J Korea Industr Inf Syst Res Volume 18 Number 4
http://dx.doi.org/10.9723/jksiis.2013.18.4.009

UBIFS 메모리 할당에 관한 I/O 성능 분석†

(I/O Performance Analysis about Memory Allocation of the UBIFS)

이 재 강*, 오 세 진**, 정 경 호***, 윤 태 진****, 안 광 선*****
(Jaekang Lee, Sejin Oh, Kyungho Chung, Taejin Yun, and Kwangseon Ahn)

요 약 플래시 메모리는 비휘발성, 저전력, 빠른 입출력, 충격에 강함 등과 같은 많은 장점으로 스마트 기기 및 임베디드 시스템의 저장매체로 많이 사용되고 있다. 낸드(NAND) 플래시에 사용되는 파일시스템(File System)은 대표적으로 YAFFS2, JFFS2, UBIFS 등이 있다. 본 논문에서는 최근 리눅스 커널에 포함된 UBIFS 파일시스템에 메모리 할당을 달리하여 I/O 성능을 실험한다. 제안한 I/O 성능 분석은 순차접근 방법과 랜덤접근으로 분류하고, 메모리 할당은 kmalloc(), vmalloc(), kmem_cache()를 사용하여 6가지 유형으로 나누어 실험하였다. 실험을 통하여 6가지 유형 중 UBI 서브시스템과 UBIFS에 vmalloc()과 kmalloc()을 적용한 2번째 유형이 순차읽기 12.45%, 순차다시쓰기 11.23%의 빠른 성능을 보였으며 랜덤 읽기에는 7.82% 랜덤 쓰기에서는 6.90%의 성능 향상을 보였다.

핵심주제어 : UBIFS, 낸드 플래시, 파일시스템, 메모리 할당

Abstract Flash memory is mostly used on smart devices and embedded systems because of its nonvolatile memory, low power, quick I/O, resistant shock, and other benefits. Generally the typical file systems base on the NAND flash memory are YAFFS2, JFFS2, UBIFS, and etc. In this paper, we had variously made an experiment regarding I/O performance using our schemes and the UBIFS of the latest Linux Kernel. The proposed I/O performance analyses were classified as a sequential access and a random access. Our experiment consists of 6 cases using kmalloc(), vmalloc(), and kmem_cache(). As a result of our experiment analyses, the sequential reading and the sequential rewriting increased by 12%, 11% when the Case 2 has applied vmalloc() and kmalloc() to the UBI subsystem and the UBIFS. Also, the performance improved more by 7.82%, 6.90% than the Case 1 at the random read and the random write.

Key Words : UBIFS, NAND Flash, FileSystem, Memory Allocation

1. 서 론

최근 스마트 기기의 빠른 보급으로 다양한 애플리케이션을 개발하고 있다. 이러한 스마트기기의 성능 저하 원인으로 프로세서의 성능과 무선네트워크의 연결 문제로 보고 있다. 하지만, 최근 스마트기기에 사용하는 낸드 플래시 메모리의 빈번한 I/O가 성능저하의 더 큰 원인으로 분석하고 있다. 특히 순차적인 I/O을 수행하는 멀티미디어 애플리케이션에 비해 랜덤으로 I/O하는 데이터베이스 애플리케이션에서 성능저하가 나타난다[1]. 또한 낸드 플래시 파일시스템의 I/O

* 경북대학교 IT대학 컴퓨터학부, 제1저자
** 경북대학교 IT대학 컴퓨터학부, 제2저자
*** 경북대학교 IT대학 컴퓨터학부, 제3저자
**** 경운대학교 IT에너지대학 모바일공학과, 제4저자
***** 경북대학교 IT대학 컴퓨터학부 (e-mail:gsahn@knu.ac.kr)

스케줄러 성능분석 등 I/O성능 향상을 위한 연구가 많이 이루어지고 있다[2][13][14]. 따라서 낸드 플래시 메모리의 I/O 향상을 위한 연구가 필요하다.

본 논문은 스마트기기에 사용하는 낸드 플래시 메모리의 I/O 성능을 UBI(Unsorted Block Images) 서브시스템과 UBIFS(Unsorted Block Images File System) 파일시스템에서 I/O 성능을 분석한다. UBI 서브시스템은 플래시 메모리를 위한 논리볼륨을 관리하는 시스템이고, UBIFS는 UBI 서브시스템 상에서 동작하는 파일시스템이다. 성능 분석은 UBI 서브시스템의 2개의 버퍼와 UBIFS 파일시스템의 4개의 버퍼에 메모리 할당기법을 달리하여 순차 I/O 방법과 랜덤 I/O로 분류한다. 그리고 메모리 할당은 kmalloc(), vmalloc(), kmem_cache()를 사용하여 6가지 유형으로 나누어 비교분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대해 살펴보고 3장에서는 UBIFS 메모리 할당에 관한 I/O 실험방법을 제안한다. 4장에서는 I/O성능을 비교분석하고 마지막으로 결론을 맺는다.

2. 관련연구

본 장에서는 스마트기기에 사용되는 낸드 플래시 메모리의 성능 저하의 원인을 살펴보고 낸드 플래시 메모리의 파일시스템과 메모리 할당방식에 대해서 기술한다.

2.1 낸드 플래시 메모리 성능

낸드 플래시 메모리는 스마트 기기의 저장매체로 많이 사용되며 정보를 유지함에 있어 전력이 필요 없는 비휘발성 특징을 가진다. 또한 외부의 충격에 강하며 배터리로 동작하는 스마트기기의 저장 장치로 많이 사용된다[3]. 하지만, 낸드 플래시 메모리는 블록 내에서 특정 단위로 읽고 쓸 수 있지만, 블록 단위로 지워야 하는 단점과 지우기 횟수가 제한되어 있다.

일반적으로 스마트 기기의 성능저하 원인을 프로세서의 성능과 무선네트워크의 연결 문제로 보고 있다. 하지만 낸드 플래시 메모리의 빈번한 I/O가 성능 저하의 더 큰 원인이 되고 있다[1]. 예를 들어 스마트기기에서 많은 애플리케이션이 데이터를 랜덤으로 기록하도록 만들어져 있어 플래시 메모리의 성능은 더욱 떨어진다. <표 1>은 스마트기기에서 사용하는 주요 애플리케이션의 디스크 I/O와 네트워크의 송수신량을 보여준다.

<Table 1> 스마트기기의 주요 애플리케이션의 디스크 I/O와 네트워크 송수신량

앱	디스크(MB)		네트워크(MB)	
	읽기	쓰기	수신	송신
AngryBird	20.69	0.04	4.09	4.44
Gallery	1.88	0.00	0.00	0.00
Twitter	4.62	0.06	5.62	1.61

애플리케이션은 크게 게임, 멀티미디어 서비스, 소셜 네트워크 서비스로 분류할 수 있다. 게임과 멀티미디어 서비스는 디스크 I/O가 네트워크의 송수신량보다 훨씬 크다. 소셜네트워크 서비스 경우 네트워크 송수신량이 디스크 I/O보다 다소 크지만 디스크 I/O도 비교적 크다. 따라서 낸드 플래시 메모리의 데이터 I/O 성능은 스마트 기기의 성능 향상의 중요한 요소라 할 수 있다. <표 2>는 낸드 플래시를 사용하는 주요 SD 카드의 I/O 성능을 보여준다.

<Table 2> 주요 SD Card I/O 성능

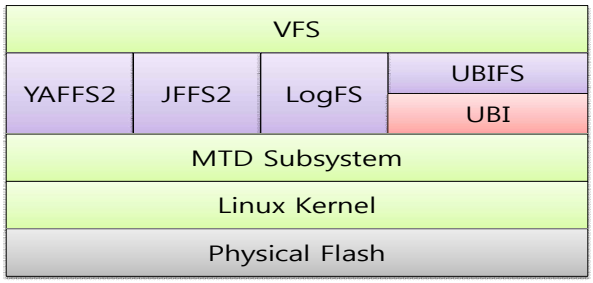
SD 카드 (16 GB)	순차(MB/s)		랜덤(MB/s)	
	쓰기	읽기	쓰기	읽기
SanDisk	4.93	8.44	0.67	0.73
Kingston	4.56	9.84	0.01	1.94

SD 카드의 I/O 성능은 순차 I/O 보다 랜덤 I/O의 데이터 전송량이 현저히 낮은 것을 볼 수 있다. 따라서 랜덤 I/O 성능은 낸드 플래시 메모리의 성능과 밀접한 관계가 있다. 이와 같이 낸드 플래시 메모리의 I/O 성능 향상을 위해서는 낸드 플래시 메모리를 지원하는 파일시스템의 분석과 연구가 필요하다.

2.2 낸드 플래시 메모리의 파일시스템

낸드 플래시 메모리를 지원하는 파일시스템은 MTD(Memory Technology Device) 서브시스템을 기반으로 한다. MTD는 다양한 플래시 메모리에 대한 추상계층을 지원하는 인터페이스로서 서로 다른 유형에서도 동일한 API를 사용가능하도록 해준다[3]. 대표

적인 낸드 플래시 메모리 전용 파일시스템은 YAFFS2(Yet Another Flash File System 2)[5], JFFS2(Journaling Flash File System 2)[6], UBIFS(Unsorted Block Image File System)[7] 등이 있다. UBIFS는 JFFS2 파일시스템이 가지고 있는 동적 압축, 전원 차단에 대한 내성 및 복구, 마운트 성능 개선, 대용량 파일 처리, 쓰기 성능을 개선한 파일시스템이다. <그림 1>은 MTD 기반 플래시 파일 시스템 구성도 이다.



<Fig 1> MTD기반 플래시 파일 시스템 구성도

2.2.1 UBI(Unsorted Block Images) 서브시스템

UBI 서브시스템은 <그림 1>과 같이 MTD 서브시스템 상에서 LEB(Logical Erase Block)로 볼륨을 구성하고, 플래시 메모리를 위한 논리볼륨을 관리하는 시스템이다. UBI 서브시스템의 기능은 크게 네 가지로 볼 수 있다. 첫째, 플래시 메모리의 수명이 짧아지는 것에 대한 마모평준화 기능이다. 둘째, 사용자가 배드 블록에 대한 고려 없이 LEB를 그대로 사용하도록 배드 블록을 관리해주는 기능. 셋째, LEB 와 PEB(Physical Erase Block)의 매핑 기능이다. 마지막으로 볼륨의 생성, 제거, 크기 변경을 동적으로 수행하는 기능이다[8]. <표 3>은 UBI 서브시스템에 사용하는 버퍼이다.

<Table 3> UBI 서브시스템에 사용하는 버퍼

버퍼	내용	메모리 할당
peb_buf1	PEB Management	vmalloc()
peb_buf2	PEB Management	

UBI 서브시스템의 메모리할당은 peb_buf1, peb_buf2 두개의 버퍼에 vmalloc()를 통해 메모리 할당을 한다. peb_buf1과 peb_buf2 두개의 버퍼는 PEB 관리를 위한 버퍼로 사용되며, UBI 서브시스템이 커널에 적재될 때 할당된다.

2.2.2 UBIFS 서브시스템

UBIFS는 JFFS2보다 발전한 파일시스템으로서 다른 파일시스템과는 달리 UBI 서브시스템 상에서 동작한다[6]. UBIFS의 특징은 플래시 메모리의 크기에 비례하여, 마운트 시간이 빠르고 비정상으로 종료되었을 때 저널링을 통한 완벽한 복구가 가능하다. 또한 Write-back 지원으로 빠른 I/O성능을 가지고 있다. <표 4>는 UBIFS 파일시스템에 사용하는 4가지 버퍼를 나타낸다.

<Table 4> UBIFS 파일시스템에 사용하는 버퍼

버퍼	내용	메모리 할당
ileb_buf	index LEB	vmalloc()
lpt_buf	LEB Properties tree	
sbuf	Scanning, GC, Reply	
orphan	orphan node, commit	

GC : Garbage Collection

UBIFS 파일시스템에 사용하는 4개의 버퍼는 LEB와 GC 관리를 위한 버퍼이며, 모두 vmalloc()를 통해 메모리 할당을 한다.

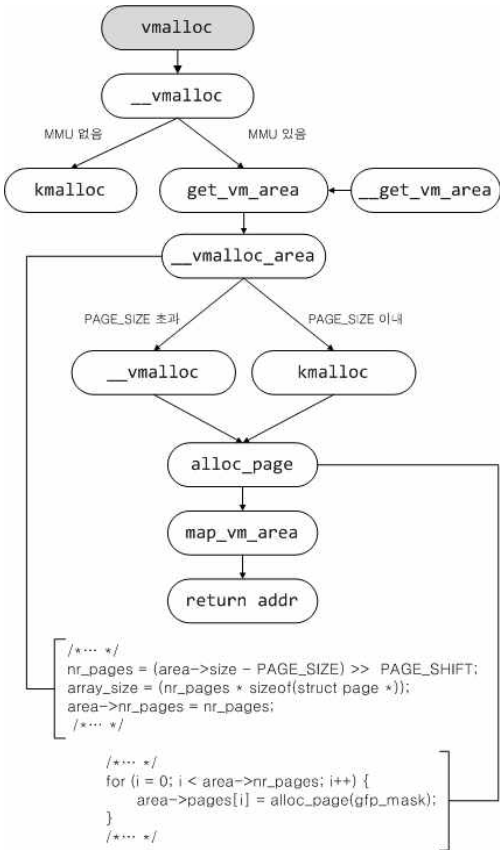
2.3 메모리 할당 방식

최근 많이 보급된 안드로이드 스마트기기는 리눅스 운영체제 기반이며, 낸드 플래시 메모리를 지원한다. 이러한 낸드 플래시 메모리 상의 파일시스템은 다양한 메모리 할당방식을 사용한다. 리눅스 커널은 메모리 할당을 위해 대표적으로 vmalloc()과 kmalloc() 두 가지 함수를 제공한다. 본 절에서는 낸드 플래시 메모리에 사용하는 파일시스템 메모리 할당방식에 대해서 살펴본다.

2.3.1 vmalloc()

vmalloc() 함수는 커널 내의 연속된 가상 메모리 공간을 할당하며, 비연속적으로 존재하는 페이지들을 가지고 새로운 페이지 테이블을 생성하여 메모리를 관

리한다. vmalloc()은 가상 메모리와 물리 주소 간의 변환을 담당하는 MMU (Memory Management Unit) 유무에 따라서 함수 호출이 달라진다[9]. <그림 2>는 vmalloc() 호출 절차를 보여준다.

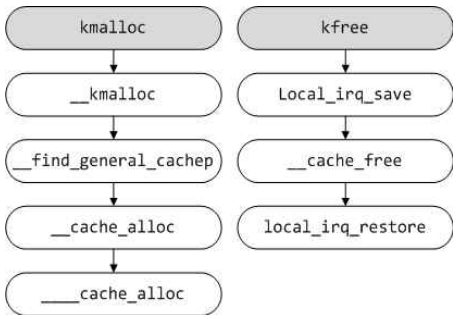


<Fig 2> vmalloc() 호출 절차

vmalloc() 호출 절차는 MMU의 유무에 따라 달라진다. MMU가 없으면 vmalloc()은 kmalloc()을 호출한다. MMU가 있으면 get_vm_area()를 통해 가상메모리 공간을 할당하며, 메모리 공간의 페이지 수에 따라 페이지 크기가 초과할 경우 alloc_pages() 함수를 호출하여 페이지 단위로 메모리 공간을 할당한다. 따라서 메모리 공간은 물리메모리에 비연속적으로 할당되어 있으며, 가상메모리에서 연속적으로 메모리를 할당하기 위해 map_vm_ares() 함수를 호출한다. 결과적으로 vmalloc()은 비연속적인 물리메모리를 가상의 연속적인 메모리로 사용하기 위하여 별도의 페이지 테이블을 관리한다. 따라서 오버헤드가 발생하므로 kmalloc()보다 느리다.

2.3.2 kmalloc()

kmalloc() 함수는 커널 영역에서 연속된 물리 메모리를 바이트 단위로 할당하기 위해 사용하며, 메모리를 최대한 효율적으로 할당하기 위해 커널은 일반 캐시를 사용한다[9]. <그림 3>은 kmalloc()과 kfree()의 호출 절차를 보여준다.

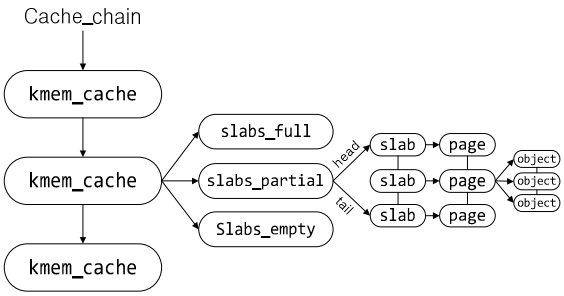


<Fig 3> kmalloc(), kfree() 호출 절차

kmalloc()은 연속적으로 메모리를 할당하므로 vmalloc()에 비해 별도의 페이지 테이블을 관리하지 않는다. 또한 호출과정에서 내부 함수를 통하여 일반 캐시를 찾고, 해당 캐시에 메모리를 할당하므로 구조가 단순하고 메모리 할당 속도가 빠르다.

2.3.3 슬랩 할당자(Slab Allocator)

kmalloc()은 메모리를 할당하고 해제할 때 내부적으로 슬랩 할당자에서 처리하며, 슬랩 할당자는 메모리의 내부 단편화를 해결하기 위해 사용한다. 커널은 내부적으로 같은 크기의 객체를 할당하여 메모리를 관리하는데, 슬랩 할당자는 이처럼 빈번하게 할당되는 객체를 미리 만들어 두고 필요할 때 할당하고 객체를 사용하지 않으면 할당을 해제한다. 따라서 객체를 생성할 때 드는 비용과 제거할 때 비용을 최적화하고, 페이지 내의 메모리를 효율적으로 관리한다[10, 11]. <그림 4>는 슬랩 할당자를 구성하는 주요 구조이다.



<Fig 4> 슬랩 할당자를 구성하는 주요 구조

슬랩 할당자는 kmem_cache라 불리는 여러 캐시를 연결한 cache_chain으로 구성된다. 슬랩은 객체를 할당하는 최소단위로서 슬랩 할당자에 의해 여러 개의 객체가 할당되고 해제된다. 각 캐시는 슬랩 목록을 포함하는 연속적인 메모리 블록으로서 크기가 정해진 객체를 담아서 정의 한다. 슬랩 목록은 슬랩이 완전히 할당된 slabs_full과 일부가 할당된 slabs_partial, 비어있거나 객체를 할당하지 않은 slabs_empty로 세 가지로 구성된다. 커널이 관리하는 페이지 단위는 커널이 사용하기에 다소 큰 단위이다. 따라서 슬랩은 커널 내에서 흔히 일어나는 동적 메모리 할당의 오버헤드를 줄이기 위한 캐싱 역할을 하고, 내부 단편화 문제를 해결하여 성능 개선을 가져온다.

3. 메모리 할당을 통한 I/O 성능 분석

UBI와 UBIFS 파일시스템은 시스템 별로 서로 다른 버퍼를 사용하며, 메모리 할당 또한 각각 수행한다. 본 장에서는 앞 장에서 서술한 UBI 서브시스템과 UBIFS 파일시스템의 I/O 성능을 분석하기 위해 메모리 할당방법을 사용하여 I/O 성능을 비교 분석한다. 메모리 할당은 kmalloc()과 vmalloc() 그리고 슬랩 할당자를 이용한 kmem_cache()를 각각 사용한다.

3.1 I/O 성능 분석 기준

본 논문에서는 메모리 할당방법을 kmalloc()과 vmalloc() 차례로 적용하였으며, 6가지 Case로 분류하여 I/O 성능 분석 방법을 제안한다. <표 5>는 UBI 서브시스템과 UBIFS 파일시스템에 메모리 할당방법을 달리하여 성능분석을 위한 6가지 Case를 보여준다.

<Table 5> UBIFS I/O 성능분석을 위한 분류표

시스템 / 유형	UBI 서브시스템	UBIFS 파일시스템
Case 1	vmalloc()	vmalloc()
Case 2	vmalloc()	kmalloc()
Case 3	kmalloc()	vmalloc()
Case 4	kmalloc()	kmalloc()
Case 5	kmem_cache()	vmalloc()
Case 6	kmem_cache()	kmalloc()

Case 1은 UBI 서브시스템과 UBIFS 파일시스템의 기본적인 메모리 할당방식으로 모두 vmalloc()를 사용한다. Case 2와 Case 3은 UBI 서브시스템과 UBIFS 파일시스템에 서로 다른 메모리 할당방식을 각각 적용하며 Case 4는 모두 kmalloc()를 적용하였다. Case 5와 Case 6은 슬랩 할당자에서 사용하는 kmem_cache() 함수를 UBI 서브시스템에 적용하고 UBIFS 파일시스템에는 각각 vmalloc()와 kmalloc()를 적용하였다. 이와 같이 UBIFS와 UBI 서브시스템의 메모리 할당방식을 서로 다르게 하여 파일시스템의 I/O성능을 비교한다.

3.2 I/O 성능 분석 방법

본 논문에서는 UBIFS 파일시스템의 I/O 성능 분석을 위하여 메모리 할당별로 크게 순차 I/O 방식과 랜덤 I/O 방식으로 구분한다. 낸드 플래시 메모리의 순차 I/O와 랜덤 I/O의 데이터 전송량은 메모리 성능을 분석하는 중요한 요소이다. 성능분석을 위해서 순차 I/O 방식은 읽기, 쓰기, 다시쓰기의 3가지 동작으로 구분하고 랜덤 I/O는 랜덤읽기, 랜덤쓰기의 2가지 동작으로 구분하여 실험 한다. <표 6>은 6가지 Case별로 각각 순차 및 랜덤 I/O의 성능을 분석하기 위한 기준을 나타낸다.

<Table 6> 실험 기준

시스템 / 유형	UBI 서브시스템	UBIFS 파일시스템
메모리할당 방식	kmalloc() vmalloc() kmem_cache()	kmalloc() vmalloc()
순차 I/O	순차 읽기, 순차 쓰기, 순차 다시쓰기	
랜덤 I/O	랜덤 읽기, 랜덤 쓰기	
페이지 사이즈	4K, 8K, 16K, 32K, 64K, 128K, 256K	
파일크기	8Mbyte	

UBIFS 파일시스템의 I/O 성능 분석은 동일한 파일크기에 서로 다른 페이지사이즈를 통해 성능을 테스트한다. 파일크기는 일반적으로 많이 사용하는 8Mbyte로 하였으며 파일의 페이지 단위는 4K, 8K, 16K, 32K, 64K, 128K, 256K의 6가지 크기로 나누어 실험 한다.

실험은 ARM계열의 S3C6410 프로세스와 128MB 메모리, 512MB 낸드 플래시 메모리를 탑재한 타겟보드에서 수행하였다. 리눅스 커널은 2.6.29 버전을 사용하였으며, UBI 서브시스템과 UBIFS 파일시스템은 모듈

형태로 리눅스 커널 상에 적재하였다. 실험은 Sys-bench 0.40.12[12]를 사용하여 순차 및 랜덤 I/O의 데이터를 수집하고 분석하였으며, 마모평준화 실행 방지를 위하여 <표 7>의 MTD 유틸리티의 명령을 매회 수행하였다.

<Table 7> MTD 유틸리티

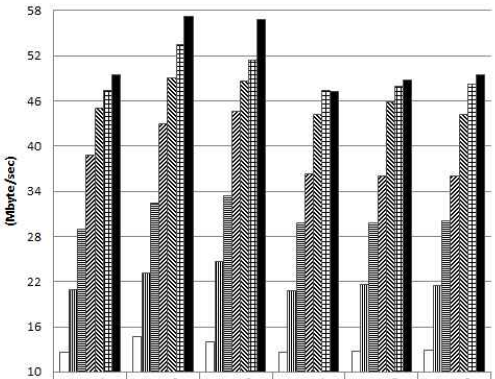
MTD 유틸리티	내 용
flash_eraseall	플래시 메모리 전체 지움
ubiformat	UBIFS 파일 포멧
ubimkvol	UBIFS 볼륨 생성

4. I/O 성능 분석 결과

본 장에서는 앞 장에서 제안한 UBI 서브시스템과 UBIFS 파일시스템에 메모리 할당방법에 따라 순차 I/O 성능의 분석 결과와 랜덤 I/O 성능의 분석결과를 차례로 기술한다.

4.1 순차 I/O 성능

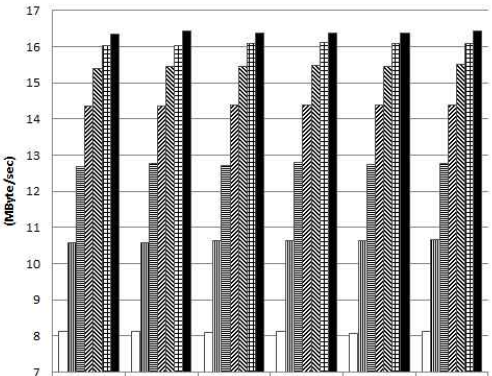
<그림 5>는 순차 I/O의 읽기 성능을 측정한 결과를 나타낸다.



	case1	case2	case3	case4	case5	case6
4k	12.603	14.722	14.009	12.603	12.805	12.864
8k	20.962	23.104	24.706	20.812	21.624	21.532
16k	28.973	32.483	33.415	29.813	29.77	30.112
32k	38.872	43.002	44.668	36.31	36.09	36.023
64k	44.989	49.032	48.606	44.177	45.925	44.196
128k	47.383	53.48	51.444	47.398	47.95	48.258
256k	49.518	57.221	56.864	47.273	48.794	49.483

<FIg 5> 순차 I/O 읽기 성능 결과

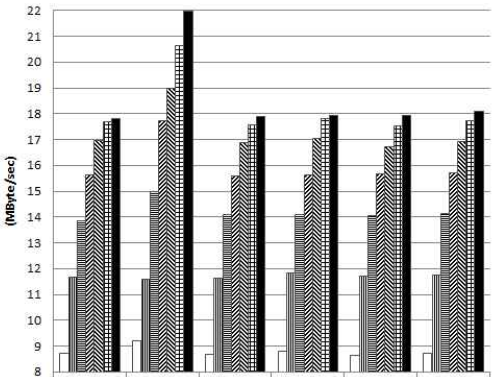
Case 2는 4K, 64K, 128K, 256K 4개 항목에서 높은 전송률을 보였으며, Case 3은 8K, 16K, 32K 3개 항목에서 높은 전송률을 볼 수 있다. 따라서 Case 1과 비교하여 Case 2는 12.45%, Case 3은 12.96% 향상된 전송률을 보이므로 순차 I/O 읽기 에서는 Case 2와 Case 3의 성능이 높다. <그림 6>은 순차 I/O의 쓰기 성능을 측정한 결과를 보여준다.



	case1	case2	case3	case4	case5	case6
4k	8.1228	8.1294	8.114	8.1202	8.0787	8.1239
8k	10.598	10.577	10.633	10.638	10.633	10.684
16k	12.699	12.766	12.723	12.799	12.739	12.779
32k	14.376	14.363	14.391	14.398	14.399	14.399
64k	15.413	15.462	15.45	15.475	15.449	15.503
128k	16.044	16.029	16.088	16.13	16.101	16.098
256k	16.347	16.44	16.372	16.384	16.368	16.445

<Fig 6> 순차 I/O 쓰기 성능 결과

Case 6는 8K, 32K, 64K, 256K 4개 항목에서 미세한 차이로 조금 높은 전송률을 보였다. 따라서 Case 6은 0.45%, Case 4는 0.35%로 메모리 할당방식에 상관없이 큰 차이를 보이지 않았다. <그림 7>은 순차 I/O의 다시쓰기 성능을 측정한 결과를 보여준다.



	case1	case2	case3	case4	case5	case6
4k	8.7326	9.2093	8.699	8.8032	8.6547	8.7251
8k	11.669	11.61	11.651	11.831	11.727	11.764
16k	13.877	14.975	14.108	14.099	14.042	14.136
32k	15.621	17.757	15.601	15.636	15.67	15.734
64k	16.964	19.004	16.872	17.05	16.708	16.921
128k	17.699	20.637	17.591	17.822	17.527	17.718
256k	17.793	21.961	17.893	17.943	17.947	18.084

<Fig 7> 순차 I/O 다시쓰기 성능 결과

순차 I/O 다시쓰기 성능 결과는 <그림 6>에서 나타난 순차 I/O 쓰기 결과보다 높은 속도를 보였으며 특히, Case 2는 11.23% 향상된 전송률을 볼 수 있다. 따라서 Case 1과 비교하여 6개 항목 4K, 16K, 32K, 64K, 128K, 256K에서 높은 속도를 보여준다. <표 8>은 UBI 서브시스템과 UBIFS 파일시스템에 기본으로 사용하는 Case 1을 기준으로 순차 I/O 성능 결과를 백분율로 분석한 결과이다.

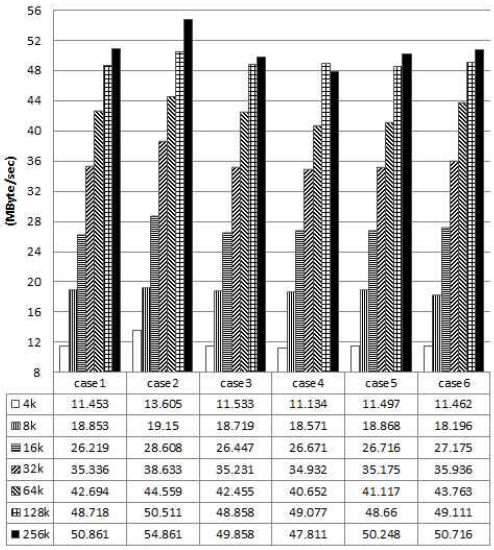
<Table 8> 순차 I/O 성능 결과

유형 \ I/O	순차읽기 (%)	순차쓰기 (%)	순차 다시쓰기(%)
Case 2	4K	16.81	0.08	5.46
	8K	10.22	-0.20	-0.51
	16K	12.11	0.53	7.91
	32K	10.62	-0.09	13.67
	64K	8.99	0.32	12.03
	128K	12.87	-0.09	16.60
	256K	15.56	0.57	23.42
	평균	12.45	0.16	11.23
Case 3	4K	11.16	-0.11	-0.38
	8K	17.86	0.33	-0.15
	16K	15.33	0.19	1.66
	32K	14.91	0.10	-0.13
	64K	8.04	0.24	-0.54
	128K	8.57	0.27	-0.61
	256K	14.84	0.15	0.56
	평균	12.96	0.17	0.06
Case 4	4K	0.00	-0.03	0.81
	8K	-0.72	0.38	1.39
	16K	2.90	0.79	1.60
	32K	-6.59	0.15	0.10
	64K	-1.80	0.40	0.51
	128K	0.03	0.54	0.69
	256K	-4.53	0.23	0.84
	평균	-1.53	0.35	0.85
Case 5	4K	1.60	-0.54	-0.89
	8K	3.16	0.33	0.50
	16K	-9.51	0.31	1.19
	32K	-7.16	0.16	0.31
	64K	2.08	0.23	-1.51
	128K	2.82	0.36	-0.97
	256K	-1.46	0.13	0.87
	평균	-1.21	0.14	-0.07
Case 6	4K	2.07	0.01	-0.09
	8K	2.72	0.81	0.81
	16K	3.93	0.63	1.87
	32K	-7.33	0.16	0.72
	64K	-1.76	0.58	-0.25
	128K	1.85	0.34	0.11
	256K	-0.07	0.60	1.64
	평균	0.20	0.45	0.69

순차 I/O 성능 결과에서 순차 읽기는 Case 3이 12.96%로 가장 높은 전송률을 보였고, 순차 쓰기는 Case 6이 0.45%로 높았지만, 전체적으로 큰 차이를 보이지 않았다. 하지만 순차 다시쓰기는 Case 2의 경우만 11.23%의 성능 향상을 보였으며 다른 Case는 큰 변화가 없었다.

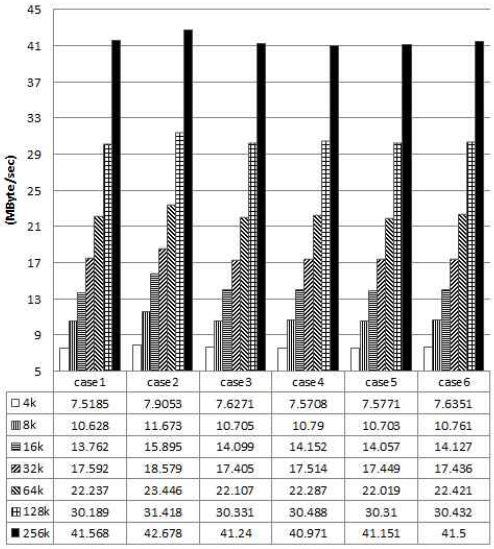
4.2. 랜덤 I/O 성능

<그림 8>은 랜덤 I/O의 읽기 성능을 측정한 결과를 보여준다.



<Fig 8> 랜덤 I/O 읽기 성능 결과

Case 2는 4K부터 256K까지 7개 항목 모두 높은 전송률을 보였으며, Case 1과 비교하여 7.82% 향상된 성능을 볼 수 있다. <그림 9> 랜덤 I/O의 쓰기 성능을 측정한 결과를 보여준다.



	case 1	case 2	case 3	case 4	case 5	case 6
4k	7.5185	7.9053	7.6271	7.5708	7.5771	7.6351
8k	10.628	11.673	10.705	10.79	10.703	10.761
16k	13.762	15.895	14.099	14.152	14.057	14.127
32k	17.592	18.579	17.405	17.514	17.449	17.436
64k	22.237	23.446	22.107	22.287	22.019	22.421
128k	30.189	31.418	30.331	30.488	30.31	30.432
256k	41.568	42.678	41.24	40.971	41.151	41.5

<FIg 9> 랜덤쓰기 I/O 성능 결과

랜덤 I/O 쓰기에서는 Case 2가 7개 항목 모두 높은 성능을 나타냈으며, Case 1과 비교하여 약 5.6% 높은 성능을 보였다. <표 9>는 Case 1을 기준으로 랜덤 I/O 성능 결과를 백분율로 계산한 결과이다.

랜덤 I/O 성능 결과에서 랜덤 읽기는 Case 2가 7.82%로 가장 높은 전송률을 보였으며, Case 6이 0.71%로 낮은 성능을 보였다. 나머지 Case 3 ~ Case 5는 Case 1에 비해 성능이 낮음을 알 수 있었다. 랜덤 쓰기는 Case 3 ~Case 6 모두 미세한 성능 향상을 보였지만 Case 2가 6.90%으로 높은 전송률을 보였다.

5. 결 론

최근 스마트기기의 일반적인 성능 저하 원인으로 프로세서의 성능과 무선네트워크의 연결 문제로 보고 있다. 하지만, 낸드 플래시 메모리의 빈번한 I/O가 성능 저하의 더 큰 원인으로 분석되고 있다. 특히 I/O 성능은 순차 I/O 보다 랜덤 I/O의 데이터 전송량이 낮기 때문에 랜덤 I/O 성능은 낸드 플래시 메모리의 성능과 밀접한 관계가 있다고 할 수 있다.

본 논문에서는 낸드 플래시 메모리의 I/O 성능을 UBI 서브시스템과 UBIFS 파일시스템에서 I/O 성능을 분석하였다. 성능 분석은 UBI 서브시스템의 2개의 버퍼와 UBIFS 파일시스템의 4개의 버퍼에 메모리 할당 기법에 따라 6가지로 분류하고, 순차 및 랜덤 I/O로 I/O 성능을 분석하였다. 순차 I/O는 읽기, 쓰기, 다시

<Table 9> 랜덤 I/O 성능 결과

유형 \ I/O	랜덤읽기(%)	랜덤쓰기(%)
Case 2	4K	18.79	5.14
	8K	1.58	9.83
	16K	9.11	15.50
	32K	9.33	5.61
	64K	4.37	5.44
	128K	3.68	4.07
	256K	7.86	2.67
	평균	7.82	6.90
Case 3	4K	0.70	1.44
	8K	-0.71	0.72
	16K	0.87	2.45
	32K	-0.30	-1.06
	64K	-0.56	-0.58
	128K	0.29	0.47
	256K	-1.97	-0.79
	평균	-0.24	0.38
Case 4	4K	-2.79	0.70
	8K	-1.50	1.52
	16K	1.72	2.83
	32K	-1.14	-0.44
	64K	-4.78	0.22
	128K	0.74	0.99
	256K	-6.00	-1.44
	평균	-1.96	0.63
Case 5	4K	0.38	0.78
	8K	0.08	0.71
	16K	1.90	2.14
	32K	-0.46	-0.81
	64K	-3.69	-0.98
	128K	-0.12	0.40
	256K	-1.21	-1.00
	평균	-0.44	0.18
Case 6	4K	0.08	1.55
	8K	-3.48	1.25
	16K	3.65	2.65
	32K	1.70	-0.89
	64K	2.50	0.83
	128K	0.81	0.80
	256K	-0.29	-0.16
	평균	0.71	0.86

쓰기의 3가지 동작과 랜덤 I/O는 랜덤읽기, 랜덤쓰기의 2가지 동작으로 구분하여 테스트를 수행하였다.

순차 I/O의 성능 분석결과 읽기와 다시쓰기 I/O에서 Case 1에 비해 Case 2의 경우가 가장 높은 전송률을 보였고, 랜덤 I/O의 경우는 읽기, 쓰기 모두 Case 2의 경우가 높은 전송률을 보였다. 따라서 기존 메모리 할당 방법에서 vmalloc()와 kmalloc()을 다르게 함으로써 순차 I/O는 최대 12%, 랜덤 I/O는 7% 이상의 성능 향상을 볼 수 있었다.

본 논문에서 제안한 UBIFS 메모리 할당에 관한 I/O 성능 분석은 기존 메모리 할당방식을 다르게 적용함으로써 더욱 향상된 I/O 성능을 가져온다. 따라서 제안된 성능분석은 향후 스마트기기 성능 향상을 위한 연구의 자료로 활용할 수 있을 것으로 기대된다.

References

[1] H. Kim, N. Agrawal, and C. Ungureanu, “Revisiting storage for smartphones,” in Proc. of the 10th USENIX conference on File and storage technologies, 2012.

[2] 이영석, 이창희, 정경호, 김용환, 안광선, "NAND 플래시 파일시스템의 I/O 스케줄러 성능분석", 한국산업정보학회논문지, 제18권, 제2호, pp.27-34, 2013.

[3] F. Douglis, R .Caceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber, “Storage Alter-natives for Mobile Computers,“ Proceedings of the 1st Symposium on Operating System Design and Implementation(OSDI), pp. 25-37, 1994.

[4] MTD(Memory Technology Devides), “http://www.linux-mtd.infradead.org/doc/general.html”.

[5] YAFFS2(The Yet Another Flash File System2), “http://www.yaffs.net/”.

[6] JFFS2(Journaling Flash File System2), “http://www.sourceware.org/jffs2/”.

[7] UBIFS(Unsorted Block Images File-System). “http://www.linux-mtd.infradead.org/doc/ubifs.html”.

[8] UBI(Unsorted Block Images), “http://www.linux-mtd.infradead.org/doc/ubi.html”.

[9] 한동훈, “리눅스 커널 프로그래밍”, 한빛미디어, 2007.

[10] J. Bonwick, “The Slab Allocator: An Object-Caching Kernel Memory Allocator”, USENIX Summer 1994 Technical Conference, pp. 87-98, 1994.

[11] Anatomy of the Linux slab allocator, “https://www.ibm.com/developerworks/library/l-linux-slab-allocator/”.

[12] benchmarking, “http://sysbench.sourceforge.net”.

[13] 최훈, 최유정, "스마트폰 애플리케이션 품질이 신뢰 및 지속적 사용의도에 미치는 영향:성별의 조절효과를 중심으로", 한국산업정보학회논문지, 제16권, 제4호, pp.151-162, 2011.

[14] 김순철, "가변 비트율 주문형 비디오 서버에서 자원 활용률을 높이기 위한 버퍼 관리기법", 한국산업정보학회논문지, 제9권, 제3호, pp.1-10, 2004.

이 재 강 (Jaekang Lee)

- 학생회원
- 가야대학교 컴퓨터공학과 학사
- 경북대학교 컴퓨터공학과 석사
- 경북대학교 IT대학 컴퓨터학부 박사수료

- 관심분야 : 임베디드 시스템, 파일시스템, RFID

오 세 진 (Sejin Oh)

- 경운대학교 컴퓨터공학과 학사
- 경북대학교 전자전기컴퓨터학부 석사
- 경북대학교 IT대학 컴퓨터학부 박사수료

- 관심분야 : RFID, 충돌방지, 정보보호, 프로트콜, 임베디드 시스템

정 경 호 (Kyungho Chung)

- 대구대학교 컴퓨터정보공학과 학사
- 경북대학교 컴퓨터공학과 석사
- 경북대학교 컴퓨터공학과 박사
- 경북대학교 IT대학 컴퓨터학부 외래교수

- 관심분야 : 임베디드 시스템, RFID, 정보보호

윤 태 진 (Taejin Yun)

- 경북대학교 컴퓨터공학과 학사
- 경북대학교 컴퓨터공학과 석사
- 경북대학교 컴퓨터공학과 박사
- 경운대학교 IT에너지대학 모바일학과 교수

- 관심분야 : 정보보안, 센서네트워크, 임베디드 시스템

안 광 선 (Kwangseon Ahn)

- 정회원
- 연세대학교 전기공학과 학사
- 연세대학교 전자공학과 석사
- 연세대학교 전자공학과 박사
- 경북대학교 IT대학 컴퓨터학부 교수

- 관심분야 : 임베디드 시스템 설계, RFID 시스템

논 문 접 수 일 : 2013년 05월 24일
1차수정완료일 : 2013년 06월 14일
2차수정완료일 : 2013년 07월 09일
게 재 확 정 일 : 2013년 07월 29일