임베디드 모바일 디바이스를 위한 스냅샷 지원 내장형 플래시 메모리 파일 시스템

(A NAND Flash Memory file system support Snapshot for Embedded Mobile Devices)

오 용 석*, 박 찬 익
포항공과대학교 정보통신공학화
(Yong-Seok Oh, Chan-Ik Park)
(Department of GIST Pohang University of Science and technology)

Abstract : In this work, we present a novel snapshot file system for embedded mobile platforms. this file system is specifically designed for NAND flash memory. It is designed for mobile platform such as mobile phone and PDA. Many solutions are developed for mobile devices backup. Our file system used NAND flash memory characteristic especially out-place update for snapshot backup. we also using A LFS(Log File System) Technique for Snapshot. in this paper we present a file system support snapshot and comparable with Yaffs2 NAND flash memory file system

Keywords : NAND Flash, File System, Snapshot, backup

Ⅰ. 서론

낸드 플래시 메모리는 비휘발성, 저전력, 빠른 입출력, 출격에 간함 등과 같은 많은 장점을 가지고 있으며 휴대폰, MP3, PDA 등과 같은 다양한 모바일 기기에 적용 되어 왔으며 그 사용이 지속적으로 증가해 왔다. 또한 모바일 디바이스의 성능이 높아지고 다양한 기능을 제공하며 모바일 기기의 스토리지 용량이 증가함에 따라서 데이터의 백업이 중요한 이슈가 되고 있다. 플래시 메모리는 기존 디스크와 달리 그 특성으로 인하여 한번 데이터를 쓴 페이지는 블록을 지우기 전까지 다시 사용할 수 없으며 또한 읽기 속도는 매우 빠르지만 쓰기 속도와 지움 연산의 속도가 상대적으로 느리다. 한 번에 지울 수 있는 크기가 일정하며 지움 연산의 횟수가 제한적이다. 본 논문에서 제한하는 파일 시스템은 이런 플래시 메모리의 Out-Place-Update 연산의 특징과 LFS(Log Structured File System) 기법을 이용 하여 최소한의 오버헤드를 갖는 Snapshot 지원 파일 시스템을 소개한다. LFS는 모든 쓰기연산에 대하여 Log와 같이 파일 시스템에 순차적으로 기록하는 방식을 사용하며 기존 플래시 메모리 파일 시스템에서 낸드 플래시 메모리의 Out-place 업데이트 연산을 제거하기 위하여 널리 사용되어 왔다. 기존 JFFS2, YAFFS2 와 같은 파일 시스템에서 이 Out-Place 업데이트 연산을 제거하는 방안으로 LFS를 사용한다. 본 논문에서 제안하는 파일 시스템은 이 Out-place 업데이트의 특성을 이용하여 out-place 업데이트를 이용하여 Snapshot 이미지를 생성하고 관리한다.

본 논문에서의 구성은 다음과 같다. 2장에서 배경지식에 대하여 소개하고 3장에서 관련연구에 대하여 기술한다. 4장에서 Snapshot 파일시스템을 소개하며 5장에서 그 성능을 비교한다. 마지막으로 6장에서 결론을 맺는다.

1) 본 연구는 지식경제부 및 정보통신 연구진흥원의 IT R&D 프로그램의 연구 결과로 수행되었음 [2008-S034-01, Development of Collaborative Virtual Machine Technology for SoD (System on-Demand) Service]

Ⅱ. 배경 지식

1. NAND Flash Memory

플래시 메모리의 데이터를 읽고 쓰고 하는 기본적인 단위는 페이지 단위이며 한 페이지의 크기는

512바이트에서 2048바이트 이다. 각 페이지는 해당 페이지에서 발생 할 수 있는 에러를 보정하고 관련 메타 데이터를 보관 할 수 있는 여분의 공간을 가지고 있으며 이것을 Spare영역 또는 OOB(Out Of Band) 영역이라 한다. 플래시 메모리의 기본적인 연산의 단위인 페이지들을 블록ㄷ으로 구성하고 이 블록은 플래시 메모리상에서 관리하는 최소의 단위가 된다. 데이터를 읽고 쓰는 연산은 기본적으로 페이지 단위로 관리가 되며 페이지를 지울 때 즉 데이터를 지우는 연산은 블록을 기본 단위로 하여 이루어진다. 다음 <그림 1>은 플래시 메모리의 기본 연산 수행 시간을 보여준다.

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

그림 1. 낸드 플래시 메모리 연산 수행 시간
Fig. 1. NAND Flash Memory Operation time

Ⅲ. 관련 연구

1. LFS(Log Structured File System)[2]

LFS는 파일 시스템으로 전달되는 모든 쓰기 연산에 대하여 로그 방식과 같이 새로운 영역에 기록하는 방법으로 기존 디스크 기반 방식에서 개발된 파일 시스템이다. 본 논문에서는 LFS 방식을 이용하여 플래시 메모리의 Out-place 업데이트를 파일 단위의 Snapshot 이미지로 기록하는 방법을 사용하여 Snapshot을 기록한다. 로그 기반 파일 시스템의 특성상 업데이트가 되기 전 기존의 데이터는 남겨놓고 새로운 영역에 데이터 쓰기 연산을 수행함으로 플래시 메모리의 out-place 특성과 같은 효과를 나타낸다.

2. 기존 Disk 기반 Snapshot File System

파일 시스템 단위의 Snapshot 기능을 지원하는 파일 시스템은 WAFL[3], VxFS[4], FFS[5] and Ext3cow[6] 등이 있으며 WAFL(Write Anywhere File Layout)은 LFS[7]를 기반으로 개발 되었으며 WAFL의 경우 모든 데이터와 메타데이터를 특정 트리 구조를 이용하여 관리하고 이 트리 구조를 이용하여 파일 시스템 Snapshot 기능을 제공한다. VxFS(Veritas File System)은 Varitas 소프트웨어 사에서 개발된 첫 번째 저널링 파일 시스템이다. VxFS는 블록 수준의 COW(Copy On Write)기법을 이용하여 Snapshot 기능을 지원한다. VxFS의 경우 Snapshot을 위해서 반드시 빈 공간을 할당해야 하는 단점이 있다. FFS(Fast File System) 역시 COW를 이용하여 Snapshot 기능을 제공하며 Snapshot 이미지를 관리하기 위하여 Snapshot 파일이라는 특별한 파일을 이용하여 관리 한다. Ext3Cow는 리눅스 커널 2.4기반으로 개발 되었으며 기존 ext3 파일 시스템을 확장하여 Snapshot 기능을 지원 한다. Ext3Cow의 경우 Ext3의 성능과 비교하여 낮은 오버헤드를 가지며 Snapshot 기능을 제공하고 있다.

Ⅳ. 디자인 및 구현

1. 파일 시스템 디자인

본 논문에서 제안하는 파일 시스템의 구조는 다음과 같다.

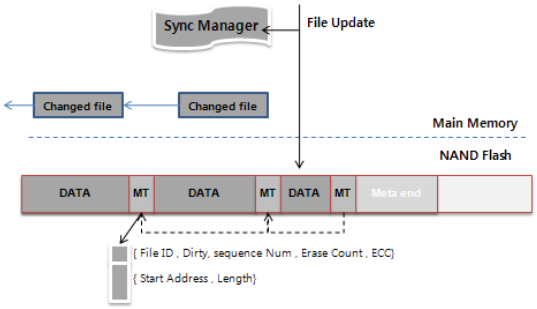

그림 2 제안 파일 시스템 동작 구조
fig. 2. Overview of the proposed method

파일 시스템 이미지 변화를 추적하기 위해서, 각 I/O 동작 마다 어느 파일이 영향을 받고 있는지 확인 할 수 있도록 Filter Function 두고 Syncmanager로 filter function을 통해서 변경된 파일 정보를 확인한다. 본 논문에서 제안하는 파일 시스템상의 파일 이름을 확인할 수 있는 가장 저수준에 위치한다. 본 파일 시스템의 데이터 영역에 저장 되는 파일들을 추적하기 위해서 MT(Meta Management Area)에 각 파일에 대한 log-page를 할당하고 파일 갱신이 일어나게 되면 Syncmanager는 해당 파일을 관리하게 되며 그 방식은 다음과 같다. 해당 파일이 현재 최근 갱신 목록에 있는지 확인하고, 갱신 목록에 존재 하지 않다

면 해당 파일을 갱신 목록에 추가하고 MT상의 해당 log-page를 Snapshot 상태로 변경한다. MT상 log-page에 기록되는 내용은 기존 파일의 맵핑 정보를 기록하며 해당 log-page 그 pare영역에 Snapshot으로 표시한다. 시스템의 Idle 시간 또는 Syncmanager가 관리하는 갱신목록의 크기가 일기록 수준은 넘게 되면 MT영역에 현재 갱신목록에 있는 파일에 관련된 log-page를 할당하고 관련 데이터를 기록한다. 기존 업데이트로 인하여 invalid 되는 페이지는 Snapshot 이미지의 데이터 영역으로 표시하고 Garbage Collection 대ÁMT영쵵거 한다. 본 논문에서 제안하는 파일 시스템은 전체 파일 시스템 단위의 Snapshot 기법이 아닌 파일 단위 Snapshot 기법을 사용하며 다음 <그림 3>은 파일 단위 Snapshot 기법의 메타 관리 구조 이다.

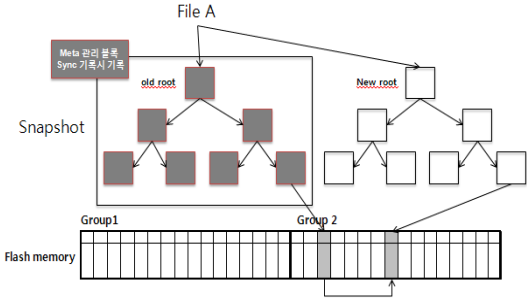

그림 3 메타 관리 구조상 맵핑 기록

fig. 3. Meta Management Block

Snapshot 정보를 관리하기 위해서 Meta 관리 블록에는 기존에 업데이트로 인하여 변경되기 전의 맵핑을 기록하고 후 Snapshot을 검색할 때 이 메타 관리 블록만을 검색하여 Snapshot 파일을 구성한다.

2. 파일 시스템 Snapshot Operation

본 논문에서 제안 하는 Snapshot 지원 파일 시스템의 경우 Snapshot의 기본 단위는 파일 시스템 전체 이미지에 대한 단위가 아닌 파일 단위의 Snapshot 기능을 제공한다. 파일 단위 Snapshot 기능을 지원하기 위해서 별도 operation은 정의 하지 않으며 플래시 메모리의 out-place 업데이트 특징을 이용하여 Snapshot 기능을 제공한다. NAND 플래시 메모리에서 데이터의 업데이트가 일어나면 즉 page에 Write가 일어나면 플래시 메모리는 in-place 업데이트가 불가능하기 때문에 새로운 page를 할당하고 이 페이지에 데이터를 기록하게 된다. 업데이트 되어 dirty 된 페이지를 Snapshot 페이지로 설정하고 이 페이지를 포함하는 맵핑을 메타관리 블록을 이용하여 기록하고 해당 Snapshot을 로딩 할 수 있다. 파일 시스템 단위 Snapshot을 지원하기 위한별도 구조는 다음 절에서 설명한다. 현재 구현된 파일 시스템의 경우는 파일 단위의 Snapshot만을 지원하며 현재 파일 시스템 단위의 Snapshot 지원을 위한 개발이 진행 중에 있다.

기록된 파일 단위의 Snapshot을 읽기 기를 위해서는 초기 응용프로그램은 Snapshot된 데이터를 읽기 위해서 meta관리 블록을 스캔하여 파일 맵핑 트리를 재구성한다. 현재 로딩 되어 있는 맵핑 즉 최신 데이터에 대한 맵핑을 메타 관리 블록에 기록하고 Snapshot 데이터에 대한 맵핑을 로드한다. 응용프로그램과 파일 시스템간의 인터페이스는 Proc 파일 시스템을 이용하여 상호 동작하며 해당 응용프로그램은 Proc 파일 시스템을 이용하여 Snapshot을 읽고 검색할 수 있다.

Ⅴ. 성능 평가

1. 실험 환경

본 논문에서 제안하는 파일 시스템의 성능을 비교하기 위해서 본 논문에서는 YAFFS2 낸드 플래시 메모리 파일 시스템과 비교 한다.

다음 <그림 5>은 실험 하드웨어 환경이다.

Processor	PXA 270
SDRAM	Samsung 128M SDRAM
Nor flash	Intel StrataFlash 64MB
NAND Flash	Samsung 256MB

그림 4 하드웨어 실험 환경

fig. 4. Experimental Environment

실험은 실제 휴대폰과 같은 환경을 구성하기 위하여 Android 모바일 플랫폼 환경을 구성하고 Android 커널 2.6.27 버전의 커널을 이용하여 실험을 수행하였으며 하드웨어에 커널을 포팅 수행한 후 파일 시스템의 성능을 측정하였다. 성능 측적을 위하여 다음과 같이 두 가지 경우의 성능 실험을 수행 하였다.

IOzone benchmark를 통하여 파일 시스템의 성능을 측정하였으며 파일 개수와 사용량에 따른 I/O 성능을 측정하였다.

다음 <그림 5>는 IOzone Benchmark를 통한 I/O 성능을 측정한 결과이다. IOzone을 실험 보드에 포팅을 수행하고 성능을 측정한다. 요청 크기는 4KB의 고정크기이며 파일 크기는 150MB 파일을 대상으로 수행하였다. 제안 하는 파일 시스템의 성능이 더 낳은 성능을 보이는 것을 볼 수 있는데 이

이유는 Garbage Collection을 최소한으로 수행하기 때문이며 그 이유는 Garbage Collection의 대상이 되는 데이터가 Snapshot 데이터로 Garbage Collection 대상에서 제외되기 때문이다.

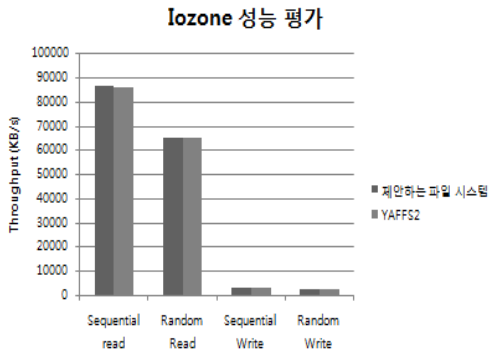

그림 5 IOZone 성능 평가

Fig. 5. IOzone Performance Test

다음 <그림 6>은 제안 하는 파일 시스템의 Write 오버헤드를 알아 보기위한 Write 성능 실험이다.

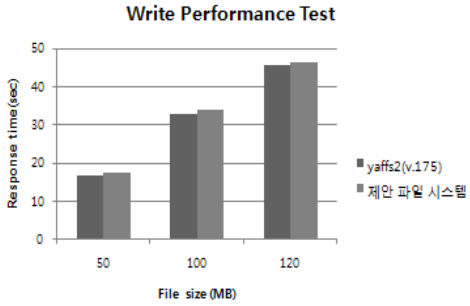

그림 6 파일 크기별 쓰기 성능

Fig. 6. Write test of used file Size

<그림 6> 파일 크기별에 따른 write I/O 성능을 보여 준다. 본 논문에서 실험은 파일을 한번 쓴 후 다시 Over-Write 할 때 성능을 나타낸다. YAFFS2에 비하여 성능이 높아지는 것은 Over-Write로 인한 메타 데이터 관리 블록의 기록 오버헤드 이다.

다음 <그림 7>은 파일의 개수에 따른 write I/O 성능을 보여준다. 앞서 <그림 6> 과같이 Over-Write를 수행하여 성능 평가를 수행하였다.

YAFFS2와 비교하여 성능이 초기 에는 좋고 후에 나빠지는 이유는 메타 관리 블록 오버헤드 이며 초기 성능이 좋은 이유는 Garbage Collection 수행을 하지 않기 때문이며 파일 개수가 늘어나면 스냅샷 개수가 늘어나므로 Garbage collection 수행을 자주 하게 되어 성능이 낮아짐을 볼 수 있다.

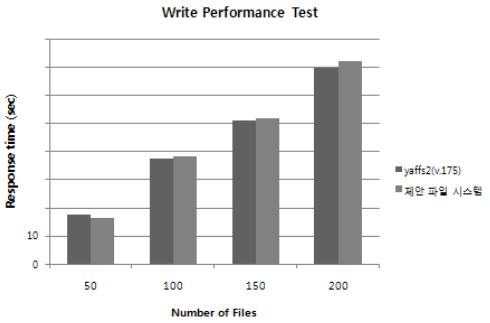

그림 7 파일 개수별 쓰기 성능 측정

Fig. 7. Write test of number of files

Ⅴ. 결론

본 논문은 낸드 플래시 메모리의 특성을 이용하여 Snapshot을 지원하는 파일 시스템을 소개하고 그 성능을 평가 하였다. 플래시 메모리의 특성을 이용하기 때문에 I/O에 큰 영향을 주지 않으며 스냅샷을 수행 할 수 있다. 현재 파일 시스템 단위의 Snapshot 기능에 대한 고려와 플래시 메모리에 따른 공간 효율성을 위한 Snapshot Garbage Collection 기 법등에 대한 연구를 진행 중이다. 본 논문에서 제안한 파일 시스템의 경우 임베디드 모바일 기기의 파일 시스템으로 적용이 가능하며 파일 시스템과 인터페이스를 통한 응용프로그램 작성을 수행 할 수 있도록 구현 하였다.

참고 문헌

[1] D. Hitz, J. Lau, and M. Malcolm, 1994, File SystemDesign for an NFS File Server Applicance.,USENIX Technical Conference

[2] Toby Creek, 2003, VERITAS VxFS, Technical Report TR-3281

[3] M. K. McKusick, et al., 1994, The Design and Implementation of the 4.4 BSD Operating System, Addison Wesley

[4] Z. Peterson and R. Burns, 2005, Ext3cow: A Time-Shifting File System for Regulatory Compliance,
ACM Transactions on Storage.

[5] M. Rosenblum and J. K. Ousterhout, 1991, The Design and Implementation of a Log-Structured File System, Symposium on Operating Systems Principles