되기 때문에 플래시 메모리의 스캔이 불필요하다. 이러한 요소들로 인해 마운트 시간이 빠르다. 또한 UBI 서브시스템의 웨어 레벨링을 통해 I/O 로드를 전체 플래시 메모리로 분산시켜주므로 쓰기 성능이 향상된다.

4.2 어플리케이션에 따른 I/O 성능 평가

YAFFS2의 경우, 읽기 I/O에서 좀 더 나은 성능을 보여주므로 순차 읽기를 처리하는 멀티미디어 어플리케이션에서 적합하다고 할 수 있다. 하지만 대용량 파일에서 랜덤 읽기를 처리하는 멀티태스킹과 같은 작업에서는 오히려 JFFS2가 좋은 성능을 나타낼 수 있다. 안드로이드와 같은 스마트 기기에는 연락처, 통화기록, 메일 목록, 웹 히스토리, 시스템 설정 및 구글 지도 등의 어플리케이션에서 데이터베이스를 사용한다. 주로 파일 기반의 데이터베이스인 SQLite를 사용하며, 데이터를 저장하는데 수행 시간의 50% 이상을 사용하므로 데이터베이스 의존도가 매우 크다 [3]. SQLite는 어플리케이션의 정보를 파일에 저장하고 있으므로 NAND 플래시 메모리의 쓰기 연산이 빈번히 일어나며, 자체적으로 저널링을 쓰기 때문에 순차 I/O 동작보다는 랜덤 I/O 동작을 더 많이 수행한다. 따라서 웹 브라우징이나 데이터베이스와 같은 랜덤 I/O를 처리하는 어플리케이션에서는 UBIFS가 안정적인 I/O를 기대할 수 있다. 하지만 UBIFS는 저용량일 때 높은 I/O 성능을 보이지만 대용량 데이터 처리에서는 I/O의 성능은 낮아질 수 있다.

Ⅳ. 결 론

최근 스마트 기기의 일반적인 성능 저하 원인으로 프로세서의 성능과 무선네트워크의 연결 문제로 보고 있다. 하지만, NAND 플래시 메모리의 빈번한 I/O가 성능 저하의 더 큰 원인으로 분석되고 있다. 따라서 스마트 기기의 효율적인 I/O를 위해서는 다양한 플래시 파일 시스템의 I/O 성능을 비교하고 분석할 필요가 있다.

본 논문에서는 YAFFS2, JFFS2, UBIFS를 대상으로 순차 I/O와 랜덤 I/O로 구분하여 읽기, 쓰기의 성능을 실험하고 성능을 분석하였다. 특히 UBIFS의 경우 다른 플래시 파일 시스템에 비해 전체적으로 좋은 I/O 성능을 나타냈으나 대용량 데이터를 처리할 때 I/O의 성능이 낮아질 수 있다. 따라서 스마트 기기의 효율적인 I/O 향상을 위해서는 NAND 플래시 메모리 특성을 이해하고, 리눅스 기반의 커널 분석을 통해 I/O 성능을 향상시킬 수 있는 다양한 접근 방법이 필요하다. 이를 바탕으로 어플리케이션 I/O 성능의 문제가 되는 요소들을 찾고, 어플리케이션 특성에 따른 적합한 플래시 파일 시스템을 선택할 수 있어야 한다.

본 논문의 플래시 파일 시스템의 I/O 성능 실험은 임베디드 리눅스 커널에 대한 분석 능력이 많이 요구되며, 긴 시간을 통해 꾸준히 실험을 수행해야만 결과를 얻을 수 있다. 하지만 빠른 하드웨어의 발전에 따라 대상과 실험 환경이 계속 달라지므로 꾸준한 연구가 더욱 필요하다. 또한 실험 결과와 분석을 바탕으로 좀 더 구체적인 스마트 기기에서 어플리케이션을 수행하는 실험과 연구가 필요하다.

References

[1] H. Kim, N. Agrawal, C. Ungureanu, "Revisiting storage for smartphones," Journal ACM Transactions on Storage, Vol. 8, No. 4, pp. 1-14, 2012.

[2] S. Lee, J. Park, H. Lee, M. Kim, "A study on Smart-phone traffic analysis," Proceedings of 13th Network Operations and Management Symposium, pp. 1-7, 2011.

[3] K. Lee, Y. Won "Smart layers and dumb result: IO characterization of an android-based smartphone," Proceedings of the 10th ACM International Conference on Embedded Software, pp. 23-32, 2012.

[4] S. Kim, "Overview of Flash File Systems," KISS J. Communications, Vol. 25, No. 6, pp. 11-17, 2007 (in Korean).

[5] T. Chung, D. Park, S. Park, D. Lee, S. Lee, H. Song, "A survey of Flash Translation Layer," Journal of Systems Architecture - Embedded Systems Design, Vol. 55, No. 5, pp. 332-343, 2009.

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

[7] http://www.yaffs.net/yaffs-overview
[8] http://www.linux-mtd.infradead.org/doc/ubifs.html
[9] J. Ryu, C. Park, "A Technique to Enhance Performance of Log-based Flash Memory File Systems," IEMEK J. Embed. Sys. Appl., Vol. 2, No 3, pp. 184-193, 2007 (in Korean).
[10] J. Park, D. Park, "An Efficient Buffer Page Replacement Strategy for System Software on Flash Memory," KISS J. Database., Vol. 34, No. 2, pp. 133-140, 2007 (in Korean).
[11] J. Choi, "Performance Comparative Analysis of Flash File System for Embedded Systems on Linux Environment," KIICE J., Vol. 18, No 1, pp.109-114, 2014 (in Korean).
[12] http://sysbench.sourceforge.net/

Kyung-Ho Chung (정 경 호)

He received Ph.D. degree in Computer Engineering from Kyungpook National University, Korea, in 2011. He is professor in the department of Computer Engineering in Kyungwoon University, Korea. His research Interests Embedded System, RFID, Mobile Computing.
Email: mccart@ikw.ac.kr

Yong-Hwan Kim (김 용 환)

He received Ph.D. degree in Computer Engineering from Kyungpook National University, Korea, in 2013. He is professor in the department of Computer Engineering in Kyungwoon University, Korea. His research Interests Embedded System, RFID, Sensor.
Email: hypnus@ikw.ac.kr

Sang-Jin Kim (김 상 진)

He received Ph.D. degree in Computer Engineering from Kyungpook National University, Korea, in 2000. He is professor in the department of Computer Engineering in Kyungwoon University, Korea. His research Interests Algorithm, Game Algorithm.
Email: sjkim@ikw.ac.kr

Young-Seok Jung (정 영 석)

He received Ph.D. degree in Computer Engineering from Yeungnam University, Korea, in 2003. He is professor in the department of Computer Engineering in Kyungwoon University, Korea. His research Interests Computer Network, Mobile Computing.
Email: ysjung@ikw.ac.kr

Sung-Soo Kim (김 성 수)

He received Ph.D. degree in Computer Engineering from Kyungpook National University, Korea, in 2012. He is professor in the department of Mobile Engineering in Kyungwoon University, Korea. His research Interests Embedded System, RFID, Sensor.
Email: ninny@ikw.ac.kr