안드로이드 기반 모바일 플랫폼을 위한 새로운 NAND 플래시 메모리 파일 시스템

오용석° 박찬익
포항공과대학교 정보통신공학과
justgn@postech.ac.kr° cipark@postech.ac.kr

A New NAND Flash Memory File System for Android Based Mobile Platform

Yong Seok Oh° Chan Ik Park
Department of GIST Pohang University of Science and technology

요 약

플래시 메모리는 비 휘발성, 저전력, 빠른 입출력, 충격에 강함 등과 같은 많은 장점을 가지고 있으며, 휴대폰, MP3, PDA, 디지털 카메라와 같은 다양한 멀티미디어 기기에 널리 적용 되었고 그 사용이 지속적으로 증가해 왔다. 이에 따라 임베디드 디바이스에 적용되는 NAND 플래시 메모리 전용 파일 시스템에 대한 연구가 활발히 이루어 지고 있다. 전통적으로 플래시 메모리 파일시스템에 대한 연구는 Garbage collection, Out-Place Update, Wear-leveling 등을 중요시 해 왔으나, 플래시 메모리의 용량이 증가함에 따라서 빠른 초기화와 빠른 응답시간이 플래시 메모리 파일시스템에 주용한 이슈가 되었다. 본 논문에서는 빠른 초기화와 응답시간을 위한 안드로이드 기반 모발일 플랫폼에 적용가능 한 새로운 NAND 플래시 메모리 전용 파일 시스템을 제안하고 기존 NAND 플래시 메모리 파일 시스템과의 성능을 비교한다.

1. 서 론

플래시 메모리는 전력공급이 없이도 데이터를 저장할 수 있는 비 휘발성 특성과 외부 충격에 강하며, 전력 소모량이 작은 특징 등으로 인하여 휴대폰 PDA, PMP와 같은 모바일 기기의 저장 매체로 그 사용이 증가하고 있다.[1] 플래시 메모리는 기존 디스크와 달리 그 특성으로 인하여 한번 데이터를 쓴 페이지는 블록을 지우기 전가지 다시 사용할 수 없다. 또한 읽는 속도는 매우 빠르지만, 쓰기 속도와 지우는 속도가 상대적으로 느리며, 한번에 지울 수 있는 크기가 일정하며, 지움 연산의 횟수가 제한적이다. 이런 플래시 메모리는 NOR 형태와 NAND형태의 2가지 형태로 나눌 수 있으며, NOR형 플래시 메모리의 경우 NAND 플래시 메모리 보다 빠른 읽기 속도를 가진다. 하지만 NAND 플래시 메모리에 비해 NOR 플래시 메모리는 단위 용량당 가격이 높으며 따라서 대용량 임베디드 장치에 사용이 어렵다.

플래시 메모리를 이용하여 파일 시스템 서비스를 제공하기 위한 방법은 크게 두 가지로 나뉜다. 첫 번째 방법은 FTL(Flash Transaction Layer)[2]를 이용하여 기존 디스크에서 사용해오던 파일 시스템 (FAT 또는 EXT2/3)을 사용하는 방법과 플래시 메모리의 물리적 특징을 이해하고 효율적으로 관리가 가능한 플래시 메모리 전용 파일 시스템을 이용하는 것이다. NAND 플래시 메모리 전용 파일 시스템에 대한 대표적인 예로 JFFS2(Journaling Flash File System)[3], YAFFS2(Yet Another File System)[4] 등이 있다. FTL의 경우 순차적인 플래시 공간이 디스크의 섹터처럼 보이게 하기 위하여 논리 주소와 물리 주소간의 매핑(Mapping) 관리를 수행하는 방법을 사용하며, JFFS2는 플래시 메모리 공간을 순차적으로 저장하는 LFS(Log-structured File System)[5] 처럼 플래시 메모리에 대한 갱신 연산을 다른 빈 페이지에 기록함으로써 플래시 메모리상의 지움 연산을 최소화 하는 방법을 사용한다. 이를 통해서 Out-Place 업데이트를 연산을 최소화하는 방법으로 사용한다. YAFFS2의 경우 역시 LFS의 방법을 사용하며 최근 업데이트 된 YAFFS2의 경우 checkpoint 정책을 도입하여 빠른 초기화 수행을 지원하고 있다.

본 논문에서는 기존 NAND 플래시 메모리 전용 파일 시스템과 같은 LFS 방법을 사용하여 빠른 초기화 및 빠른 I/O 응답시간을 보장하며, 효율적인 복구 기법을 제한한다.

본 논문의 구성은 다음과 같다. 2장에서는 배경 지식에 대하여 소개하고 3장에서는 관련연구에 대하여 기술 한다. 4장에서는 빠른 응답 시간과 빠른 초기화를 위한 파일 시스템을 소개하며 5장에서는 기존 파일

[1] "본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음" (NIPA-2009-C1090-0902-0045)

시스템과 그 성능을 비교한다. 마지막으로 6장에서 결론을 맺는다.

2. 배경 지식

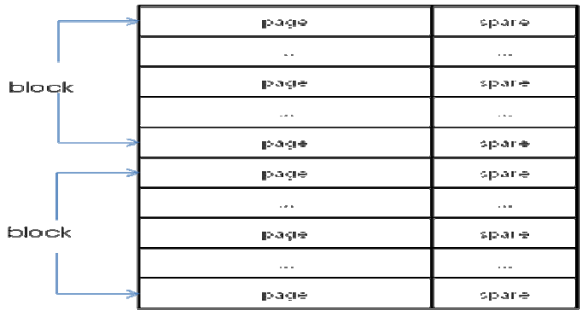
2.1 NAND Flash Memory

플래시 메모리에서 데이터를 읽고 쓰고 하는 기본적인 단위는 페이지 단위이며 한 페이지의 크기는 약 512바이트 에서 2048바이트 이다. 각 페이지는 해당 페이지에서 발생 할 수 있는 에러를 봉정하고 관련 메타 데이터를 보관 할 수 있는 여분의 공간을 가지고 있으며 이것을 Spare 영역 또는 OOB(Out-Of-Band) 영역이라 한다. 플래시 메모리는 기본적인 연산의 단위인 페이지들을 블록으로 구성하고 이 블록은 플래시 메모리 상에서 관리하는 최소의 단위가 된다. 데이터를 읽고 쓰는 연산은 기본적으로 페이지 단위로 관리가 되며 페이지를 지울 때 즉 데이터를 지우는 연산은 블록을 기본 단위로 하여 이루어 진다. 다음 <표1> 은 플래시 메모리의 기본 연산 수행 시간을 보여준다.

<표 1 플래시 메모리 기본 연산 수행시간>

Media	Read	Write	Erase
DRAM	60ns(2B) 2.56us(512B)	60ns(2B) 512us(512)	N/A
NOR Flash	150ns(1B) 14.4us(512B)	211ns(2B) 3.53us(512B)	1.2s(128KB)
NAND Flash	10.2us(1B) 35.9us(512B)	201us(2B) 226us(512B)	2ms(16KB)
Disk	12.4ms(512B) (Average)	12.4ms(512B) (Average)	N/A

플래시 메모리는 블록들로 구성되며 각 블록들은 페이지들로 구성 된다. 각 블록의 경우 32개 혹은 그 이상의 페이지 들로 구성이 되며 각 페이지는 512Byte 에서 2048Byte의 크기를 갖는다. 또한 플래시 메모리 내부의 여유 공간이 부족한 경우 블록 내부의 유효하지 않은 페이지를 정리하여 여유공간을 할당 하는 Garbage Collection 작업을 수행한다. 위 <그림 1>은 NAND 플래시 메모리의 구조를 보여준다.



<그림 1 플래시 메모리 내부 구조>

3. 관련 연구

플래시 메모리를 이용하여 파일 시스템 서비스를 제공하기 위한 연구는 크게 두 가지로 나뉘며 그 중 하나가 플래시 메모리의 특성을 이해하고 플래시 메모리에 특성화된 파일 시스템이다. 다른 하나는 FTL과 같이 기존 파일 시스템을 사용하면서 마치 플래시 메모를 블록 디바이스와 같이 에뮬레션 해주는 것이다. FTL은 플래시 메모리와 블록 디바이스의 중간에 위치하며 플래시 메모리를 블록 디바이스처럼 보이게 한다. 즉 논리 주소에 대한 쓰기 요청을 가능한 물리 주소에 수행하고 그 매핑 정보를 다로 관리하는 방법이다. 따라서 FTL을 사용하게 되면 기존 블록 디바이스 파일 시스템을 이용하여 플래시 메모리를 블록 디바이스처럼 사용할 수 있다. 플래시 메모리 파일 시스템 중 대표적인 것은 JFFS와 YAFFS가 있으며 JFFS2[]는 199년에 발표한 JFFS를 발전 시킨 것으로 LFS[] 구조를 사용한다. LFS는 파일 시스템의 메타데이터와 데이터를 묶어 로그 구조로 만들고 이를 디스크에 순차적인 방법을 사용하여 저장해 나가는 방식을 말한다.

3.1 JFFS2 (Journaling Flash File System 2)

JFFS는 데이터를 로그 형태로 플래시 메모리에 순차적으로 쓰고, 읽기 연산은 로그를 역순으로 검색하여 가장 최신의 데이터를 읽어 들인다. JFFS에서는 저널링 노드로써 jffss_node 라는 구조체를 사용하며 저널링 노드의 크기는 48Byte로 큰 부분을 차지 한다. 이런 부하를 줄이기 위해서 JFFS2는 next_in_ino, next_phys, flash_offset, totlen 값만으로 구성된 jffs2_raw_node_ref라는 구조체(16Byte)로 jffs_node를 대체하여 메모리에 저장한다. 하지만 플래시 메모리 용량의 상당 부분을 jffs2_raw_node_ref를 위해 사용해야 하며 노드를 찾고 파일의 구조를 결정하기 위한 스캔 시간이 길다. 또한 마운트할 때 플래시 메모리 전체를 스캔해야 하기 때문에 플래시 메모리의 용량에 따라서 마운트 시간이 길어 지게 된다. JFFS의 경우 NOR 플래시 메모리를 기반으로 설계되었기 때문에 NAND 플래시 메모리용 파일 시스템으로 사용되기에는 메모리 사용량, 마운트 및 플래시 메모리 스캔 시간 그리고 Garbage Collection 시간 등에서 문제점이 있다.

3.2 YAFFS2 (Yet Another Flash File System)

YAFFS는 JFFS2가 저널링에 사용되는 메모리 소모량이 큰 것과, 느린 마운트 성능을 개선하기 위해서 개발된 NAND 플래시 전용 파일 시스템이다. YAFFS2는 2002년 Alepha on 사에서 개발된 NAND 플래시 메모리

전용 파일 시스템으로 로그 구조를 기반으로 하여 플래시 메모리에 대한 업데이트 연산을 추가 연산으로 변형하여 처리하는 Out-Place 업데이트 기법을 사용한다. 이를 통하여 플래시 메모리의 in-place 업데이트가 안 되는 단점을 에 대한 문제를 해결 한다.

YAFFS2는 쓰기 연산을 위한 공간 요청 발생 시 빈 블록을 찾기 위해서 블록 상태 정보를 순차적으로 검색하고 이를 통한 할당으로 인해 블록 사용 횟수에 대한 고려 없이 할당 하게 되므로 플래시 메모리의 균등 사용을 지원하지 못하였다. 또한 순차적으로 블록을 찾기 때문에 초기화 시점에 지연이 발생하였다. 하지만 최근 업데이트된 YAFFS2는 체크포인트 기능을 도입하여 초기화 시 오래 걸리는 단점을 보완 하고 있다. 체크 포인트 기능은 초기화를 수행할 때 NAND플래시 메모리 전체를 다시 순차적으로 검색하는 오버헤드를 줄이기 위해서 제안되었다. 체크포인트 기능은 현재 NAND 플래시 메모리의 상태를 마운트가 해제되는 시점에 기록하고 다음 초기활 때에는 YAFFS2는 파일 시스템을 재구성하기 위하여 NAND 플래시 전체를 검색하지 않고 체크 포인트 영역을 읽어 들여 파일 시스템의 상태를 복원한다. 하지만 이 체크 포인트 기능은 마운트가 정상적으로 해제되지 않은 다면 체크 포인트는 무시 되며 결과적으로 파일 시스템 상태를 복원하기 위해서 NAND 플래시 메모리 전체를 다시 검색하여 읽어 와야 한다.

NAND 플래시 메모리의 기본 단위는 앞서 설명한 바와 같이 페이지와 Spare 영역이며 이들을 모아서 블록을 만든다. YAFFS2에서는 페이지에 대항 하는 것을 Chunk, Spare에 해당 하는 것을 Tag라 부르며 Chunk에 Object 헤더와 파일 데이터를 저장하며, tag에는 chunk와 Object의 정보들을 저장한다. 하나의 파일은 여러 개의 최소 2개 이상의 Chunk와 Tag로 이루어 지며 같은 Object ID값을 가진다. 여러 개의 chunk는 Chunk ID라는 숫자로 구분하며 첫 번째 Chunk에 기본 적으로 해당하는 Object 헤더가 기록 된다. YAFFS2는 Tnode-Tree를 구성하여 하나의 파일을 구성하고 있는 chunk를 관리 한다. 하나의 Tnode 엔트리가 가지고 있음 값은 페이지의 물리 주소가 아닌 16개의 페이지들로 구성된 그룹 주소를 가지고 있으며 따라서 요구 하는 페이지를 찾기 위해서는 해당 페이지들을 순차적으로 각 페이지의 tag 영역에 있는 논리 주소와 일치하는 지를 확인 하는 과정을 통해서 요구 페이지를 찾고 원하는 데이터를 얻는다.[6]

4. 디자인 및 구현

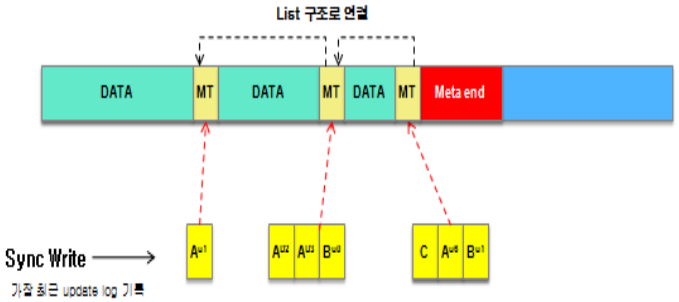
본 논문은 YAFFS2 파일 기스템을 기반으로 구현이 되었으며 다음과 같은 3가지 구성으로 나뉜다.

- 파일 시스템 마운트 시간 단축(빠른 초기화)
- 빠른 응답시간
- 파일시스템 에서 상황에서 빠른 복구

모바일 기기의 부팅 시간 및 빠른 응답성을 위하여 빠른 초기화 및 I/O 응답시간을 개선하고 플래시 메모리의 에러로 인한 복구 시간을 단축하는 것을 목적으로 구현되었다.

4.1 빠른 초기화 기법

본 논문에서 제안하는 NAND 플래시 메모리 전용파일 시스템은 다음 <그림 2>와 같이 초기화를 위한 메타 관리 블록을 구성하고 메타 관리 블록과 데이터 블록을 나누어 구성한다. 메타 관리 블록은 플래시 메모리 상의 빈 블록에 할당이 되며 Sync Manager에 의해 관리 된다. 메타 관리 블록은 매 파일 시스템의 업데이트 된 정보를 관리 하며 각 파일마다 업데이트 되는 정보를 기록한다. 메타 관리 블록에 기록되는 정보는 Object 번호화 해당 Object의 데이터 맵핑 트리 등과 같은 정보를 기록한다.

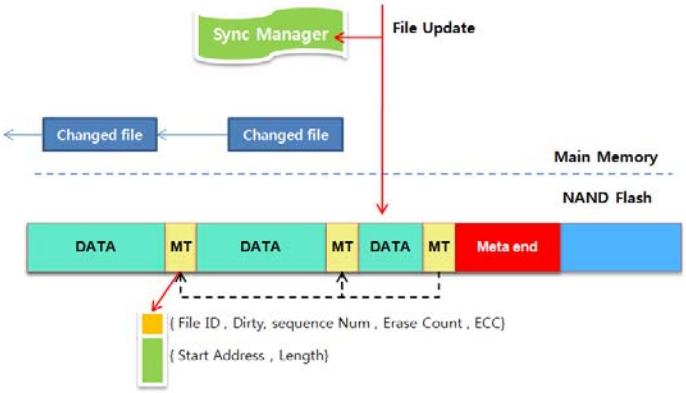


<그림 2 메타 관리 블록>

메타 관리 블록은 필요 시 마다 빈 블록을 할당하여 기록하며 메타 관리 블록 할당 시 미리 고정된 개수의 블록을 미리 할당하여 파일 연산 발생시 메타 관리 블록의 할당으로 인한 오버헤드를 줄인다. 메타 엔드(Meta end)블록의 경우 파일 시스템의 마운트 해제 시 기록되는 정보로 현재 NAND플래시 메모리의 상태정보 (e.g. 할당된 블록 개수, 여유블록 개수 등)와 각 블록에 대한 정보 (e.g. 페이지 사용 개수, 블록 상태 등.), 현재 사용중인 Chunk의 Ussage Bitmap, 파일 시스템 내의 Object 정보 등을 기록한다.

마운트 수행 시 제안하는 NAND 플래시 메모리 파일 시스템의 경우 메타 엔드 블록을 통해서 마운트를 수행하고 NAND 플래시의 경우 Object를 접근하기 위해서는 해당 Object의 데이터 매핑 트리를 구성해야 하기 때문에 메타 데이터 블록을 읽어 해당 Object의 매핑을 로딩한다. 단지 메타 엔드 블록만을 읽어 마운트를 수행하기 때문에 마운트 수행 시 별도의 NAND 플래시 메모리 스켄 작업이 필요하지 않다. 또한 Object의 데이터 매핑은 예를 들어 파일이 변경이 될 때마다 메타 관리 블록에 기록이 되므로 메타 관리 블록을 스켄하여 쉽게 해당 파일에 대한 데이터 매핑을 로드 할 수 있다. 메타 관리 블록에 파일 업데이트 연산

시 기록되는 오버헤드를 줄이기 위해서 Sync Manager를 두고 파일 시스템이 일정시간 동안 사용되지 않을 때 즉 Idle 타임에 메타데이터 블록을 갱신하는 방법을 사용하여 메타관리 블록과 데이터 블록의 쓰기 지연을 최소화 한다. 다음 <그림 3> 은 Sync Manager에 의해 관리 되는 것을 보여주는 그림이다.



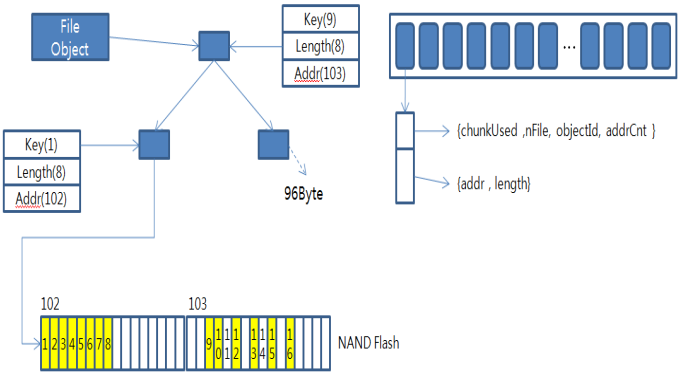
<그림 3 빠른 초기화를 위한 Sync Manager>

앞서 언급한 바와 같이 파일 시스템 이미지의 변화를 추적하기 위해서 매 I/O 수행 시 영향을 받는 파일을 확인하기 위해서 Filter function이 삽입 되었다. 이 Filter Function은 파일 이름을 확인 할 수 잇는 가장 저 수준에 위치한다. 본 논문에서 제안하는 파일 시스템은 데이터 영역에 저장하는 파일들을 추적하기 위해서 메타 관리 블록에 각 파일 마다 하나의 로그 페이지를 할당 하고 파일을 갱신하게 되면 Sync Manager는 해당 파일이 최근 갱신 목록에 있는지 확인하고 갱신 목록에 존재 하지 않으면 해당 파일을 갱신 목록에 추가, 메타 관리 블록의 로그 페이지는 더 이상 유효하지 않은 값을 가지므로 해당 로그 Chunk의 tag영역에 Dirty 표시를 한다.

4.2 빠른 I/O 응답시간을 위한 Chunk Mapping 기법

본 논문에서 제안하는 파일 시스템은 데이터 Chunk의 매핑을 위하여 B-Tree를 사용한다. B-Tree를 이용하여 빠른 Search 성능을 가질 수 있으며 또한 기존 YAFFS2는 페이지를 매핑하기 위해서 앞서 언급한 바와 같이 페이지 그룹을 매핑하지만 제안하는 파일 시스템은 페이지를 직접 매핑함으로써 그룹내의 페이지를 순차적으로 찾아 확인하는 오버헤드를 최소화 한다. 기존 YAFFS2의 Tnode Tree의 경우 NAND 플래시 메모리가 커질수록 실제 메모리 상에 유지해야 하는 매핑의 크기를 줄일 수 있지만 I/O 성능에는 악영향을 준다. 따라서 B-Tree 직접 매핑을 통하여 I/O 성능을 향상시키고 기존 메인 메모리 상의 매핑 데이터를 줄이기 위해서 Range 형식의 맵핑을 가진다. Range 형식의 매핑의 경우 동일 블록내의 페이지는 Range 형식으로 매핑 함으로써 각 페이지를 인덱싱하는 오버헤드를 줄이고 또한 일정 시간 사용하지 않는 Objcet의 매핑을 메인 메모리 상에서 내려 메인 메모리 사용량을 줄였다. 하지만 기존 Tnode 매핑의 경우 보다 더 많은 메인 메모리를 사용하게 되는데 이는 RAM의 가격이 싸지고 대용량화 됨에 따라서 충분히 수용할 수 있을 것이다.

다음 <그림 4>는 제안하는 파일 시스템의 B-Tree 매핑의 예이다.



<그림 4 B-Tree Mapping>

- Key : 그룹 내의 해당 파일에 대한 Chunk의 첫 번째 chunked
- Length : 그룹내의 파일에 해당하는 Chunk 개수
- Addr : 해당 Chunk 속해 있는 그룹의 Base address

예를 들어 ChunkID가 4인 chunk를 찾기 위해서 B-tree를 검색하기 위해서는 102번 그룹의 시작 Key값이 1이고 그 크기가 8이므로 4번 Chunk는 102번 그룹에 해당하고 102 번 그룹의 시작 Chunk가 128번째 Chunk라고 할 때 실제 물리 주소는 128+4-1 즉 131번째 chunk임을 알 수 있다. 따라서 그룹을 순차적으로 감색하지 않고 바로 해당 Chunk를 찾을 수 있다.

4.3 파일시스템 오류 시 빠른 복구 기법

파일 I/O연산 수행 시 전원이 꺼지는 등의 오류로 인하여 파일 시스템 오류가 발생하게 되면 다음 마운트 수행 시 파일 시스템 상태를 복구 하기 위한 복구 프로세스가 동작 하게 된다. 해당 복구 프로세스는 기존 NAND 플래시 메모리 파일 시스템과 달리 전체 NAND 플래시 메모리를 스캔 하지 않고 메타 관리 블록만을 스캔하여 가장 최근 업데이트된 매핑을 복구 하는 방법을 사용하여 복구를 진행한다. 제안하는 파일 시스템은 복구 프로세스 중 전체 NAND플래시 영역을 스캔하여 최근 업데이트된 내용으로 복구하는 방법을 사용하지 않고 제한된 메타 관리 블록을 스캔하여 복구 프로세스를 진행 하기 때문에 기존 파일 시스템에 비하여 빠른 성능을 보인다. 복구의 경우 정확하게 데이터를 복구해 내는 것이 더 중요한 사항이며 이를 위해서 정확한 맵핑 정보가 필요하게 된다. 이는 메타

관리 블록에 이미 저장된 이전 업데이트 시점에 매핑 정보를 이용하여 복구를 수행하기 때문에 정확한 데이터 복구 수행을 보여 준다.

5. 성능 비교 분석

5.1 실험 환경

제안 하는 파일 시스템과 성능을 비교하기 위하여 본 논문에서는 YAFFS2와 성능을 비교한다. 실험 환경은 다음과 같다. 다음 <표 2> 실제 하드웨어 환경이다.

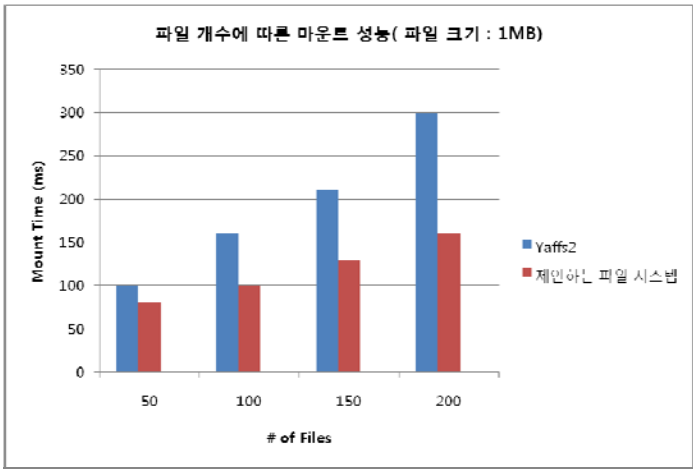
<표 2하드웨어 환경>

Processor	Intel Xscale 624MHz
SDRAM	Samsung 128M mobile SDRAM
NOR Flash	Intel StrataFlash 64M NOR
NAND flash	Samsung 256MB NAND
CDMA	CDMA 1x, HSDPA
T-DMB	넥실리온 NX3301
Camera	MCNEX 2M camera Module
Wireless LAN	USI 802.11b/g WLAN Module

실험은 Android 모바일 플랫폼 환경을 구성하기 위하여 Android 커널 2.6.27 버전의 커널을 이용하여 실험을 하였으며 하드웨어에 커널을 포팅하고 파일시스템의 성능을 측정하였다. 성능 실험을 위해서 다음과 같이 2가지 경우의 성능 실험을 수행하였다. IOZone[7] benchmark 툴을 이용하여 파일 시스템의 성능을 측정 하였으며 파일의 개수와 사용용량에 따른 마운트 타임과 I/O성능 및 복구 성능을 측정하는 실험을 수행하였다.

5.2 파일 개수에 따른 마운트 성능 비교

다음 <그림 4>의 경우 50~200개의 평균 1MB 파일을 기록하고 마운트 성능을 측정해본 결과이다.

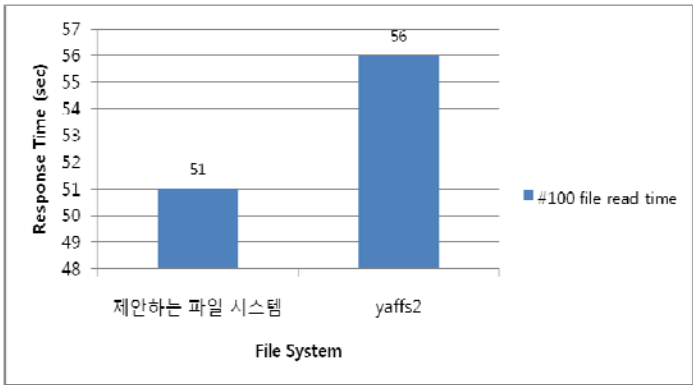


<그림 5 마운트 성능 비교>

제안하는 파일 시스템의 마운트 시간이 더 작은 것을 알 수 있다. 이 것은 YAFFS2의 경우 파일 Object 맵핑을 Checkpoint 시에 모두 기록하고 이를 마운트 수행 시 로딩하는 오버헤드로 인하여 제안하는 파일 시스템의 성능이 더 나음을 알 수 있다. 또한 파일 개수에 따라서 마운트 시간이 증가 하는 이유는 Object 정보를 기록하고 이를 로딩하는데 걸리는 오버헤드 이다.

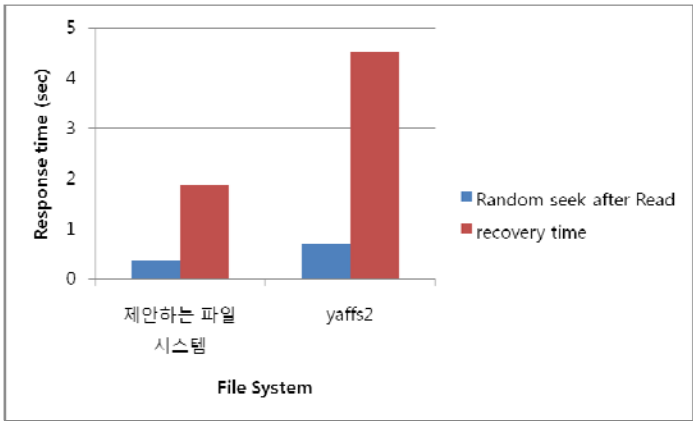
5.3 파일 I/O 성능 및 복구 시간 성능 비교

파일 I/O 성능 실험은 매핑 차이에 따른 성능을 알아보기 위하여 임의읽기를 수행하여 실험 하였다. 제안하는 파일 시스템은 임의 읽기가 기존 YAFFS2 파일 시스템에 비하여 더 낳은 성능을 보임을 알 수 있다.



<그림 6 1MB 파일 임의 선택하여 1MB 읽기>

또한 복구 시간을 측정하기 위하여 100MB 파일과 500KB 텍스트 파일을 생성 후 임의로 전원을 OFF하여 다음 마운트 수행 시 마운트 수행에 걸린 시간을 측정하였으며 임의 읽기 성능을 위해서 100MB 파일을 임의의 위치에서 1KB 읽는 것을 10,000번 수행한 수행 시간이다.

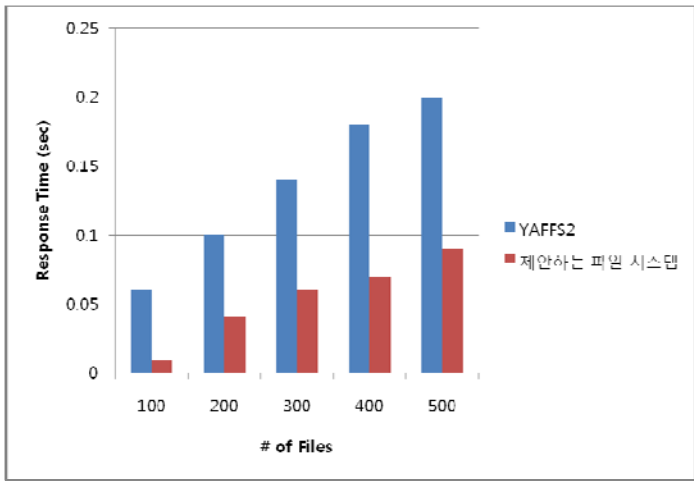


<그림 7 임의 읽기 및 복구 시간>

5.4 파일 개수에 따른 마운트 해제 수행시간

기존 YAFFS2의 경우 모든 Object에 대하여 해당 매핑을 마운트 해제 수행 시 모두 기록하며 제안하는

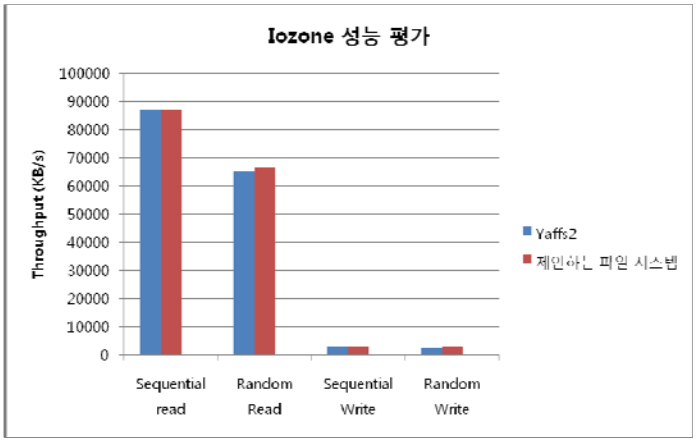
파일 시스템은 Object의 매핑을 별로도 마운트 해제 시 수행하지 않으며 기존 매핑의 경우 메타 관리 블록에서 관리 되고 있다.



<그림 8 파일 개수에 따른 마운트 해제 시간>

5.5. I/Ozone Benchmark 성능 실험

IOZone을 실험 보드에 포팅을 수행하고 그 성능을 측정 한다. 요청 크기는 4KB의 고정 크기이며 파일 크기는 150MB 파일을 대상으로 실험을 수행하였으며 다음 <그림 9> 그 결과 이다.



<그림 9 IOZone 성능 평가>

제안 하는 파일 시스템이 비교적 나은 성능을 보여줌을 알 수 있다.

6. 결론

본 논문에서 제안하는 파일 시스템은 구글의 안드로이드 환경에 적용 가능한 파일시스템으로 빠른 초기화를 지원하며, I/O 성능을 향상 시켰다. 기존 모바일 디바이스의 파일 시스템의 복구 수행 시간을 효율적으로 단축 시킬 수 있는 기법을 제안 하였으며 현재 안드로이드 커널상에 포팅 되었다. 성능 평가에서 나타난 바와 같기 기존 YAFFS2보다 나은 성능을 보이고 있다. 또한 모바일 디바이스의 스토리지 용량은 미래에 더 커질 것으로 예상이 되며 따라서 대용량 모바일 디바이스에 효과적은 내장형 플래시 메모리 전용 파일 시스템을 제안하였다.

Reference

[1] F. Douglis, R. Caceres, F. Kaashoek, K. Li,B.Marsh, and J. A. Tauber, “Storage Alternatives for Mobile Computers,” In Proceedings of the 1st Symposium on Operating Systems Design and Implementation, pp.25-37, 1994.

[2] Intel Corporation, ”Understanding the Flash Translation Layer (FTL) Specifica-tion”.

[3] D. Woodhouse, Red Hat, Inc. "JFFS: The Journaling Flash File System," Ottawa Linux Symposium,2001

[4] Aleph One Company, "Yet Another Flash FilingSystem," http://www.aleph1.co.uk/yaffs/.

[5] M. Rosenblum, and J.K. Ousterhout, ” The Design and Implementation of a Log-Structured File System,” ACM Transactions on Computer Systems 10(1) (1992)

[6] Samsung Electronics Co., .NAND Flash Memory& SmartMedia Data Book., 2002.

[7] IOZone http://www.iozone.org/