플래시 메모리 파일 시스템을 위한 가비지 콜렉터 설계 및 구현

김 기 영[†] · 손 성 훈[††] · 신 동 하[†††]

요 약

최근 들어 많은 임베디드 기기들이 휴대성과 성능 향상을 위해 플래시 메모리를 저장 매체로 사용하고 있다. 플래시 메모리는 일반적인 디스크와는 다른 특성과 제약 조건으로 인해 파일 시스템 설계에 있어서 여러 가지가 고려되어야 하며, 디스크와 다르게 덮어 쓰기가 불가능하다. 플래시 메모리 파일 시스템은 LFS(Log-structure File System)의 형태를 가지며, 따라서 가비지 콜렉터를 사용한다. 블록을 재사용하기 위해서는 가비지 콜렉터의 역할이 크며, 가비지 콜렉터는 파일 시스템의 성능에 직접적으로 영향을 주기 때문에 플래시 메모리의 특성을 고려하여 설계해야 한다. 이에 본 논문에서는 JFFS2(Journaling Flash File System II)의 가비지 콜렉터를 개선한 플래시 메모리 파일 시스템을 제시하고, 이를 임베디드 시스템 실험 보드에서 테스트하였다. 그 결과 기존의 파일 시스템에 비해 메모리 사용률을 감소시킬 수 있었으며, 이로 인한 플래시 메모리 수명 연장, 쓰기 평준화(wear-leveling) 개선 등의 성능 향상이 나타남을 확인할 수 있었다.

키워드 : 플래시 메모리, JFFS2, 파일 시스템, 가비지 콜렉터, 쓰기 평준화 기법, 저전력 시스템

A garbage collector design and implementation for flash memory file system

Ki-Young Kim[†] · Sung-Hoon Son[††] · Dong-Ha Shin[†††]

ABSTRACT

Recently flash memory is widely accepted as a storage device of embedded systems for portability and performance reasons. Flash memory has many distinguishing features compared to legacy magnetic disks. Especially, a file system for flash memory usually assumes the form of log-structured file system and it employs garbage collector accordingly. Since the garbage collector can greatly affect the performance of file system, it should be designed carefully considering flash memory features. In this paper, we suggest a new garbage collector for existing JFFS2 (Journaling Flash File System II) file system. By extensive performance evaluation, we show that the proposed garbage collector achieves improved performance in terms of flash memory consumption rate, increased flash memory life time, and improved wear-leveling.

Key Words : Flash Memory, JFFS2, File System, Garbage Collector, Wear-leveling, Low-energy system

1. 서 론

플래시 메모리는 MP3 플레이어, PMP(Portable Multimedia Player) 등과 같은 임베디드 기기에서 디스크를 대신하는 저장 매체로써, 가볍고 충격에도 강하며 빠른 메모리 접근 속도를 가지기 때문에 다양한 분야의 기기에 널리 사용되고 있다. 그러나 읽기 속도에 비해 쓰고 지우는 속도가 느리며, 메모리를 구성하는 블록의 사용 수명이 있기 때문에 플래시 메모리 사용에 걸림돌로 작용되고 있다.

플래시 메모리는 정해진 일정 크기의 블록 단위로만 쓰고 지우는 것이 가능하며, JFFS2에서는 이를 지움 블록(eraseblock)[1]이라고 한다. 지움 블록의 크기는 메모리 특성에 따라 차이가 있지만, 블록 구조를 변경함에 따라 큰 블록의 쓰기가 가능한 파일 시스템도 개발되고 있다[1]. 그리고 플래시 메모리에 데이터를 덮어쓸 수 없으며, 각 지움 블록은 사용 수명 이상 지워지면 더 이상 데이터를 쓸 수 없다. 이 같은 메모리적 특성상 디스크 기반 파일시스템과는 다른 플래시 전용 파일 시스템이 필요하며, 대표적인 플래시 메모리 파일 시스템으로는 JFFS(Journaling Flash File

† 준 회 원 : 상명대학교 일반대학원 컴퓨터과학과 석사 과정
†† 정 회 원 : 상명대학교 소프트웨어학부 조교수
††† 정 회 원 : 상명대학교 소프트웨어학부 부교수
논문접수 : 2006년 7월 7일, 심사완료 : 2006년 11월 1일

1) 지움 블록은 플래시 메모리와 파일 시스템에서 공통된 의미를 가진다. 그렇기에 본 논문에서는 플래시 메모리의 관점에서는 지움 블록, 파일 시스템의 관점에서는 블록이라는 용어를 사용하였다.

System), JFFS2, YAFFS(Yet Another Flash File System), YFFS2, RFS(Robust File System) 등이 있다[2].

플래시 메모리는 디스크처럼 데이터를 덮어 쓸 수 없기 때문에 다시 쓰기 위해서는 데이터를 지우는 초기화 작업을 거쳐야 하고, 정해진 크기의 블록 단위로 지워야 하기 때문에 블록 내의 유효 데이터는 자유(free) 블록으로 다시 써야 한다. 이러한 대부분의 작업들은 파일 시스템의 가비지 콜렉터 모듈에 의해 수행되기 때문에 플래시 메모리 파일 시스템에서 가비지 콜렉터가 가지는 역할은 중요하다[3].

플래시 메모리에서 지움, 쓰기 작업은 다른 작업들에 비해 많은 시간과 자원을 소비하며[4], 가비지 콜렉터가 자주 호출되면 지움, 쓰기 작업도 같이 증가하기 때문에 이러한 상황은 시스템에 오버헤드를 가져온다. 이렇듯 가비지 콜렉터의 잦은 호출은 직접적으로 장치 성능의 저하를 가져 오게 될 수 있다.

JFFS2(Journaling Flash File System II)는 YAFFS와 함께 임베디드 시스템에서 가장 널리 사용되는 플래시 메모리 파일 시스템이다. JFFS2의 가비지 콜렉터는 파일 시스템 내에서 쓰레드(thread)로 동작하며, 파일 기록과 변경 시 수퍼블록을 갱신하는 함수와 메모리 공간을 확보하기 위한 함수에서 자동적으로 호출된다. 일반적으로 가비지 콜렉션은 블록 단위로 수행되며, 더 이상 사용되지 않는 공간을 지워 재사용 가능한 공간으로 만드는 데에 목적이 있다. 이렇듯 가비지 콜렉터의 역할은 가비지 콜렉션 작업을 통해 블록 내의 사용하지 않는 공간을 수집함으로써 새로운 데이터를 쓸 수 있는 공간을 확보하는 데에 있다[5].

JFFS2의 가비지 콜렉터의 문제점은 자주 접근은 되지만 변경이 일어나지 않는 데이터나 읽기 전용 데이터에 대해서 다시 쓰기 작업을 일으켜 메모리 사용률을 증가시키고, 이로 인한 전력과 플래시 메모리 사용 수명에 영향을 주는 데에 있다. 그렇기 때문에 본 논문에서는 다시 쓰기 작업을 감소시킬 수 있도록 가비지 콜렉터를 설계하여 오버헤드를 줄이고, 시스템 성능을 향상시키는데 목적을 두었다.

새로운 가비지 콜렉터의 동작은 다음과 같다. 먼저 가비지 콜렉션의 대상이 되는 데이터가 여러 번의 다시 쓰기 작업이 수행되었는지 검사한다. 그리고 설정한 임계치(threshold) 이상 다시 쓰기가 수행된 데이터는 변경되지 않거나 읽기 전용 데이터로 가정하고, 이러한 데이터에 대해서는 다시 쓰기가 수행될 때 현재 자유 블록들 중 가장 사용률이 높은 블록에 다시 쓰기를 수행한다. 그리고 이렇게 쓰여진 블록을 가비지 콜렉션의 대상이 되는 것을 회피하여 다시 쓰기 되는 것을 방지하였다. 이렇게 구조를 변경함으로써 불필요한 다시 쓰기 작업을 줄여 메모리 사용률을 감소시켰다.

본 논문에서는 EZ-X5 보드에 3가지 작업부하(workload)를 적용하여 수행함으로써 새로 설계한 가비지 콜렉터에 대한 성능평가를 수행하였다. 성능 평가를 위한 테스트에서 기존의 JFFS2 가비지 콜렉터에 비해 각각 50%, 34%, 14%의 지움 횟수 감소와 개선된 쓰기 평준화(wear-levleing)의 성능을 보였다. 이러한 성능 평가 결과는 특정 조건의 상태에서 기존의 파일 시스템에 비해 개선된 성능을 보여줄 수 있음을 뒷받침해 주고 있다.

본 논문은 2장에서 플래시 메모리, LFS, JFFS2 논문과 관련 연구에 대해 살펴 보도록 하고, 3장에서는 기존의 가비지 콜렉터와 본 논문에서 제시하는 개선된 가비지 콜렉터의 구조와 동작에 대해 자세히 설명하였다. 그리고 4장에서는 테스트 결과를 토대로 한 개선한 가비지 콜렉터의 성능 평가를 중점적으로 살펴보고, 성능평가를 통한 결론 도출과 향후 과제는 5장에 서술하였다.

2. 관련 연구

2.1 플래시 메모리

플래시 메모리란 Flash EEPROM(Electrically Erasable and Programmable Read Only Memory)을 말하며, 램과 롬의 중간 형태인 메모리이다. 플래시 메모리는 셀을 구성하는 게이트에 따라 크게 NOR 플래시 메모리와 NAND 플래시 메모리로 구분된다. NOR 플래시 메모리와 NAND 플래시 메모리의 가장 큰 차이는 버스 인터페이스에 있다. NOR 플래시는 SRAM과 같은 메모리 디바이스들과 마찬가지로 주소(address)/데이터(data) 버스에 바로 연결되고, 이에 반해 NAND 플래시 메모리는 일부 추가된 제어 핀들을 가지는 다중 I/O 인터페이스를 사용한다[6]. 현재는 NAND 플래시 메모리가 폭넓게 쓰이고 있는 실정이다. 기본적으로 모든 플래시 메모리는 같은 위치에 데이터를 덮어 쓸 수 없으며, 데이터를 쓰는 작업은 1을 0으로 바꾸거나 1을 그대로 둠으로써 수행된다. 그리고 플래시 메모리는 약 100,000회 정도 지워지게 되면 지움 블록을 다시 사용할 수 없게 되기 때문에 모든 지움 블록을 균일하게 사용하기 위한 다양한 쓰기 평준화 기법이 연구 되고 있다[2, 7-11]

또 하드웨어적으로 지움 블록 단위로 삭제를 수행해야 하기 때문에 이와 같은 하드웨어적 특성들은 일반적으로 사용되는 디스크 기반의 파일 시스템과는 다른 플래시 메모리 전용 파일 시스템을 필요로 한다.

2.2 LFS(Log-structured File System)

임베디드 리눅스에서 사용되는 대부분의 플래시 전용 파일 시스템은 LFS(Log-Structured File System) 에 기반을 두고 있다. LFS는 디스크 쓰기 성능의 향상을 위해 순차적으로 데이터를 기록함으로써 쓰기를 위한 위치 탐색에서 디스크 헤드를 움직이는데 발생하는 오버헤드를 줄이기 위한 방법으로 고안되었다. 기본적인 아이디어는 변경된 데이터를 캐시에 버퍼링하고, 일련의 쓰기 동작으로 데이터를 순차적으로 쓰는 데에 있다. 이것은 모든 데이터를 디스크에 "로그"라고 불리는 순차적 파일 구조로 씀으로써 가능하며, 이러한 접근은 거의 모든 탐색 시간을 제거함으로써 쓰기 성능을 동적으로 증가시킨다[12]. 그리고 캐시에 버퍼링하기 때문에 가장 최근에 변경된 로그 데이터가 저장 장치에 쓰인다. 이러한 구조는 쓰기 성능뿐 아니라 변경된 파일의 이

전 파일이 남아 있기 때문에 빠른 캐시 복구를 가능하게 하여 안전한 시스템을 제공한다.

하지만 LFS에 있어서 문제점은 자유 공간의 관리에 있다. 디스크 공간이 유한하기 때문에 데이터를 계속 쓰기 위해서는 공간을 재사용하여야 하며, 시스템이 효율적으로 동작하기 위해서는 항상 새로운 데이터를 쓸 수 있도록 충분히 자유 공간이 보장되어야 한다. 그렇기 때문에 LFS에서는 세그먼트(segment)와 지움 정책(cleaning policy)의 개념을 도입하였다. 세그먼트란 일정 크기의 공간을 고정된 크기의 공간으로 사용할 수 있도록 나눈 것을 말하는데, 세그먼트 단위로 데이터를 쓰고, 지워 공간의 낭비를 막고, 지움 정책을 통해 사용되지 않는 세그먼트에 대한 지움 작업으로 자유 공간을 확보한다[13].

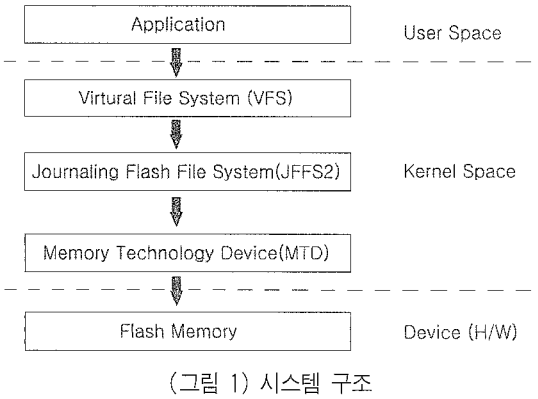
2.3 JFFS2(Journaling Flash File System II)

최초의 JFFS는 디스크가 없는 임베디드 시스템에서 기대되지 않은 파일 손상 및 손실에 대한 안전한 파일시스템을 제공하고자 Axis Communication AB에 의해 개발되었으며[2], REDHAT에 의해 JFFS2로 버전업을 하면서 기존의 JFFS보다 압축, 하드 링크 등 향상된 기능을 제공하게 되었다. 그리고 LFS기반의 파일 시스템으로 설계되었기 때문에 신뢰성 있는 저널링 기능을 제공한다.

JFFS2는 MTD(Memory Technology Device) 레벨 위에서 동작한다. MTD는 디스크가 아닌 메모리 장치들을 위해 제공되는 인터페이스이며, 페이지 단위의 쓰기 동작과 지움 블록단위의 지움 동작, 데이터의 압축 저장을 지원한다. 또 디스크 기반의 파일 시스템과 마찬가지로 디스크 캐시를 제공한다. (그림 1)은 플래시 메모리 파일 시스템의 구조를 보이고 있다.

JFFS2는 데이터를 노드(node)라는 단위로 쓰며, 한번에 플래시 메모리에 쓰는 것이 아니라 일정 크기의 바이트를 플래시 메모리에 여러 번에 걸쳐서 쓰도록 쓰기 버퍼를 지원한다. 쓰기 버퍼의 크기는 버전에 따라 다르지만 보통 256, 512 바이트를 지원하며, 이 쓰기 버퍼는 램에 위치하고 있다. 파일이 쓰여지기 위해서는 먼저 여러 개의 노드로 나눠지게 되고, 각 노드는 차례로 쓰기 버퍼에 쓰여지게 된다. 그리고 쓰기 수행 중 쓰기 버퍼가 가득 차게 되면 flush를 수행하게 된다. flush란 플래시 메모리에 데이터를 쓰는 작업을 말하며, flush 과정은 데이터 쓰기와 쓴 데이터의 고정 작업으로 이루어진다.

JFFS2는 파일 시스템을 초기화하는 과정에서 블록들을 수퍼 블록의 리스트에 삽입하여 관리한다. 이 과정에서 지움 블록의 물리적인 특성은 고려되지 않으며, 해당 데이터의 종류에 따라 각각의 리스트로 분류되어 관리된다[7]. 그리고 마운트(mount) 시 한 번의 블록 순환과 가비지 콜렉터의 블록 선택 구조를 통해서 쓰기 평준화 기법이 사용한다. 그러나 이러한 쓰기 평준화 기법은 크게 만족스러운 성능을 발휘하고 있지 못하고 있다.



(그림 1) 시스템 구조

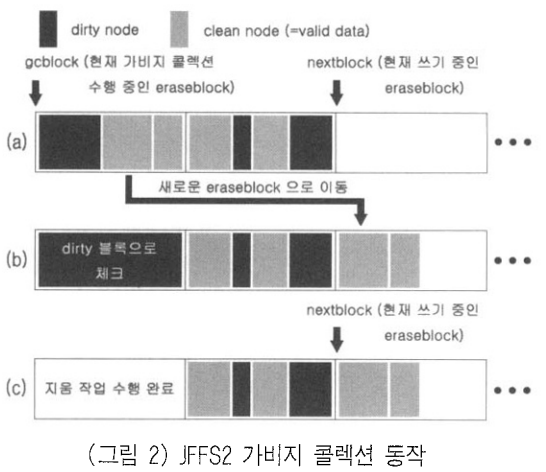
3. 가비지 콜렉터 설계

3.1 JFFS2 가비지 콜렉터

JFFS2의 특징 중 하나는 모든 블록을 연결 리스트의 구조로 가지고 있다는 점이다(<표 1> 참조). 그렇기 때문에 특정 블록을 가져오기 위해 매번 모든 블록을 검사할 필요가 없으며, 해당 블록이 포함된 연결 리스트에서 순차적으로 꺼내옴으로써 블록 탐색 시간을 줄일 수 있다. 그렇지만 마운트 시에 모든 블록에 대해서 리스트 구조를 모두 구성해야 하기 때문에 다른 파일 시스템에 비해 마운트 시간이 길다는 단점을 가지고 있다.

<표 1> JFFS2 연결 리스트 구조체

리스트	리스트 역할
clean_list	100% 유효한 데이터가 들어 있는 블록의 리스트
dirty_list	일부 데이터가 dirty 인 블록의 리스트
very_dirty_list	대부분의 데이터가 dirty 인 블록의 리스트
erasing_list	삭제 중인 블록의 리스트
erase_pending_list	삭제 대기중인 블록의 리스트
erase_complete_list	삭제가 완료된 블록의 리스트
free_list	사용 가능한 블록의 리스트
bad_list	사용 불가능한 블록의 리스트
bad_used_list	블록의 일부가 bad sector 인 블록의 리스트



(그림 2) JFFS2 가비지 콜렉션 동작

JFFS2에서는 사용되지 않는 dirty 노드에 대해 가비지 콜렉션을 수행함으로써 공간을 확보하게 되는데 (그림 2)는 이러한 과정을 보여주고 있다.

가비지 콜렉션이 수행되면 가장 먼저 하는 것이 가비지 콜렉션을 수행할 블록을 선택하는 작업이다. 현재의 타이머 인터럽트 발생 횟수 값을 상수 128로 나눈 나머지 값을 가지고 리스트에서 블록을 선택하는데 bad_used_list > erasable_list > very_dirty_list > dirty_list > clean_list 순서의 확률로 블록이 선택된다. 가비지 콜렉션을 수행할 블록이 선택되면, 블록 내의 노드들을 차례로 검사하여 노드의 타입, 유효 노드의 유무 등을 체크한다. 그래서 유효 노드에 대해서는 새로운 노드로 다시 쓰기를 수행한다. 이 같은 이유는 여러 번 언급했듯이 플래시 메모리 특성상 지움 블록 단위의 삭제만 지원하기 때문이다. 그리고 유효 노드를 새로운 블록으로 다시 쓰면, 기존의 노드는 dirty 노드로 체크된다. 블록 내의 모든 유효 노드가 새로운 블록으로 다시 써지고, 기존의 모든 노드가 dirty 노드로 체크되면 해당 블록은 obsolete 블록, 즉 사용되지 않는 블록으로 체크되어 지우는 것이 가능하게 된다. 이후 지움 작업을 통해 obsolete 블록을 자유 블록으로 바꿔 줌으로써 메모리 공간 확보를 하게 된다.

3.2 개선된 가비지 콜렉터

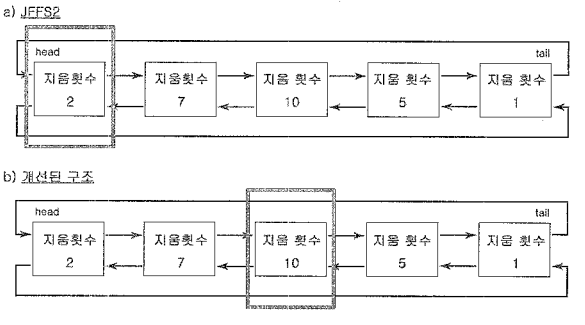
JFFS2 가비지 콜렉터의 주된 작업은 블록 내의 유효 노드를 새로운 블록으로 다시 쓰기 작업을 수행하고, 그래서 생긴 dirty 데이터, 즉 쓸모 없어진 공간을 수집한다는 것에 그 목적이 있다. 그러나 공간 확보를 위한 가비지 콜렉션 동작에서 유효 노드가 반복적으로 다시 써지게 된다. 그렇기 때문에 이동이 필요 없는 변경되지 않는 읽기 전용 노드나 자주 접근은 되지만 변경이 일어나지 않는 노드의 경우 가비지 콜렉터에 의해 다시 쓰기 작업이 수행 될 수 있다. 이것은 지움 블록을 자주 지워지게 만들어 메모리 사용률을 높이고, 이로 인한 파일 시스템 동작 지연을 초래한다. 그렇기 때문에 본 논문에서 제시하는 새로운 가비지 콜렉터로 다시 쓰기 작업을 줄임으로써 블록 지움 횟수를 줄여 메모리 수명 연장과 다시 쓰기 작업에서 오는 오버헤드를 감소시키는 효과를 기대할 수 있다. 게다가 현재의 JFFS2 가지비 콜렉션 기법이 구조적으로 가지고 있는 쓰기 평준화 한계를 지움 정책 변경을 통해 보다 개선될 수 있도록 하였다.

3.3 복사 횟수(Copy Count)와 지움 횟수(Erase Count)

본 연구의 핵심 내용은 읽기 전용 노드와 변경되지 않은 노드의 적절한 관리에 있다. 이를 위해 노드 구조체에 다시 쓰기를 수행한 횟수를, 블록 구조체에는 각 블록의 지움 횟수를 추가하였다. 복사 횟수는 노드가 얼마나 오랫동안 플래시 메모리에 머무르는지를 판단할 수 있는 척도인데, 복사 횟수가 많다는 것은 노드에 자주 다시 쓰기 작업이 수행되었음을 의미한다. 그리고 해당 노드가 다른 노드들에 비해 변경되지 않는 노드이거나 읽기 전용 노드일 가능성이 높다는 것을 의미하기도 한다. 이 같은 예측을 참고로 정해 놓은 임계치 이상의 복사 횟수를 보이는 노드를 읽기 전용이거나 변경되지 않는 노드로 가정하고, 이렇게 가정한 노드에 대해서는 기존 노드와는 다른 구조를 통한 가비지 콜렉션 동작이 수행하도록 하였다.

3.4 추가된 쓰기 버퍼 (Write Buffer)

본 논문에서는 기존의 쓰기 버퍼에 새로운 쓰기 버퍼를 추가하였다. 기존에 쓰기 버퍼와 추가된 쓰기 버퍼의 차이는 블록 할당과 블록을 리스트에 삽입하는 구조에 있다. 기존의 쓰기 버퍼는 flush를 수행할 블록을 할당할 때 free_list에서 가장 앞쪽에 위치한 헤드(head)에 위치한 블록을 할당 받는다((그림 3)(a)).



(그림 3) 쓰기 버퍼의 블록 할당 구조 비교

그리고 쓰기 버퍼가 할당 받은 블록이 모두 쓰여지고 리스트에 블록을 삽입해야 할 때 블록 내의 dirty 데이터 크기에 따라서 very_dirty_list, dirty_list, clean_list 중 하나에 삽입된다. 이러한 구조는 지움 블록의 사용률과 수명에 대한 고려 없이 수행되기 때문에 플래시 메모리 사용에 있어 효율적이지 못하다. 이에 새로 추가한 쓰기 버퍼에서 이러한 점을 개선할 수 있는 구조를 적용하였다. 쓰기 버퍼가 블록을 할당 받을 때는 free_list 내에서 가장 사용률이 높은 블록, 즉 가장 빈번하게 지워진 블록을 할당한다((그림 3)(b)). 그리고 새로운 블록을 할당 받아야 할 때에는 새롭게 추가된 egc_list에 사용한 블록을 삽입하고, 앞에서와 같이 사용률이 가장 높은 블록을 다시 할당 받는다. egc_list는 리스트 내 블록들을 가비지 콜렉션이 자주 일어나지 않도록 방지해 줌으로써 전체블록을 고르게 쓸 수 있다는 점에서 기존보다 개선된 쓰기 평준화 성능을 보여준다.

3.5 egc_list

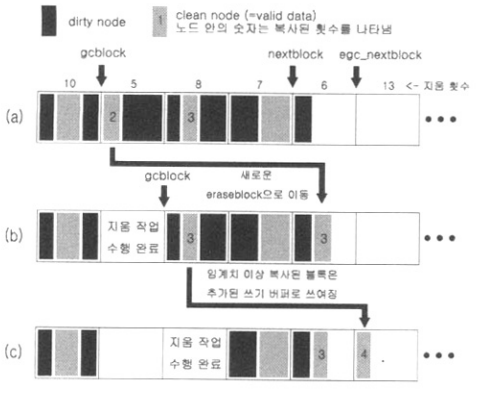
개선한 가비지 콜렉터에서는 새로운 쓰기 버퍼와 함께 egc_list를 추가하였다. egc_list의 역할은 리스트 내의 블록이 다른 블록들 보다 가비지 콜렉션의 대상이 되는 확률을 낮추는 데에 있다. 실제로 egc_list는 블록 선택 구조를 수행하는 jffs2_find_gc_block 함수에서 clean_list에 블록이 없을 경우 egc_list가 선택되도록 수정하였기 때문에 낮은 확률로 블록이 선택된다. 이러한 구조는 반복적으로 다시 쓰기가

수행된 데이터를 가지는 블록이 가비지 콜렉션의 대상이 되지 않도록 방지함으로써 다시 쓰기 작업이 수행되지 않도록 한다. 그리고 사용률이 높은 지움 블록에 대해서 지움 작업을 억제하여 전체 지움 블록의 지움 횟수를 고르게 하는 역할을 수행하며, 결과적으로 다시 쓰기 작업을 줄여서 메모리 사용률을 감소시키는 효과를 가져온다.

3.6 개선된 가비지 콜렉션 동작

(그림 4)에서 각각 2회와 3회 다시 쓰기 작업이 수행된 유효 노드가 있음을 볼 수 있으며, 노드의 다시 쓰기 임계치는 3으로 (4.3 실험 결과에서 3 또는 4가 최적값으로 나타났다.) 설정하였다. 블록 선택 구조에 의해서 두 번째 블록이 가비지 콜렉션의 대상이 되면 복사 횟수 2회의 유효 노드는 복사 횟수가 임계치를 넘지 않기 때문에 현재 사용중인 다섯 번째 블록에 노드로 다시 쓰게 되고, 이후 세 번째 블록이 가비지 콜렉션의 대상이 되면 블록 내의 복사 횟수가 3회인 유효 노드는 임계치를 넘게 된다는 것을 알 수 있다. 그렇게 되면 가지비 콜렉터는 이 유효 노드를 읽기 전용이거나 변경되지 않는 노드로 인식하게 된다. 3.4장에서 언급한 것과 같이 임계치 이상 다시 써지게 된 노드는 새로 추가한 쓰기 버퍼에 쓰여지게 된다. 임계치 이상의 다시 쓰기 작업이 수행된 해당 유효 노드는 현재 지움 횟수가 가장 높은 여섯 번째 블록에 저장된다. 그리고 추가된 쓰기 버퍼가 할당 받은 블록의 모든 공간이 데이터로 가득 차게 되어 리스트에 블록을 삽입하고 새로운 블록을 할당 받아야 할 시점이 오게 되면, 추가된 리스트 구조인 egc_list에 블록을 삽입하게 된다. 그리고 추가된 쓰기 버퍼는 이전과 마찬가지로 free_list에서 가장 사용률이 높은 블록을 할당 받는다. 이 egc_list는 가비지 콜렉션의 대상이 되는 블록 선택 구조에서 bad_used_list > erasable_list > very_dirty_list > dirty_list > clean_list > egc_list 의 확률로 블록을 선택되도록 하여서 읽기 전용이거나 변경되지 않는 노드에 대해서는 가비지 콜렉션 대상이 되는 횟수를 줄이도록 만든다.

결과적으로 이 구조는 가비지 콜렉터에 의한 불필요한 반복적 다시 쓰기 작업을 억제하는 역할을 수행하여 가비지 콜렉터 호출과 지움 블록의 지움 횟수를 감소시키고, 개선된 쓰기 평준화 동작을 지원하게 된다.



(그림 4) 개선된 가비지 콜렉션 동작

4. 성능평가

본 논문에서는 시뮬레이션을 통한 성능평가보다 정확한 데이터를 얻기 위해 FALINUX 사의 EZ-X5 보드를 사용하여 성능평가를 수행하였다.

4.1 테스트 환경(Test Environment)

성능평가를 위해 EZ-X5 보드에서 20,000회 이상의 읽기, 쓰기와 삭제가 수행되는 스크립트 파일을 생성하여 23k 바이트의 임의의 파일을 대상으로 테스트를 하였다. 그리고 스크립트 수행 중 파일이 없는 상황에서의 삭제 동작을 방지하고자 테스트 수행에 앞서 플래시 메모리 크기의 약 90% 정도를 임의의 파일로 채워 놓았다. 또 새롭게 설계한 가비지 콜렉터 구조의 성능 평가를 위해서 3가지 작업부하를 가정하여 테스트를 수행하였다. 첫 번째는 파일의 삭제, 쓰기 동작을 비연속적으로 수행하는 작업부하, 두 번째는 일부 영역의 파일들만 쓰기, 삭제가 되고 나머지 파일들은 읽기만 수행하는 작업부하이다. 그리고 마지막은 기본 동작이 두 번째와 같지만 100번의 읽기, 쓰기, 삭제 동작 중 1번은 나머지 파일들을 대상으로 삭제와 쓰기를 수행하는 작업부하이다. 본 논문에서는 3가지 작업부하를 테스트 함으로써 다양한 환경에 대해 신뢰성 있는 데이터를 기대 할 수 있다. 한편 표 2는 실험에 사용된 하드웨어 환경을 요약하고 있다.

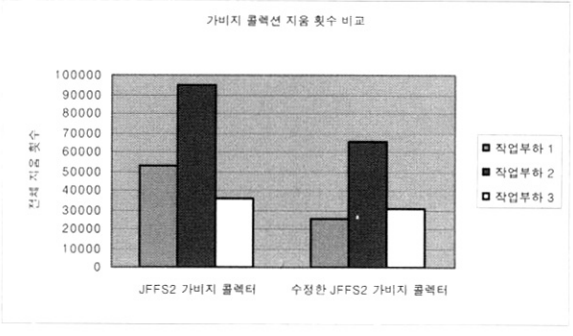
〈표 2〉 테스트 환경[13]

테스트 수행 환경	
테스트 보드	FALINUX EZ-X5
보드 커널 버전	KERNEL-2.4.17
JFFS2 버전	JFFS2-v2.2
MTD 버전	2003년 4월 10일 update 버전
플래시 메모리	
모델명	K9F1208U0M
메모리 타입	64M x 8 Bit NAND Flash Memory
전체 메모리 크기	64M + 2,048K(OOB)
블록 크기	16K (JFFS2 는 32K)

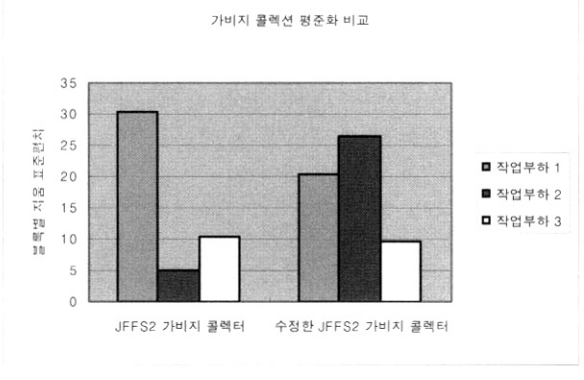
4.2 실험 결과

본 논문에서는 각각의 작업부하를 기존의 JFFS2와 개선된 JFFS2에서 테스트함으로써 얻어진 결과를 분석하였으며, 평가된 결과는 다음과 같다.

(그림 5)는 동일한 작업부하를 수행했을 때 지움 블록의 전체 지움 횟수를 비교한 것이다. 기존의 JFFS2에 비해 첫 번째 작업부하는 약 50%, 두 번째는 작업부하는 32%, 세 번째는 작업부하의 경우는 14% 정도의 지움 감소 효과를 보였다. 지움 횟수 감소는 메모리 사용률과 직접적으로 연



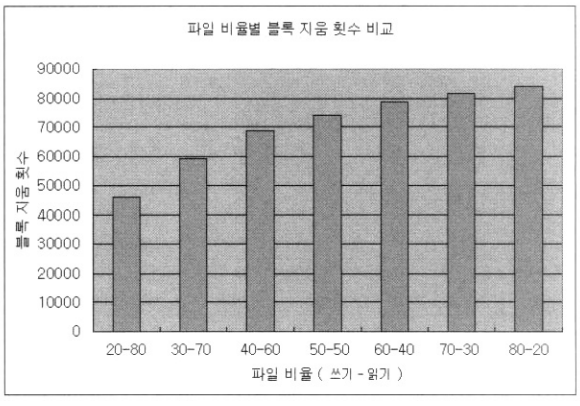
(그림 5) 가비지 콜렉터 성능 비교 - 지움 횟수



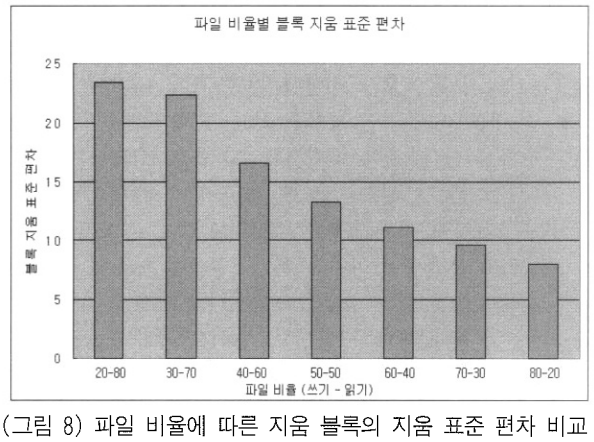
(그림 6) 가비지 콜렉터 성능 비교 - 쓰기 평준화

관되어 있으므로 지움 횟수를 사용률로 보아도 무방하다. (그림 6)은 각 지움 블록당 지움 횟수에 대한 표준 편차를 계산하여 쓰기 평준화의 개선 정도를 비교한 그림이다. 작업부하 1과 작업부하 3에서는 각각 23%, 6% 정도의 감소 효과를 보이고 있다. 다만 작업부하 2에서만은 표준편차가 상승하였다. 이 같은 결과를 보이는 이유는 작업부하 1과 3은 전체 파일을 대상으로 읽기, 쓰기, 지우기를 수행하여 지움 블록이 순환할 수 있는 조건을 만족시켜줌으로써 개선된 쓰기 평준화를 보인 반면, 작업부하 2의 경우는 일부 한정된 파일만을 대상으로 쓰기, 지우기를 수행하였기 때문에 나머지 읽기만 수행하는 파일들의 경우 egc_list에 오랫동안 머무르게 되어 가비지 콜렉션의 대상이 될 확률이 매우 낮아진다. 따라서 일부 블록들이 극단적으로 가비지 콜렉션의 대상이 되는 것을 피하게 되어 그림과 같은 결과가 나타나게 되었다.

(그림 7)과 (그림 8)은 플래시 메모리에 저장되어 있는 파일의 특성을 고려한 테스트로 읽기 전용 파일과 변경 가능한 파일의 비율을 조정함에 따라 성능이 어떻게 달라지는가를 비교한 것이다. 읽기 전용 파일이 많은 비율을 차지하고 있는 작업부하에서는 지움 블록 사용률이 현저히 감소하는 것을 볼 수 있다(그림 7). 그리고 읽기 전용 파일의 비율을 낮춤으로 지움 블록 사용률이 올라가는 것을 알 수 있다. 이 같은 이유는 읽기 전용 파일의 비율을 높임으로써 새로운 가비지 콜렉터 구조에서 처리되는 파일의 비중이 늘어났기 때문으로 볼 수 있다. 그러나 사용률과는 달리 각 블



(그림 7) 파일 비율에 따른 지움 블록의 전체 지움 횟수 비교



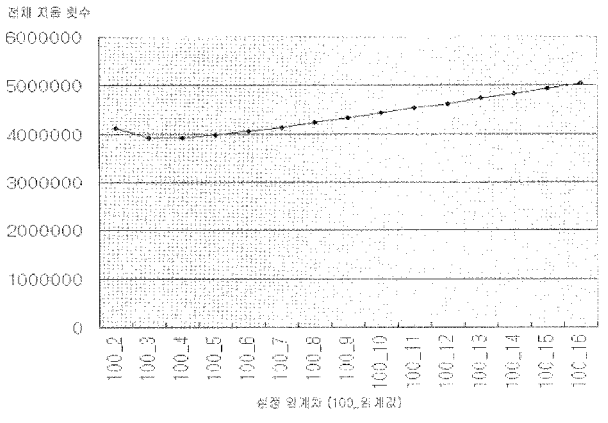
(그림 8) 파일 비율에 따른 지움 블록의 지움 표준 편차 비교

록들 간의 쓰기 평준화를 알 수 있는 지움 블록의 사용에 대한 표준 편차를 비교한 그래프에서는 반대로 읽기 전용 파일의 비율이 낮아 질수록 표준 편차가 커지는 것을 보여주고 있다(그림 8). 표준 편차가 커진다는 것은 그만큼 일부 지움 블록만이 빈번히 사용되고 있다는 사실을 말해주고 있기 때문에 가비지 콜렉션의 동작 횟수가 올라감에 따라 사용 횟수가 적은 지움 블록들이 빈번하게 선택되어 사용 편차를 줄여주고 있는 것이다.

4.3 임계치 설정

앞장에서 언급했듯이 최적화된 가비지 콜렉터의 동작을 구현하기 위해서 복사 횟수에 대한 적절한 임계치 설정이 중요한 과제이다. 그렇기 때문에 임계치에 따른 성능비교를 통해서 적절한 임계치 값을 구하였다.

현재 본 논문에서의 작업부하는 읽기 전용이거나 변경되지 않는 파일을 임의로 정해 두고 테스트를 수행하였기 때문에 임계치가 1인 경우에서 최적화된 성능을 보인다. 그렇기 때문에 랜덤한 동작을 수행하는 작업부하에 대해 적절한 임계치 값을 찾는 테스트를 수행하였다. (그림 9)는 랜덤한 동작을 수행하는 작업부하를 통해서 얻은 임계치에 따른 전체 지움 횟수를 비교한 결과를 보이고 있는데, 임계치 3과 4에서 가장 적은 지움 횟수를 보임을 알 수 있다. 이와 같은 이유는 임계치라는 것이 일정 횟수 이상 다시 쓰기가 수행



(그림 9) 임계치 설정에 따른 전체 지움 횟수 비교

된 유효 노드를 가려내기 위한 수치이기 때문에 랜덤한 작업부하에서는 임계 횟수 이상 다시 쓰기가 수행된 모든 유효 노드가 읽기 전용이거나 변경되지 않는 파일의 일부라고 확정할 수 없다. 언제든 변경이 될 수도 있고, 아닐 수도 있기 때문에 임계치에 따라서 위의 그림과 같은 수치를 보이게 된다.

5. 결과 및 향후 과제

본 논문에서는 임베디드 시스템에서 쓰이는 플래시 메모리 기반의 파일 시스템 중 하나인 JFFS2의 가비지 콜렉터의 문제점을 분석하고 개선된 가비지 콜렉터를 설계, 구현하였다. 성능 평가 결과를 종합해 볼 때 새로운 JFFS2 가비지 콜렉터의 동작이 실험에서 사용한 작업부하에서 기존 JFFS2에 비해 크게는 50%, 적게는 10% 정도의 향상된 성능을 제공하며, 전체 파일 시스템에서 읽기 전용 파일의 비율이 높을 때 보다 향상된 성능을 보인다는 것을 알 수 있었다. 그리고 실제로 작업부하의 수행시간을 비교해 보았을 때 약 10~30분 정도가 감소되었다. 이는 결과적으로 가비지 콜렉터의 호출 횟수를 줄이며, 동시에 잦은 다시 쓰기 작업에서 오는 오버헤드를 낮추게 되고, 발생되는 오버헤드에 대한 지연을 줄여줌으로써 파일 시스템의 안정화와 플래시 메모리의 수명 연장 효과를 보였다. 하지만 본 논문에서 제시한 가비지 콜렉터 구조에서 제기되는 문제점은 2가지가 있다. 첫 번째는 지움 블록의 크기에 비해 저장되는 파일의 크기가 커짐에 따라서 기대되는 효과가 감소될 수 있으며, 추가된 구조로 인해서 발생되는 오버헤드에 있다. 그리고 두 번째는 파일이 오랜 시간 동안 접근되지 않거나 변경되지 않으면 파일이 저장되어 있는 블록이 가비지 콜렉션의 대상이 될 확률이 너무 낮아지게 되어 (그림 6)의 작업부하 2와 같이 각 지움 블록당 지움 횟수가 편차가 크게 증가하게 되어 일부 지움 블록만을 극도로 많이 사용하게 되는 문제점을 발생시킨다는 것이다.

이러한 문제점들은 장기간 가비지 콜렉션의 대상이 되지 않은 블록에 대한 지움 정책의 변경으로 일부 해결될 수 있을 것으로 생각된다. 그리고 작업부하 상의 문제 해결과 리스트에 블록을 삽입하는 과정 중 블록 지움 횟수에 따른 삽입 구조를 변경함으로써 지금보다 개선된 쓰기 평준화를 지원할 수 있을 것이다.

참 고 문 헌

[1] 박정태, "큰 블록 NAND 플래시 메모리를 위한 파일 시스템의 설계 및 구현", 중앙대학교 대학원 석사학위논문, Dec., 2004.

[2] David Woodhouse, "JFFS: The Journaling Flash File System", Red hat Inc, 2004.

[3] LI-PIN CHANG, TEI-WEI KUO and SHI-WU LO, "Real-Time Garbage Collection for Flash-Memory Storage Systems of Real-Time Embedded Systems", ACM Transactions on Embedded Computing Systems, Vol.3, Issue.4, pp.837-863, November, 2004.

[4] Hyung Gyu Lee and Nae Hyuck Chang, "Low-Energy Heterogeneous Non-volatile Memory Systems for Mobile Systems", Journal of Low Power Electronics, Vol.1, No.1, pp.52-62, April, 2005.

[5] Paul Griffin and Witawas Srisa-an and J. Morris Chang, "An Energy Efficient Garbage Collector for Java Embedded Devices", in Proceeding of "ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems", June, 2005.

[6] Memory Technology Device (MTD) Subsystem for Linux, "Memory Technology Devices", http://www.linux-mtd.infradead.org/doc/nand.html

[7] 전승진, 공기석, 황달연, "플래시 메모리 파일 시스템의 지움 정책 개선에 관한 연구", 한국정보과학회 제30회 추계학술발표회, 2003.

[8] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda, "A Flash-Memory Based File System", in Proceeding of "Usenix Technical Conference", 1995.

[9] ERAN GAL AND, SIVAN TOLEDO, "Algorithms and Data Structures for Flash Memories", ACM Computing Surveys, Vol.37, Issue.2, pp.138-163, June, 2005.

[10] 김경윤, 김영필, 송인준, 유혁, "Greedy 방법을 개선한 플래시 메모리 지움 정책", 한국정보처리학회 춘계학술발표대회, April, 2004

[11] M. L. Chiang and R. C. Chang, "Cleaning policies in mobile computers using flash memory," The Journal of Systems and Software, Vol.48, No.3, pp.213-231, 1999

[12] Mendel Rosnblum and John K. Ousterhout, "The Design and Implementation of a Log-Structured File System", Vol.10, Issue.1, pp.26-52, ACM Transactions on Computer Systems, 1992.

[13] 최혁근, 장태무, "LFS를 위한 디스크 배열의 구성", 산업기술논문집, Vol. 6, pp.35-48, 1995

[14] Samsung Electronics, "K9F1208U0M Flash Memory Hardware Manual", April, 2001

김 기 영

e-mail : kgykingdom@smu.ac.kr
2005년 상명대학교 소프트웨어학부(학사)
2005~현재 상명대학교 일반대학원
컴퓨터과학과 석사과정
관심분야 : 임베디드 소프트웨어, 임베디드
운영체제, 파일 시스템 등

손 성 훈

e-mail : shson@smu.ac.kr
1991년 서울대학교 계산통계학과(학사)
1983년 서울대학교 전산과학과(석사)
1999년 서울대학교 전산과학과(박사)
1999년~2004년 한국전자통신연구원
선임연구원
2004년~현재 상명대학교 소프트웨어학부 조교수
관심분야 : 임베디드 시스템, 운영체제, 멀티미디어 시스템 등

신 동 하

e-mail : dshin@smu.ac.kr
1980년 경북대학교 전자공학과(학사)
1982년 서울대학교 전자계산기공학과(석사)
1994년 University of South Carolina
컴퓨터과학과(박사)
1982년~1996년 한국전자통신연구원
책임연구원
1997년~현재 상명대학교 소프트웨어학부 부교수
관심분야 : 저 전력 임베디드 소프트웨어, 실시간 임베디드
운영체제, 임베디드 리눅스 운영체제 등