Journal of The Korea Society of Computer and Information
Vol. 22 No. 9, pp. 1-8, September 2017

www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.09.001

A File System for Large-scale NAND Flash Memory Based Storage System

Sunghoon Son*

Abstract

In this paper, we propose a file system for flash memory which remedies shortcomings of existing flash memory file systems. Besides supporting large block size, the proposed file system reduces time in initializing file system significantly by adopting logical address comprised of erase block number and bitmap for pages in the block to find a page. The file system is suitable for embedded systems with limited main memory since it has small in-memory data structures. It also provides efficient management of obsolete blocks and free blocks, which contribute to the reduction of file update time. Finally the proposed file system can easily configure the maximum file size and file system size limits, which results in portability to emerging larger flash memories. By conducting performance evaluation studies, we show that the proposed file system can contribute to the performance improvement of embedded systems.

▸Keyword: NAND Flash memory, Flash file system, Embedded system

I. Introduction

최근 수년간 플래시 메모리는 용량이나 접근 속도 면에서 비약적인 발전을 거듭하고 있다. 현재 휴대용 디지털 기기에는 주로 수십 GB의 NAND 플래시 메모리가 저장장치로 사용되고 있으며, 최신의 12 ~ 15nm 수준의 반도체 제작 공정 하에서는 512GB 급의 NAND 플래시 메모리 칩이 개발된 바 있다. 이러한 플래시 메모리의 용량 증가는 플래시 메모리가 사용되는 기기의 종류에도 크게 영향을 미치고 있다. 이제는 단순한 임베디드 장치뿐만 아니라 특히 노트북 등과 같이 이동성이 요구되는 다양한 휴대용 디지털 기기에서 기존의 하드 디스크 대신 플래시 메모리를 기반으로 하는 저장 장치를 보조 기억 장치로 사용하고 있는 실정이다.

NAND 플래시 메모리는 읽고 쓰는 단위와 지우는 단위가 서로 다르고, 덮어쓰기가 불가능하며, 특히 지움 블록(erase block)이라는 단위로만 삭제가 가능하다는 점에서 일반적으로 자기 디스크와는 그 성격이 매우 다르다. 더욱이 블록을 무한히 지울 수 있는 것이 아니라 일정 횟수 이상 삭제가 이루어진 블록은 더 이상 사용할 수 없게 되는 특징을 가지고 있다. 이러한 특징들로 인해 플래시 메모리를 저장 장치로 사용하는 경우 전용 파일시스템의 사용은 필수적이다. 현재 YAFFS, JFFS2 등과 같은 플래시 메모리를 위한 몇몇 전용 파일시스템들이 개발되어서 사용되고 있다. 이 파일시스템들은 플래시 메모리의 특성을 고려하여 개발된 것이기는 하지만, 최근 출시되고 있는 대용량 NAND 플래시 메모리를 고려해 볼 때 몇 가지 성능적인 한계를 가지고 있다. 우선 이들 파일시스템들은 최근 플래시 메모리의 큰 블록 사이즈를 지원하지 못할 뿐만 아니라, 장치의 부팅 시 파일시스템 초기화에 오랜 시간이 걸리고, 메인 메모리 상에서의 구조가 복잡하며, 무효 블록(obsolete block)에 대한 효율적인 관리를 제공하지 못하는 등의 단점을 가지고 있다.

본 논문에서는 이러한 기존 플래시 메모리 파일시스템의 단점을 극복할 수 있는 NAND 플래시 메모리 전용 파일시스템을 제안한다. 본 논문에서 제안하는 파일시스템의 가장 큰 특징은 큰 블록 크기의 플래시 메모리를 지원할 뿐만 아니라, 플래시 메모리 상에서 파일의 데이터가 저장된 페이지를 찾기 위해 특정 지움 블록에 대한 주소와 해당 블록 내의 페이지들에 대한 비트맵(bitmap)으로 구성된 논리적 주소를 사용하는 것이다. 이러한 구조는 플래시 메모리의 용량에 관계없이 장치 부팅 시 파일시스템을 초기화하는 데 걸리는 시간을

• First Author: Sunghoon Son, Corresponding Author: Sunghoon Son
*Sunghoon Son (shson@smu.ac.kr), Dept. of Computer Science, Sangmyung University
• Received: 2017. 08. 22, Revised: 2017. 08. 30, Accepted: 2017. 09. 05.

크게 줄일 수 있을 것으로 판단된다. 또한 메인 메모리상에 유지해야 하는 파일시스템에 대한 정보가 많지 않기 때문에 제한된 크기의 RAM을 가지고 있는 임베디드 장치에도 적합할 것으로 예상된다. 뿐만 아니라 무효 블록에 대한 효율적인 관리가 가능하며, 가용 블록에 대한 적절한 관리를 통해서 파일의 갱신과 데이터의 추가에 걸리는 시간을 크게 줄이게 될 것이다. 마지막으로 제안된 플래시 파일시스템은 지원 가능한 파일의 최대 크기 및 파일시스템의 최대 크기를 쉽게 설정할 수 있어 앞으로 꾸준히 증가하게 될 플래시 메모리의 크기에 대해 뛰어난 이식성을 가질 것으로 판단된다.

성능 평가 결과 제안한 파일시스템은 기존 플래시 파일시스템들에 비해 훨씬 짧고 안정적인 초기화 시간을 제공하고, 우수한 읽기/쓰기 성능을 보이며, 메인 메모리상의 공간 소비도 크지 않은 것으로 나타났다. 본 논문에서 제안하는 플래시 메모리 파일시스템은 새로운 마운트 기법과 데이터 페이지 접근 기법 등을 포함하고 있다. 이러한 기법들은 주로 연결리스트, 비트맵, 매핑 테이블 등을 이용하여 데이터 페이지에 접근하고, 관련 정보를 마운트할 때 사용하던 기존의 플래시 메모리 파일시스템의 경우와는 전혀 다른 접근 방식이라고 할 수 있다. 또한, 짧은 파일시스템 초기화 시간 등의 장점으로 인해 플래시 메모리 사용 기기의 부팅 시간을 단축시킴으로써 플래시 메모리를 채용하고 있는 기존 임베디드 장치에 대한 성능 개선을 가져오게 될 뿐 만 아니라, 큰 블록 크기의 대용량 플래시 메모리의 지원으로 인해 플래시 메모리의 활용 영역을 크게 넓힐 수 있을 것으로 여겨진다.

본 논문은 다음과 같이 구성된다. 우선 2장에서는 플래시 메모리에 대한 동향과 기존 플래시 메모리 파일시스템에 대한 관련 연구를 살펴본다. 3장에서는 본 논문에서 제안한 플래시 메모리 파일시스템을 자세히 소개하고, 4장에서는 제안한 플래시 메모리 파일시스템의 성능 평가 결과를 설명하며, 마지막으로 5장에서 결론을 맺는다.

II. Related Works

NAND 플래시 메모리는 1980년대에 도시바에 의해 처음 발표된 이래, 앞서 개발된 NOR 플래시 메모리에 비해 빠른 쓰기 시간, 높은 집적도, 낮은 제작비용, 높은 내구성 등의 장점으로 다양한 형태로 널리 사용되고 있다[1]. 그러나 저장된 데이터에 대한 순차적 접근만을 지원한다는 점 때문에 범용 컴퓨터 시스템의 저장 장치로 사용되기 보다는 주로 휴대용 디지털 기기의 저장 장치로 주로 사용되어 왔다.

초기의 NAND 플래시 메모리는 그 용량이 너무 작아서 사용이 제한적이었으나, 2005년 도시바와 샌디스크사가 최초의 1GB급 NAND 플래시 메모리 칩을 개발하고 수백 MB 용량의 플래시 메모리 저장장치가 상용화되면서, 본격적으로 다양한 휴대용 디지털 장치들에 사용되기 시작하였다. 이후 2006년 삼성전자가 8GB의 플래시 메모리 칩을 개발하였고, 2008년에는 샌디스크사가 16GB와 32GB의 SDHC Plus card를 출시하였으며, 2012년까지 플래시 드라이브의 용량은 256GB에 이르게 된다. 최신의 2016년 12nm 반도체 제작 공정 하에서 512GB급의 NAND 플래시 메모리 칩이 개발된 바 있으며, 가까운 미래에 1TB급 플래시 메모리도 개발될 것으로 예상된다. NAND 플래시 메모리의 발전은 이를 기반으로 하는 SSD (Solid State Disk)의 개발에도 영향을 미쳐 최근 인텔과 마이크론의 합작으로 3.5TB의 NAND 플래시 스틱과 함께, 10TB의 표준 크기의 EVO의 개발을 발표하였고, 32TB의 2.5인치 SSD도 발표하였다. 더욱이, 삼성전자는 2020년까지 100TB 급의 SSD 제품을 개발할 것으로 예상하고 있다.

이러한 지속적인 용량 증가와 외관의 경박 단소화로 인해 현재 플래시 메모리는 다양한 휴대용 디지털 기기의 저장 장치로 사용 범위가 넓어지고 있다. 최근의 많은 디지털 기기들이 디스크 대신 플래시 메모리를 저장 매체로 사용하고 있으며, 고용량 USB 메모리는 사무실 환경에서 기록 가능한 CD, DVD 등을 대체하고 있다. 휴대용 디지털 장치 외에도 최근에는 기존의 하드 디스크 대신 수백 GB의 대용량 플래시 메모리 디스크인 SSD를 탑재한 휴대용 PC도 출시되고 있으며, 가격 하락으로 수년 내에 SSD가 기존의 하드 디스크를 완전히 대체할 수 있을 것으로 예상된다. 또한 현재 주로 사용되는 휴대용 디지털 장치뿐만 아니라 PC, 디지털 TV, 고용량 저장 장치와 같이 휴대가 어려운 제품에까지 그 활용범위가 넓어질 전망이다.

컴퓨터/휴대용 디지털 기기의 대용량 저장 장치로서의 NAND형 플래시 메모리는 기존의 자기 디스크와는 여러 측면에서 다른 특성을 가지고 있다. 우선 플래시 메모리에서는 데이터 기록 시 기존에 저장된 내용을 덮어 쓸 수가 없다. 즉, 기존에 저장된 내용을 변경하려면 우선 기존 내용이 저장된 블록에 대해 지움 연산을 수행한 후에 비어있는 새 블록에 쓰기 연산을 해야만 한다. 둘째, 플래시 메모리의 각 블록은 지움 연산을 할 수 있는 회수가 제한되어 있다. 일반적으로 각 블록의 지움 연산은 10,000 ~ 100,000회 정도로 제한(보통 10,000회)되는데, 이 제한을 넘어갈 경우 해당 블록은 더 이상 사용할 수 없게 되어 적절한 wear-leveling 기법이 필요하다[2]. 셋째, 지움 연산의 단위와 쓰기, 읽기 연산의 단위가 다르다는 문제를 가지고 있다. 이에 따라 블록 관리에 대한 적절한 알고리즘이 필요하다[3].

이러한 플래시 메모리의 특성으로 인해 그간 NAND 플래시 메모리를 위한 전용 파일시스템에 대한 연구 및 개발이 이루어져 왔다[4]. JFFS2는 대표적인 플래시 메모리 기반 파일시스템이다[5]. Log-structured 파일시스템을 기반으로 하여 설계된 JFFS2는 저널링 기법을 이용하여 신뢰성을 높였다. 하지만 Log-structured 파일시스템을 기반으로 하기 때문에 파일시스템을 사용함에 따라 파일들이 단편화 되고 메모리 공간을 많이 차지하여 빈 영역에 대한 관리가 힘들어지게 된다. 또한 파일의 증가에 따라 파일 관리에 필요한 메모리 공간도 함께 증가하여 메모리를 효율적으로 사용하기 어려워진다[6]. JFFS2는 플래

시 메모리의 각 블록들을 상태에 따라 다양한 리스트들을 통해 유지하고 있으며, 부팅 과정에서 파일시스템 사용을 초기화할 때 이러한 리스트들을 구성하기 위해 모든 블록들을 스캔해야 하기 때문에 장치 부팅에 많은 시간이 소요된다.

YAFFS2 파일시스템은 JFFS2와 함께 널리 사용되는 대표적인 플래시 메모리 전용 파일시스템이다[7]. 최초 버전인 YAFFS는 2002년 Aleph One사에서 개발되었다. 당초 YAFFS는 작은 블록 크기의 NAND 플래시 메모리만을 지원하였으나, 이후 발표된 YAFFS2는 큰 블록 크기를 가지는 현재의 대용량 플래시 메모리를 지원하게 되었다(보통 큰 블록 크기의 플래시 메모리는 한 개의 페이지 크기가 4KB이며, 페이지마다 128B 크기의 스페어(spare) 영역을 가진다. 또한 이러한 페이지가 128개가 모여 하나의 512KB 크기의 지움 블록을 구성하게 된다). YAFFS2는 앞서 언급한 JFFS2보다는 짧은 마운트(mount) 시간을 제공하는 것으로 알려져 있다. YAFFS2는 마운트 시 플래시 메모리의 모든 블록을 스캔하기 보다는 블록의 첫 페이지의 스페어 영역만을 읽음으로써 해당 블록이 데이터가 저장된 유효한 페이지를 포함하고 있는지 판단한다. 이 과정에서 유효 데이터가 있는 블록의 경우에만 해당 블록 내의 모든 페이지의 스페어를 읽음으로써 JFFS2보다는 읽어야 할 정보가 훨씬 적다. 하지만 여전히 모든 블록의 첫 페이지와 데이터가 있는 모든 페이지의 스페어를 읽어야 하는 단점이 있으며, 파일의 일부분을 수정할 때 내용이 변경된 페이지 외에도 파일과 관련된 모든 페이지를 다시 써야 한다는 것은 JFFS2의 경우와 같이 단점으로 작용하고 있다. 메모리상의 구조는 파일의 데이터를 표현하기 위해 트리 형태를 이용함으로써 JFFS2보다는 많이 간단해지고 메모리 소비도 줄이고 있다.

UBIFS 파일시스템(UBI File System)은 JFFS2 파일시스템의 확장성과 관련한 문제들을 해결하고자 하였다[8]. UBIFS는 논리 플래시 블록을 물리 블록으로 변환하는 역할을 하는 UBI 계층 위에서 동작한다. UBI 계층은 자체적으로 웨어레벨링(wear-leveling)과 bad block 관리를 지원하므로 UBIFS는 이러한 문제들로부터 자유로울 수 있다. JFFS2나 YAFFS2가 테이블 구조를 사용하는데 비해 UBIFS는 파일 인덱싱에 트리 구조를 사용한다. 인덱스 트리는 인덱스 노드를 이용해 플래시에 저장되는데, 인덱스 트리의 제일 아랫단 노드는 플래시 데이터나 메타 데이터의 위치를 가리킨다. UBIFS는 파일 데이터를 저장하기 위해 일반적인 데이터 노드도 사용한다. UBIFS는 전체 플래시 볼륨을 몇 개의 영역으로 나누어 사용하는데, 메인 영역에는 데이터 노드와 인덱스 노드가 저장되고, 논리 지움 블록 트리 (logical erase block property tree) 영영에는 블록에 대한 메타데이터가 저장된다. 덮어쓰기를 허용하지 않는 플래시 메모리의 특성에 따라 트리 노드를 갱신하려면 전체 부모와 조상 노드가 다른 위치로 이동해야 한다. 이러한 이유로 UBIFS의 트리를 wandering tree라고 부른다. UBIFS는 플래시 메모리 접근 횟수를 줄이기 위해서 write-back 캐시를 지원한다. 즉, 파일 데이터와 메타데이터에 대한 수정은 바로 플래시 메모리에 반영되는 대신 메인메모리에 버퍼링되었다가 일정한 주기로 플래시 메모리로 플러시 된다. 파일시스템과 관련된 모든 수정은 파일시스템 일관성 유지를 위해 로깅된다. UBIFS는 Lzo와 Zlib 압축 알고리즘을 사용한다. 트리 구조의 사용으로 인해 UBIFS 계층의 성능은 플래시 파티션 크기에 로그적으로 비례한다.

F2FS (Flash-Friendly File System) 파일시스템은 삼성전자에서 개발된 리눅스 커널을 위한 플래시 파일 시스템이다[9]. F2FS는 처음부터 SSD, eMMC, SD 카드 등 NAND 플래시 메모리 기반 저장 장치의 특성을 고려하여 설계되었다. F2FS는 log-structured 파일시스템을 기반으로 하되, wandering tree에서의 스노우볼 효과 (snowball effect)나 클리닝 오버헤드의 증가 등과 같은 기존 log-structured 기반 파일시스템들의 단점들을 극복하고자 하였다. 더욱이 NAND 기반 저장 장치들이 내부 사양이나 FTL(Flash Translation Layer)과 같은 하부의 플래시 메모리 관리 기법에 따라 다른 성능을 보임에 따라, 저장 장치 상의 레이아웃, 블록 할당 정책, 클리닝 정책 등을 파라미터로 설정할 수 있도록 하여, 사용 환경에 따라 파일시스템을 적절히 구성할 수 있도록 하였다.

III. The Proposed Flash Memory File System

본 장에서는 기존 플래시 메모리 파일시스템의 한계를 극복하기 위한 새로운 플래시 메모리 파일시스템을 소개한다. 제안하는 플래시 메모리 파일시스템은 기존 파일시스템들을 벤치마킹하여 이들에 비해 빠른 초기화 시간 제공, 큰 블록 대용량 플래시 메모리 지원, 메인 메모리 요구량 최소화, 효율적인 데이터의 추가 및 갱신 등을 목표로 한다[10,11,12,13]. 본 장의 마지막에서는 제안된 플래시 메모리 파일시스템의 예상되는 성능을 YAFFS2, JFFS2, UBIFS 등의 플래시 파일시스템과 비교한다.

1. On-flash layout and root directory structure

파일시스템의 첫 지움 블록은 전체 시스템에 대한 각종 정보들과 루트 (root) 디렉터리를 포함한다. 이 루트 디렉터리를 시작으로 하여 디렉터리와 파일들이 계층적으로 저장되게 된다. 시스템의 모든 디렉터리와 파일들의 구조는 이 루트 디렉터리를 통해서 메인 메모리에 재구성 되며, 이 과정에서 JFFS2나 YAFFS2와는 달리 파일시스템 전체를 스캔하여 메모리 자료구조를 구성하지 않아도 된다.

루트 디렉터리에는 일반적인 디렉터리나 파일 뿐만 아니라 Free File이라는 특수 파일을 포함하게 되며 이 파일은 빈 공간을 관리하기 위해 사용된다. 루트 디렉터리는 앞서 언급한 고정된 위치(시스템의 첫 지움 블록)에 위치하며, 그 외의 다른 파일, 디렉터리, 데이터, 메타 데이터들은 모두 임의의 위치에 있을 수 있다.

2. File structure

Fig. 1은 제안한 파일시스템에서 하나의 파일이 저장되는 방식을 보여주고 있다.

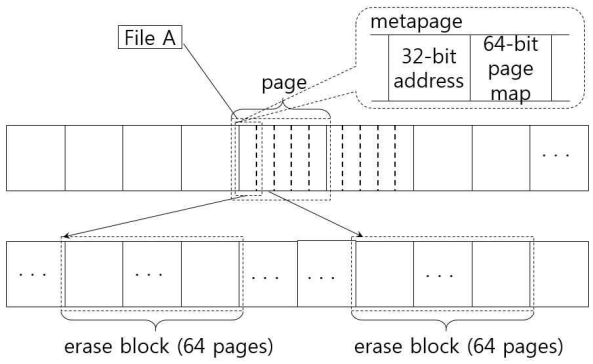

Fig. 1. File structure

디렉터리에 저장된 File A에 대한 포인터는 이 파일의 메타 페이지(Meta page)가 저장된 페이지를 가리키고 있다. 이 메타 페이지를 통해서 해당 파일의 실제 데이터가 존재하는 지움 블록과 포함된 페이지에 대한 물리적 주소를 알아 낼 수 있는 것이다. 메타 페이지는 순차적으로 저장되며 갱신될 때마다 갱신된 페이지 이후의 페이지들이 모두 다시 기록되게 된다. 한 지움 블록이 모두 사용된다면 다른 지움 블록에 다시 저장된다.

3. Meta page and page map

메타 페이지는 32비트의 블록주소와 64비트 크기의 페이지 맵(Page map)으로 구성되며, 이를 통해서 데이터가 저장되어 있는 페이지의 물리 주소를 알아낼 수 있다(Fig. 2). 여기서 블록 주소는 데이터가 저장되어 있는 페이지가 속한 지움 블록의 물리 주소이며, 페이지 맵은 그 지움 블록 내의 유효한 페이지들에 대한 맵이다. 이 두 필드를 합친 하나의 가상 블록 번호는 한 페이지에 최대 170개가 저장될 수 있으며, 이는 21.25MB의 데이터 블록을 표현할 수 있는 크기이다. 따라서 1GB 정도의 큰 파일의 경우 49개의 페이지가 있으면 표현이 가능한데, 지움 블록 하나에 64개의 페이지를 가지고 있으므로 충분히 표현이 가능하다. 모든 주소와 페이지들은 순차적으로 저장되며 임의의 위치에 저장될 수 있다.

Fig. 2. Meta page and page map

하지만 이런 메커니즘에 의해 파일을 저장할 경우 크기가 작은 파일의 경우 실제 데이터보다 더 큰 메타 페이지를 가지게 되어 오버헤드로 작용한다. 이 문제를 해결하기 위해서 126KB를 넘는 파일의 경우 위의 방법으로 파일을 표현하게 되며, 126KB 보다 작은 경우에는 Fig. 1의 File A가 가리키는 페이지가 메타 페이지가 아닌 데이터가 저장되어 있는 지움 블록이 된다. 이 때 파일이 가리키고 있는 지움 블록의 첫 페이지의 스페어 영역에는 해당 지움 블록에 대한 페이지 맵을 저장한다.

파일을 저장하는데 있어서 기존의 JFFS2나 YAFFS2의 경우 스페어 영역 외에는 파일 자체에 대한 추가 공간을 필요로 하지 않았다. 하지만 제안된 파일시스템의 경우에는 메타 페이지를 위한 추가 공간이 필요하다는 단점이 있다. 이 추가 공간을 최대한 줄이고자 앞서 언급한 바와 같이 126KB를 기준으로 다른 데이터 저장 방법을 도입하였다. 126KB 이하 크기의 파일의 경우 데이터가 있는 지움 블록의 첫 페이지의 스페어 영역에 비트맵을 저장하므로 추가 오버헤드가 없고 단지 126KB를 초과하는 파일의 경우에는 21.25MB 당 메타 페이지를 위해 2KB의 오버헤드가 존재한다. Table 1은 제안한 파일시스템에서 파일 크기에 따른 저장 공간 오버헤드를 정리한 것이다.

Table 1. Space overhead for meta page

File size	No. of files	Storage overhead
10KB	1	-
10KB	1024	-
20MB	1	2KB
20MB	1024	2MB
1GB	1	98KB

4. Block reclamation and wear-leveling

기존 JFFS2의 경우에는 웨어레벨링을 보장하기 위해서 dirty list를 이용하였다[5]. YAFFS2는 별도의 웨어레벨링 메커니즘 없이 단순히 랜덤하게 블록을 할당하여 자연스럽게 웨어레벨링이 이루어지도록 하고 있다. 제안된 플래시 메모리 파일시스템에서는 유휴 시간과 Free File의 공간이 일정 수준 이하로 낮아지면 블록 회수가 일어나게 된다. 회수의 대상은 무효(obsolete)로 마크된 메타 페이지와 데이터가 저장되어 있는 페이지이다. 메타 페이지의 경우 페이지들이 지움 블록 내에 순차적으로 저장되어야하기 때문에 저장되어 있는 지움 블록 내에 빈 공간이 없을 때 회수가 일어나게 된다. 무효로 처리된 페이지들을 찾는 방법은 Free File의 페이지와 유효한 페이지들이 아닌 페이지는 모두 무효 페이지로 간주하는 것이다.

파일이나 디렉터리를 위해 지움 블록을 할당하고자 할 때에는 지움 블록의 지움 횟수(erase count)를 보고 가장 낮은 지움 횟수를 가지는 지움 블록을 할당한다. 이 지움 횟수는 플래시 메모리의 각 지움 블록마다 존재하는 스페어 영역에 저장되며 초기화 과정에서 다른 정보들과 함께 메모리로 읽어 온다.

JFFS2나 YAFFS2의 경우에는 파일이 수정되면 파일의 모든 페이지를 다시 기록해야 한다. 회수 역시 파일의 수정 결과를 가져오므로 파일 내의 모든 페이지를 다시 기록한다고 봐야 한다. 하지만 제안된 파일시스템의 경우에는 수정된 데이터가 있는 지움 블록과 inode, 메타 페이지만 수정하면 된다. 예를 들어 100MB 크기의 파일에서 한 페이지를 수정하는 경우 JFFS2나 YAFFS2는 100MB의 파일을 전체를 다시 기록해야 한다.

반면 제안된 파일시스템은 100MB를 저장하는데 필요한 5개의 메타 페이지를 위한 10KB, inode를 위한 약 2KB, 그리고 갱신된 데이터가 포함된 지움 블록 168KB를 다시 기록하여 총 180KB만 다시 기록하면 된다. 이와 같이 파일 수정에 있어서 상당한 시간을 절약할 수 있을 것으로 판단된다. 파일의 삭제 연산은 파일의 메타 페이지를 지움으로써 이루어진다. 실제 데이터가 저장된 페이지들에 대한 삭제 연산은 파일 삭제와 동시에 일어나지 않으며 결과적으로 파일 삭제에 따른 속도 저하를 줄일 수 있다.

5. Mount/unmount

파일시스템을 마운트할 때 파일시스템의 구성 정보를 얻기 위해서 전체 파일시스템을 검색하는 JFFS2나 YAFFS2와는 달리, 제안된 파일시스템에서는 메타 페이지만 검색하여 메모리 상에 파일시스템 정보를 구성할 수가 있다. 이를 위해 루트 디렉터리를 통해서 순차적으로 각 파일 별로 메타 페이지를 읽는 방법을 사용한다.

마운트할 때 읽어 들인 메타 정보를 토대로 Fig. 3과 같이 디렉터리와 파일들이 계층적으로 구성된다. 이는 MFFS와 유사한 모습을 보이지만, MFFS에서는 한 디렉터리 내의 파일 및 디렉터리들이 모두 리스트로 연결되어 있으며 상위 디렉터리에 대한 포인터를 가지고 있지 않는 문제가 있다. 하지만 제안된 파일시스템의 경우에는 한 디렉터리 내에 파일과 디렉터리들이 연결된 리스트가 아닌 주소들을 저장해 놓는 방식으로 사용되고 있기 때문에 리스트 탐색시간을 줄일 수 있다. 또한, 상위 디렉터리에 대한 포인터를 유지 하여 스택 등의 별도의 자료구조 없이도 바로 자신의 상위 디렉터리로 이동할 수 있도록 하였다.

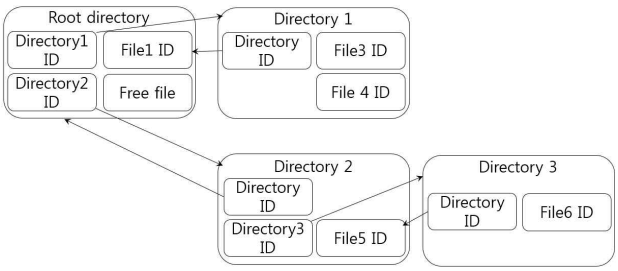

Fig. 3. Meta page and page map

읽어 들인 메타 페이지의 페이지 비트맵(Page bitmap)은 유효한 페이지를 나타내는 역할도 하게 된다. 데이터가 페이지에 저장되고 난 후에 메타 페이지를 기록하게 되어 자연히 블록주소와 페이지 비트맵이 가리키는 곳의 페이지는 유효한 페이지로 간주하게 된다.

빠른 마운트를 위해서는 유효한 페이지를 확인하는데 걸리는 시간을 단축시킬 필요가 있다. 제안된 파일시스템의 경우 유효한 페이지를 확인하기 위해서 21.25MB의 파일 당 2KB의 비트맵을 읽어야 한다. 이 비트맵이 유효성을 보장해주는 역할을 하고 있기 때문이다. 반면에 YAFFS2의 경우에는 21.25MB단위로 환산하여 보면 21.25MB당 680KB를 읽어야 해당 파일이 있는 페이지들의 유효성을 확인할 수 있다. 특히 YAFFS2의 경우에 파일 당 읽어야 하는 용량으로 비교하였지만 사실 파일 단위로 비교 할 수는 없고 전체 파일시스템에서 21.25MB가 유효함을 확인하기 위해 680KB를 읽어야 한다고 보는 것이 더 올바르다. 즉, YAFFS2에 비해 제안된 파일시스템이 페이지의 유효성을 확인하는 데에 많은 이득이 있는 것으로 볼 수 있다.

언마운트 시에는 파일시스템의 동작 중에 남은 작업을 수행하고 현재 상태를 기록한 후 종료하게 된다. 남은 작업은 메모리구조와 파일시스템 간에 미쳐 동기화 시키지 못한 페이지 갱신 내용을 말한다. 무효 페이지의 경우에는 굳이 이 단계에서 삭제하고 끝낼 필요는 없다. 언마운트의 경우 기존의 JFFS2나 YAFFS2와 비교하여 유사한 성능을 보일 것으로 예상된다. 다만 무효 상태의 파일에 대한 추가 정보를 저장할 필요가 없기 때문에 언마운트에 걸리는 시간은 무효 페이지의 양에 따라 차이가 날것으로 보인다.

6. Free file

루트 디렉터리에는 일반 응용프로그램이 사용할 수 없는 Free File이 존재한다. 이 파일은 사용자가 직접 사용하지 못할 뿐만 아니라 보이지 않아 존재여부를 알 수 없다. 이 파일은 삭제되어 다시 사용할 수 있는 영역들을 파일로 관리하기 위해 존재한다. 일반 파일과 유사한 구조를 가지고 있어 별다른 자료구조를 가지고 있을 필요가 없으며 그 때문에 메모리 구조가 단순화 될 수 있는 장점을 가진다. 무효 페이지가 블록 회수로 인해 지워지면 Free File에 새로운 데이터가 추가된 것처럼 빈 지움 블록에 대한 주소를 가지고 메타 페이지를 갱신한다.

전체를 검색하여 빈 영역을 찾아내는 JFFS2나 YAFFS2와는 달리 제안된 파일시스템의 경우에는 Free File의 메타 페이지만 읽으면 빈 영역을 찾을 수 있다. 예를 들어, 1GB의 메모리가 모두 비어 있다고 할 때 기존의 파일시스템의 경우에 모든 지움 블록을 검색하여야 하지만, 제안된 파일시스템의 경우에는 최대 1개의 지움 블록만 읽으면 비어 있는 영역을 알아 낼 수 있다. 즉, 메모리상에 free list를 구성하는데 있어서도 제안된 파일시스템의 경우가 좋은 성능을 보일 것으로 판단된다.

7. Miscellaneous

플래시 메모리의 크기가 증가하는 과정에서는 보통 페이지 크기나 지움 블록 당 페이지의 수와 같은 하드웨어적인 사양의 변화가 동반된다. 제안된 파일시스템의 경우에는 이런 변화에 따라 파일시스템이 동작하지 않는 사태를 예방할 수 있도록 페이지의 크기와 블록 당 페이지의 수 등을 조절할 수 있는 옵션을 마련하였다. 페이지 크기가 변경되면 한 메타 페이지 당 표현할 수 있는 지움 블록의 수를 새로 계산하고 메타 페이지에 저장되는 주소의 크기는 전체 파일시스템의 크기에 따라 파일시스템 생성 시에 동적으로 변화하게 된다. 따라서 한 파일의 크기, 한 파일시스템 내에서의 총 파일 개수, 파일시스템의 전

체 크기 등은 제안된 파일시스템에서는 제한되지 않는다.

운영체제의 경우 파일들의 특성이 다른 응용프로그램과는 조금 다르다. 운영체제의 동작 중 수 많은 작은 크기의 파일들(주로 시스템 파일들)이 수시로 읽히고 기록된다. 이 때 기존의 YAFFS2나 JFFS2 같은 파일시스템에 비해 파일의 갱신과 빠른 마운트 시간 등의 이점이 많기 때문에 적합한 파일시스템으로 사용될 수 있을 것으로 보인다. 실제로 데이터가 저장되어 있는 페이지에 대한 물리적 주소를 알아내는데 걸리는 메모리 상에서의 단계는 Table 2에서 확인할 수 있다.

Table 2. Steps to reach physical address of a page

File system	The worst case	Steps for 1GB file
The proposed file system	2 level	2
JFFS2	Fully fragmented	5,162,690
YAFFS2	-	9

8. Differentiation from existing file systems

최근의 JFFS2에 대한 개선 방향을 보면 Reiser4 파일시스템과 유사한 인덱스 구조를 가지고 있는데, 특이한 점은 스페어 영역을 이용하여 페이지를 관리한다는 것이다[14]. 이 스페어 영역에서 인덱스 노드와 리프 노드를 가지는 지움 블록이 각각 따로 관리되며, garbage collection이 일어나는 동안에도 한 개의 페이지만 이동하는 현재의 JFFS2와는 달리 인덱스 노드와 리프 노드가 모두 다른 지움 블록으로 옮겨지게 된다. 물론 이러한 동작 자체가 오버헤드가 될 수 있기는 하지만, 인덱스 노드 구조 덕분에 마운트 시 모든 노드들을 스캔하던 현재의 JFFS2보다는 훨씬 빠르게 마운트 할 수 있게 될 전망이다. 그러나 파일의 일부 페이지가 갱신되었을 때 관련된 노드들이 모두 갱신되어야 하는 문제는 여전히 존재한다. 이에 비해 제안된 파일시스템의 경우 파일과 관련된 노드들이 모두 갱신되어야 하는 문제점에 있어서 좀 더 효율적인 성능을 보여줄 것이다.

최신의 JFFS2도 wandering tree 구조의 특징을 가지고 있기 때문에 한 파일의 일부 데이터 수정 시 관련된 트리 전체가 다시 기록되어야 한다. 반면 제안된 파일시스템에서는 수정된 데이터가 있는 페이지를 위해 일부 메타 페이지와 해당 지움 블록만 수정하면 되는데 그 수가 JFFS2에 비해 작다는 점이 중요하다. 또한, JFFS2도 포인터로 연결되어 있는 트리 구조를 이용해서 실제 파일이 저장되어 있는 페이지를 접근하도록 하고 있다. 반면에 제시된 파일시스템의 경우에는 앞서 언급한 대로 지움 블록번호와 지움 블록에 대한 비트맵을 혼용하여 하나의 논리적 번호로 사용하는 구조를 가지고 있어서 파일의 데이터를 접근하기 위한 방법 면에서 차별성이 있다고 볼 수 있다.

제안된 파일시스템은 YAFFS2와 비교해서도 여러 장점을 가지고 있다. 플래시 메모리에서 정보를 읽어 와서 메모리에 파일시스템 관련 구조를 구성하는 작업은 전체 마운트 시간 중 많은 부분을 차지한다. 제안된 파일시스템에서는 마운트 시 메타 페이지만을 읽게 되는데, 이 때 플래시 메모리에서 읽어야 하는 데이터의 양이 21.25MB 당 약 678KB 정도 YAFFS2보다 줄어들게 된다. 이렇게 초기에 플래시 메모리에서 읽어야 할 크기가 현저하게 줄어들게 됨으로써, 결국 마운트에 걸리는 전체 시간을 줄이는데 많은 기여할 수 있다고 하겠다.

파일의 일부분을 수정할 때 내용이 변경된 페이지 외에도 파일과 관련된 모든 페이지를 다시 써야 한다는 점은 역시 동일한 단점으로 남아있다. 다만 이 부분에 대해서도 제안된 파일시스템의 경우 수정되어야 할 페이지 수가 작다는 것을 알 수 있다. YAFFS2의 경우 wandering tree 구조의 특징을 가지고 있기 때문에 한 파일의 일부 데이터가 수정된다면 그 데이터를 표현하기 위해 구성된 트리 전체가 다시 쓰여야 한다. 이런 구조는 파일시스템에게 많은 부담을 줄 수 있는 요소이다. 제안된 파일시스템에서는 수정된 데이터가 있는 페이지를 위해 일부의 메타 페이지와 해당 지움 블록만 수정하면 되는데, 그 수가 YAFFS2에 비해 적다는 점이 중요하다. 또한, 포인터로 연결되어 있는 트리 구조를 가지고 실제 파일이 저장되어 있는 페이지를 접근 할 수 있도록 하는 구조는 많은 다른 파일시스템에서도 사용하고 있는 방법이다. 반면에 제시된 파일시스템의 경우에는 지움 블록 번호와 지움 블록에 대한 비트맵을 혼용하여 하나의 논리적 번호로 사용하는 구조를 가지고 있어서 파일의 데이터를 접근하기 위한 방법 면에서 독창적이라고 볼 수 있다.

제안된 파일시스템은 UBIFS와 비교해서도 여러 장점을 가지고 있다. UBIFS도 제안된 파일시스템과 같이 플래시 메모리 크기에 관계없이 일정한 마운트 시간을 가지는 것이 장점이다. 그러나 다음 장에서도 밝힌 바와 같이 실제 마운트에 걸리는 시간에 있어서 제안된 파일시스템이 UBIFS에 비해 훨씬 짧다. 또한 동작 중에 필요한 메인 메모리 사용량이 UBIFS에 비해 월등히 작은 것으로 나타나 제안된 메모리를 가지는 모바일 장치 등에서 훨씬 유리할 것으로 예상된다. 파일 읽기/쓰기의 성능은 제안된 파일시스템과 UBIFS 간에 거의 유사한 것으로 판단된다. 다만 UBIFS는 자체적인 write-back 캐시를 가지고 있어 특히 쓰기 연산의 성능이 더 좋은 것으로 나타나지만, 이로 인해 때때로 쓰기 시의 레이턴시가 급격히 늘어나기도 한다.

IV. Experiments

이 장에서는 제안한 플래시 파일시스템의 프로토타입을 대상으로 한 성능 평가 결과를 소개한다. 프로토타입의 구현 및 실험은 ARM920T 프로세서 기반의 S3C2440 AP를 사용하는 개발 보드 상에서 이루어졌다. S3C2440 개발 보드에서 사용된 플래시 메모리는 512MB로, 페이지 크기는 2KB, 블록 크기는 128KB이다. 플래시 메모리의 기본 성능은 페이지 당 읽기 시간은 25ns, 쓰기 시간은 200ns이며, 블록 하나를 삭제하는데 걸리는 시간은 2ms이다. 마운트 시간, 메인메모리 소비, 입출

력 성능 등에 대해 평가를 수행했으며, 기존 플래시 파일시스템 중 JFFS2, YAFFS2, UBIFS를 비교 대상으로 하였다.

우선 제안된 파일시스템의 마운트 시간을 측정하였다. 마운트 시간은 시스템의 부트 시간에 결정적인 영향을 미치는 요소이다. 정확한 비교를 위해 파일시스템 공간이 각각 20% 사용된 경우, 50% 사용된 경우, 80% 사용된 경우 등 세 가지 플래시 메모리 사용 유형에 대해 마운트에 걸리는 시간을 측정하였다.

Table 3에서 보는 바와 같이 제안된 파일시스템은 마운트 시간이 비교적 짧은 것으로 나타났다. 특히 마운트에 걸리는 시간은 사용 중인 저장 공간의 크기와 무관한 것으로 나타났다. 이러한 특징은 UBIFS의 경우도 동일하다. 반면 YAFFS2는 마운트에 걸리는 시간이 저장된 데이터의 크기에 비례하여 증가하는 것으로 나타났고, JFFS2는 저장된 파일이 많을수록 마운트 시 이를 스캔하는데 많은 시간이 걸리는 것으로 나타났다.

Table 3. Mount time (sec)

File system	20%	50%	80%
The proposed file system	0.38	0.4	0.41
JFFS2	43	94	122
YAFFS2	0.67	0.69	0.7
UBIFS	0.95	0.95	0.96

다음은 제안한 파일시스템의 읽기/쓰기 성능을 기존 파일시스템들과 비교하였다. 읽기/쓰기 성능의 평가를 위해 파일시스템 성능 평가 도구인 tiobench를 사용하였다[15]. Tiobench 실행 시 한 개의 쓰레드가 256MB의 데이터에 대한 순차접근과 128MB 크기의 데이터를 임의 접근을 하도록 설정한 후 실험하였다. 이 때 사용된 블록 크기는 128KB이다.

Fig. 4는 파일시스템의 읽기/쓰기 성능을 다른 파일시스템들과 비교한 결과이다. 그림에서 보는 바와 같이 제안된 파일시스템과 UBIFS가 가장 좋은 읽기/쓰기 성능을 보이고 있다. UBIFS가 제안된 파일시스템과 유사한 성능을 가지는 원인은 UBIFS가 write-back 캐시를 지원하기 때문으로 추정된다.

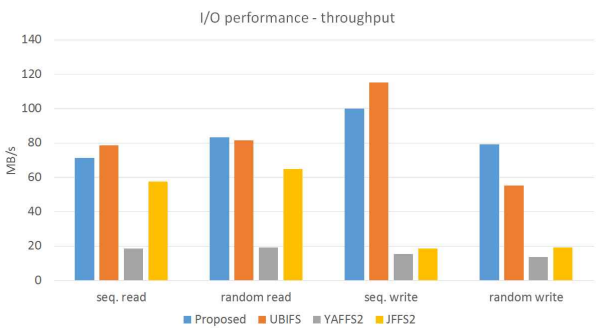

Fig. 4. Read/write throughput

Fig. 5는 제안된 파일시스템의 읽기/쓰기 레이턴시의 평균값을 다른 파일시스템들과 비교한 결과이다. 그림에서 보는 바와 같이 제안된 파일시스템과 UBIFS가 평균적인 케이스에서 낮은 읽기/쓰기 레이턴시를 나타내고 있다. 한편 그림의 평균값에는 나타나지 않으나 UBIFS는 간헐적으로 쓰기 시의 레이턴시가 급격히 늘어나는 현상을 보이는데, 이는 캐시된 데이터를 플러시 하는 동작 때문인 것으로 여겨진다.

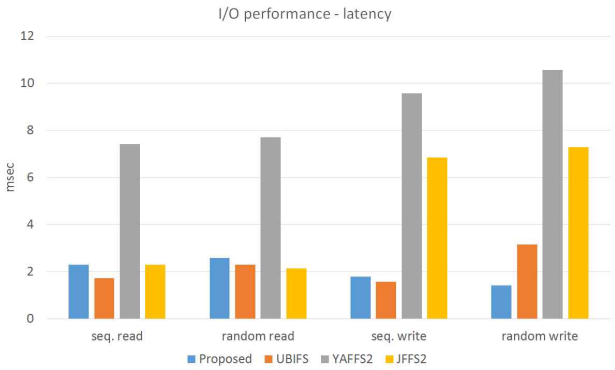

Fig. 5. Average read/write latency

다음은 제안한 파일시스템의 메인 메모리 사용을 다른 파일시스템의 경우와 비교하였다. 플래시 메모리에 저장된 파일이 메모리 사용에 얼마나 영향을 끼치는지도 함께 알아보기 위해 앞서 Table 3의 마운트 시간 측정 시와 동일한 세 가지 플래시 메모리 사용 유형에 대해 매인 메모리 사용을 측정하였다.

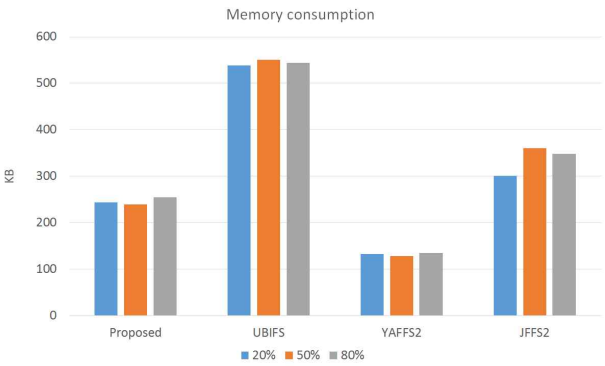

Fig. 6. Main memory consumption

Fig. 6에서 보는 바와 같이 메모리 사용이 가장 적은 파일시스템은 YAFFS2로 나타났다. 그 다음으로는 제안된 파일시스템, JFFS2, UBIFS 순으로 메모리 사용이 늘어나는 것으로 나타났다. 한편 대부분의 파일시스템에서 파일 사용 패턴은 메인 메모리 사용에 거의 영향을 끼치지 않는 것으로 보인다.

V. Conclusions

앞으로의 플래시 메모리의 용량 등을 고려해 볼 때 현재의

플래시 메모리 파일시스템들은 기능적인 측면이나 성능적인 측면에서 한계를 지니고 있다. 본 논문에서는 대용량 플래시 메모리를 지원할 뿐만 아니라, 짧은 마운트 시간을 제공하며, RAM 소비가 크지 않은 새로운 플래시 메모리 파일시스템을 제안하였다. 본 논문에서 사용한 짧은 마운트 시간을 보장하는 기법은 플래시 메모리에 저장되는 데이터에 대한 관리 기법에 기초하고 있다. 즉, 데이터를 접근하는 과정에서 지움 블록번호와 지움 블록 내의 페이지들에 대한 비트맵으로 구성된 논리적인 번호를 통해서 특정 데이터를 접근하게 되는데, 이러한 방법은 기존의 플래시 메모리 기반 파일시스템에서는 찾아볼 수 없는 독창적인 것이다. 빠른 마운트는 플래시 메모리를 사용하는 기기의 사용자에게 있어서 매우 중요한 요소이다. 이러한 연구는 결국 플래시 메모리를 사용하는 기기에 대한 부팅시간을 단축시켜 줄 것이며 좀 더 사용이 간편한 휴대용 디지털 장치의 개발에 도움이 될 것이다.

REFERENCES

[1] C. M. Compagnoni, A. Goda, A. S. Spinelli, P. Feeley, A. L. Lacaita, and A. Visconti, "Reviewing the evolution of the NAND Flash technology," Proceedings of IEEE, Vol. 105, Issue 9, pp. 1609-1633, September 2017.

[2] S. Kim and J. Kwak, "EPET-WL: Enhanced Prediction and Elapsed Time-based Wear Leveling Technique for NAND Flash Memory in Portable Devices," Journal of the Korea Society of Computer and Information, Vol. 21, No. 5, pp.1-10, May 2016.

[3] H. Choi and Y. Kim, "An Efficient Cache Management Scheme of Flash Translation Layer for Large Size Flash Memory Drives," Journal of the Korea Society of Computer and Information, Vol. 20, no.11, pp.31-38, November 2015.

[4] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," ACM Computing Surveys, Vol. 37, No. 2, pp. 138-163, June 2005.

[5] D. Woodhouse, JFFS: The Journaling Flash File System, Proceedings of Ottawa Linux Symposium, Ottawa, Canada, July 2001.

[6] Luca Boschetti, Software Profile: Journaling Flash File System, Version 2 (JFFS2), Micron Technology, Inc., March 2011.

[7] How YAFFS works, *http://www.dubeiko.com /development/FileSystems/YAFFS/HowYaffsWorks.pdf*.

[8] A brief introduction to the design of UBIFS, *http://www.linux-mtd.infradead.org/doc/ubifs_whitepaper.pdf*.

[9] C. Lee, D. Sim, J. Hwang, and S. Cho, F2FS: A New File System for Flash Storage, Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15), pp. 273-286, Santa Clara CA, USA, February 2015.

[10] K. Yim, J. Kim, and K. Koh, A fast start-up technique for flash memory based computing systems, Proceedings of the 2005 ACM symposium on Applied computing, p.843-849, Santa Fe, New Mexico, USA, March 2005.

[11] C. Wu, T. Kuo, and L. Chang, Efficient initialization and crash recovery for log-based file systems over flash memory, Proceedings of the 2006 ACM symposium on Applied computing SAC '06, p.896-900, Dijon, France, April 2006.

[12] J. Xu and S. Swanson, NOVA: A log-structured file system for hybrid volatile/non-volatile main memories, Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16), pp. 323-338, Santa Clara CA, USA, February 2016.

[13] P. Sehgal, S. Basu, K. Srinivasan, and K. Voruganti, An empirical study of file systems on NVM, Proceedings of the 2015 IEEE Symposium on Mass Storage Systems and Technologies (MSST'15), Santa Clara CA, USA, May 2015.

[14] JFFS3 design issues, *http://www.linux-mtd.infradead.org/tech/JFFS3design.pdf*.

[15] Tiobench, *https://sourceforge.net/projects/tiobench/*.

Authors

Sunghoon Son received his B.S., M.S. and Ph.D. degrees in Computer Science from Seoul National University, Korea, in 1991, 1993 and 1999, respectively. Dr. Son joined the faculty of the Department of Computer Science at Sangmyung University, Seoul, Korea, in 2004. He is currently a Professor in the Department of Computer Science, Sangmyung University. He is interested in system software, embedded system, and virtualization.