플래시 메모리 상에서 효율적인 가비지 컬렉션을 위한 라이브 페이지 캐시

이지원° 노홍찬 박상현

연세대학교 컴퓨터과학과

{sppong°, fallsmal, sanghyun}@cs.yonsei.ac.kr

Live Page Cache for Efficient Garbage Collection on the Flash-Memory Storage System

Jiwon Lee° Hongchan Roh Sanghyun Park

Department of Computer Science, Yonsei University

요 약

최근 플래시 메모리는 빠른 용량의 증가와 가격 하락으로 인하여 새로운 저장 매체로서 그 사용이 널리 퍼지고 있다. 하지만 읽기 속도에 비해 쓰기 속도와 지우기 속도가 느리고 사용할 수 있는 횟수가 제한되어 있으며 내부공간 업데이트(in-place update)가 불가능하다는 단점이 있어 플래시 메모리의 쓰기 가능한 공간이 부족할 때 공간 확보를 위한 가비지 컬렉션(garbage collection)을 따로 수행하여야 한다. 이러한 가비지 컬렉션 작업으로 인하여 진행중인 프로세스가 지연될 수 있기 때문에 그 지연 시간을 최소화 하기 위한 방법이 필요하다. 이를 위하여 본 논문은 메인 메모리의 일부분을 사용하여 플래시 메모리의 정보를 저장함으로써 그 지연 시간을 단축할 수 있는 방법을 제안한다.

1. 서 론

플래시 메모리는 비활성 저장 매체로서 모바일에서부터 시작해서 최근에는 하드디스크를 대체해 가며 빠른 속도로 계속 발전하고 있다. 최근에는 용량이 크게 늘어나고 가격이 많이 내려가면서 더욱 널리 퍼지고 있는 추세이다. 특히 플래시 메모리는 충격에 강하고 전력 손실이 적기 때문에 노트북과 같은 휴대용 컴퓨터에서 많이 사용되고 있다.

하지만 플래시 메모리는 몇 가지의 단점을 갖고 있다. 그에 앞서 플래시 메모리의 구조를 간단히 설명하면 플래시 메모리는 여러 개의 블록(block)으로 이루어지며 각각의 블록은 일반적으로 64개 또는 128개의 페이지(page)로 구성되며 페이지는 데이터 저장의 최소 단위이다. 플래시 메모리의 첫 번째 단점은 읽기, 쓰기와 지우기의 단위가 다르다는 점이다. 즉, 읽기와 쓰기는 페이지 단위로 발생하지만, 지우기는 블록 단위로 발생한다. 이러한 특징은 플래시 메모리의 내부공간 업데이트(in-place update)가 불가능하다는 특징 때문에 단점으로 작용한다[8]. 내부공간 업데이트란 기존의 하드디스크와 같이 데이터의 덮어쓰기를 하는 것이다. 하지만 플래시 메모리는 내부공간 업데이트가 불가능하기 때문에 데이터가 업데이트가 될 때 해당 데이터를 지우고 다시 쓰거나, 아니면 다른 새로운 공간에 데이터를 저장해야 한다. 그러나 해당 데이터를 지우고 쓰는 것은 지우기 작업이 블록 단위로 일어난다는 특징 때문에 업데이트가 발생한 해당 데이터 이외의 다른 데이터도 함께 지워질 수 있다. 플래시 메모리의 또 다른 단점은 지우기 횟수가 한정되어 있다는 것이다. 일반적으로 블록의 지우기 횟수가 만 번에서 백만 번을 넘게 되면 오류가 발생하여 해당 블록은 사용할 수 없다[7]. 또한 표 1에서 확인할 수 있듯이 플래시 메모리는 읽기 속도에 비해 느린 쓰기 속도를 갖고 있다.

표 1. 낸드(NAND) 플래시 메모리의 페이지 읽기, 쓰기, 블록 지우기 시간[1]

페이지 읽기 시간	페이지 쓰기 시간	블록 지우기 시간
77.8μs(2KB)	252.8μs(2KB)	1500μs(128KB)

표 2. 논문에서 사용되는 플래시 메모리 용어 정리[4]

용어	설명
프리 페이지 (free page)	블록 내 쓰기 가능한 공간
라이브 페이지 (live page)	업데이트 이후의 최신 데이터가 저장된 공간
데드 페이지 (dead page)	업데이트 이후의 삭제되지 않은 데이터가 저장된 공간

기존의 많은 파일 시스템들은 이러한 단점들을 보완하기 위해서 많은 방법들을 제시하였고 그 중에 중요한 부분을 차지하는 것이 가비지 컬렉션(garbage collection)이다. 플래시 메모리는 내부공간 업데이트가

불가능하다는 특징 때문에 업데이트 후에 삭제되지 않은 데드 페이지(dead page)들을 저장하고 있어 플래시 메모리 공간을 효율적으로 사용할 수 없다. 가비지 컬렉션은 데드 페이지들이 차지하고 있는 공간을 확보하기 위해 수행되는 과정이다. 가비지 컬렉션에 수반되는 지우기 작업은 블록 단위로 진행되기 때문에 블록 내의 유효한 정보인 라이브 페이지(live page)들을 다른 블록으로 복사를 하고 해당 블록을 지우는 방식으로 수행된다.

앞에서 언급한 바와 같이 플래시 메모리는 블록의 지우기 횟수가 만 번에서 백만 번 정도로 한정되어 있다. 따라서 어느 특정 블록에만 쓰기와 지우기가 반복된다면 그 블록은 다른 블록에 비해 먼저 사용이 불가능해진다. 이러한 문제를 해결하기 위해서는 쓰기와 지우기 작업이 어느 특정 블록에만 몰리지 않고 다른 블록들의 지우기 횟수를 고려해가며 사용하도록 해야 하며 이러한 방법을 웨어레벨링(wear leveling)이라 한다[7].

많은 파일 시스템들에서는 플래시 메모리의 사용 중에 쓰기 작업에 의해 쓰기 가능한 공간인 프리 페이지(free page)의 전체 양이 일정 수준 이하로 내려가게 되면 가비지 컬렉션이 발생된다. 가비지 컬렉션은 크게 두 가지 방법으로 수행되는데 하나는 프리 페이지의 양이 일정 수준 이하로 내려가면 또 다른 일정 수준 이상의 프리 페이지가 확보될 때까지 가비지 컬렉션을 계속 수행하는 방법이고, 다른 하나는 프로세스 진행 중에 필요한 만큼만의 프리 페이지를 확보하는 방법이다. 이 두 가지 방법 모두 우선순위가 높은 실시간 프로세스 수행 중에도 발생될 수 있으며 두 방법 모두 프로세스를 지연시키며 특히 첫 번째 방법의 경우에는 많은 프리 페이지를 확보하기 때문에 프로세스를 크게 지연시킬 수 있다[2].

본 논문에서는 이러한 가비지 컬렉션의 특징에 의해서 발생할 수 있는 문제점 중 우선순위가 높은 실시간 프로세스의 지연에 초점을 맞추고 그 문제점을 보완하기 위한 방법을 제안한다. 우선순위가 높은 실시간 프로세스는 원하는 결과를 빠른 시간 안에 얻어야 하므로 다른 작업에 의한 프로세스의 지연을 최소화 하여야 한다[2]. 제안하는 방법은 플래시 메모리의 라이브 페이지를 메인 메모리(main memory)의 일부에 저장했다가 사용하는 것으로 마치 하드 디스크의 데이터를 저장하는 커널(kernel) 메모리의 페이지 캐시(page cache)처럼 사용하므로 라이프 페이지 캐시 방법(live page cache method)이라 한다.

2장에서는 플래시 메모리 기반의 파일 시스템 및 파일 시스템 안에서 작동하는 기존에 제안되었던 가비지 컬렉션 방법에 대해서 알아본다. 3장에서는 제안하는 시스템의 구조 및 알고리즘을 설명하고 이론적으로 성능을 분석하여 본다. 마지막으로 4장에서는 본 논문의 결론을 설명하고 앞으로의 연구 방향을 제시한다.

2. 관련 연구

플래시 메모리의 파일 시스템에는 여러 가지 종류가 있고 그 안에서 작동하는 가비지 컬렉션 방법도 각기 다르다. 대표적인 플래시 메모리 파일 시스템에는 YAFFS[3], JFFS2[5], JFFS3[6] 등이 있다.

YAFFS는 낸드 플래시 메모리를 운영체제에서 사용 가능하게 하는 파일 시스템이다. YAFFS는 CPU가 쉬고 있을 때 데드 페이지로만 구성된 블록이나 단 하나의 라이브 페이지만을 갖고 나머지는 데드 페이지로 구성된 블록에 한해서 가비지 컬렉션을 수행한다[3]. 또한 플래시 메모리 내의 프리 페이지의 수가 일정 수준 이하로 내려가면 가비지 컬렉션을 수행한다. 따라서 프로세스 진행 중 가비지 컬렉션이 발생하게 되면 진행 중이던 프로세스는 지연되며 프로세스 종료 시점을 예측할 수 없다[2].

이러한 문제점을 해결하기 위해서 리핀 장(Li-Pin Chang) 등은 프로세스 도중 가비지 컬렉션이 발생하더라도 프로세스의 끝나는 시간을 예측할 수 있는 방법을 제시하였다[2]. 이 방법에서는 토큰(token)이라는 개념을 도입하였으며 토큰의 수는 사용 가능한 프리 페이지의 수이다. 프로세스는 항상 가비지 컬렉터(garbage collector)와 병렬적으로 실행되며 가비지 컬렉터가 프로세스 중간에 반복적으로 가비지 컬렉션을 수행하면서 토큰을 생성하여 사용 가능한 프리 페이지를 확보한다[2]. 이 방법은 프로세스의 종료 시점을 예측 가능하게 하지만 가비지 컬렉션의 수행 시간 자체를 단축하지는 못하였고 오히려 프로세스의 진행 시간을 더 길게 만드는 문제점이 발생할 수 있다. 따라서 플래시 메모리에서 실행되는 우선순위가 높은 실시간 프로세스의 가비지 컬렉션에 의한 지연 시간을 줄이기 위해서는 가비지 컬렉션 수행 시간 자체를 줄이는 방법이 필요하다.

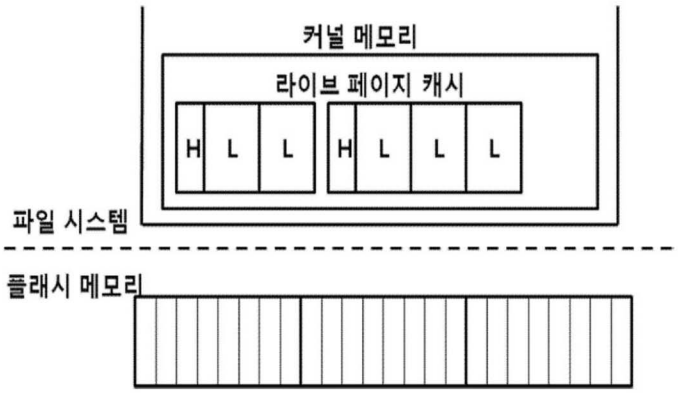
3. 라이브 페이지 캐시 방법(Live Page Cache Method)

3장에서는 1장과 2장에서 설명하였던 가비지 컬렉션의 문제점을 보완하기 위해서 본 논문이 제안하는 라이브 페이지 캐시 방법을 설명한다.

3.1 라이브 페이지 캐시의 개요 및 구조

라이브 페이지 캐시 방법은 우선순위가 높은 실시간 프로세스가 진행 중일 때 프로세스의 쓰기 작업에 의해 가비지 컬렉션이 발생하여 해당 프로세스가 지연되는 시간을 줄이기 위해 제안하는 방법이다. 기존의 많은 파일 시스템에서는 가비지 컬렉션이 발생 되면 이전에 발생하였던 읽기 작업과는 무관하게 가비지 컬렉션이 수행되었다. 가비지 컬렉션은 라이브 페이지를 다른 블록에 복사하는 작업을 포함하고 복사는 읽기 작업과 쓰기 작업을 수반하므로 각 라이브 페이지 당 1번의 읽기 작업과 1번의 쓰기 작업이 발생한다. 제안하는 라이브 페이지 캐시 방법은 이전에 읽기 작업이 발생한 블록과

그림 1. 라이브 페이지 캐시 저장 구조



플래시 메모리의 상태가 3.2절에서 설명하는 조건을 만족하게 되면 읽기 작업이 발생한 라이브 페이지들을 메인 메모리의 일부를 사용하여 저장함으로써 이후에 발생하는 가비지 컬렉션에 수반되는 읽기 작업을 줄일 수 있다. 본 논문에서는 라이브 페이지를 저장하는 메인 메모리의 일부를 라이브 페이지 캐시(live page cache)라 한다.

라이브 페이지 캐시는 그림 1과 같이 파일 시스템 안의 커널 메모리 안에 존재한다. 파일 시스템이 플래시 메모리의 라이브 페이지를 읽게 되면 해당 라이브 페이지와 블록의 정보는 라이브 페이지 캐시에 저장된다. 그림 1에서 라이브 페이지 캐시 내의 H는 블록의 정보를 저장하고 있는 헤더(header)이고, L은 라이브 페이지의 정보가 저장되어 있다. 헤더는 읽어온 라이브 페이지가 속해 있는 블록의 데드 페이지의 수, 플래시 메모리 상의 블록 주소, 그리고 읽어온 라이브 페이지의 수를 저장하고 있다. 헤더부터 다음 헤더가 나오기 이전까지 해당 블록의 정보와 소속된 라이브 페이지를 저장하며 이를 본 논문에서는 라이브 셋(live set)이라고 한다. 가비지 컬렉션의 효율을 높이기 위해 각각의 라이브 셋들은 헤더가 저장하고 있는 해당 블록의 데드 페이지 수에 따라서 내림차순으로 정렬되어 있다.

3.2 라이브 페이지 캐시 정책(live page cache policy)

3.2.1 캐시 메모리

라이브 페이지 캐시에 플래시 메모리의 데이터를 저장함으로써 그에 따른 메인 메모리에 부하가 걸릴 수 있다. 따라서 라이브 페이지를 저장하는 라이브 페이지 캐시의 양이 작아야 한다. 어느 정도의 양을 사용할 지는 시스템 환경에 따라서 달라질 수 있기 때문에 라이브 페이지 캐시 정책에서는 기존 파일 시스템의 페이지 캐시 용량의 10%로 정한다.

3.2.2 데드 페이지의 양

본 정책에서는 읽기 작업이 발생한 블록의 페이지 중 75% 이상이 데드 페이지여야 라이브 페이지들을 라이브 페이지 캐시에 저장하는 것으로 한다. 블록 내의 데드 페이지 양이 많을수록 프리 페이지를 많이 확보할 수 있기 때문에 데드 페이지가 많은 블록을 지우는 것이 가비지 컬렉션의 효율을 높인다. 따라서 모든 블록 내의 모든 페이지가 데드 페이지 일 때 가비지 컬렉션의 효율은 가장 높다. 하지만 블록 내의 모든 페이지가 데드 페이지 일 때는 읽기 작업이 발생할 수 없으므로 라이브 페이지 캐시 방법은 데드 페이지로 구성된 블록에 적용할 수 없다. 데드 페이지가 적은 블록을 지울 경우 가비지 컬렉션의 효율이 낮아지므로 그 효율을 고려하여 본 정책에서는 최소 75%가 되어야 가비지 컬렉션을 수행하는 것으로 정하였다.

3.2.3 라이브 페이지의 양

블록 내 전체 라이브 페이지의 일정 수준 이상이 읽혀져야 해당 블록을 가비지 컬렉션을 위한 블록으로 설정하도록 하며 그 일정 수준을 본 정책에서는 블록 내 전체 라이브 페이지의 75%로 한다. 라이브 페이지 캐시 방법은 파일 시스템이 블록의 모든 라이브 페이지를 읽어왔을 때 그 효율이 가장 좋고 반대로 블록의 라이브 페이지 중 적은 부분만 읽어온다면 그 페이지들을 라이브 페이지 캐시에 저장하여도 가비지 컬렉션 수행 시 소요되는 시간의 단축이 적다. 예를 들어, 플래시 메모리의 특정 블록 내에 6개의 라이브 페이지가 존재하고 한 개의 라이브 페이지만 읽혀진다면 그 페이지를 라이브 페이지 캐시에 저장하여도 가비지 컬렉션 발생 시 나머지 5개의 라이브 페이지에 대한 읽기 비용을 줄일 수 없다. 즉, 단 하나의 페이지의 읽기비용만 감소하게 된다. 이러한 경우에도 약간의 시간 단축은 있지만 한정되어 있는 라이브 페이지 캐시용량을 감안하였을 때 읽혀오는 라이브 페이지들을 모두 라이브 페이지 캐시에 저장하는 것은 좋지 않다. 따라서 본 정책에서는 최소한의 성능 보장을 위해서 블록 내의 전체 라이브 페이지의 75% 이상을 읽어올 때 라이브 페이지 캐시에 저장하는 것으로 정하였다.

3.2.4 블록 간 우선순위

가비지 컬렉션은 일반적으로 데드 페이지 수가 가장 많은 블록을 지우는 것이 웨어레벨링 측면에서 효율이 높다[3]. 따라서 라이브 페이지 캐시 내에도 다른 라이브 셋들 보다 데드 페이지 수가 더 많은 라이브 셋이 들어온다면 라이브 셋들 간의 우선 순위가 바뀌어야 한다. 또한 새로운 데이터가 라이브 페이지 캐시에 저장될 때 3.2.1절에서 정해놓은 라이브 캐시의 용량을 넘어간다면 데드 페이지의 수에 따른 우선순위가 가장 낮은 라이브 셋을 삭제하여야 한다. 알고리즘 1은 새로운 라이브 셋이 라이브 페이지 캐시에 입력될 경우 우선순위에 따라 라이브 페이지 캐시가 업데이트 되는 과정이다. 라이브 페이지 캐시에 새로운 데이터를 저장할 충분한 공간이 존재한다면 새로 읽어온 라이브 셋을 저장한다. 충분한 공간이 존재하지 않는다면 라이브 페이지 캐시 내의 가비지 컬렉션의 우선순위가 가장 낮은 라이브 셋의 헤더가 저장하는 데드 페이지 수와 새로운 라이브 셋의 데드 페이지 수를 비교하여 데드 페이지 수가 더 많은 라이브 셋을 저장하고 적은 라이브 셋은 삭제한다.

알고리즘 1. 라이브 페이지 캐시 업데이트 방법

```
1: 새로운 라이브 페이지들과 헤더로 이루어진 라이브 셋 입력
2: if (라이브 페이지 캐시 내에 새로운 라이브 셋을 저장할 충분
      한 공간 존재하지 않음)
3:    if (새로운 라이브 셋의 데드 페이지 수가 라이브 페이지 캐
         시내의 우선순위가 가장 낮은 라이브 셋의 데드 페이지
         수보다 큼)
4:       새로운 라이브 셋을 저장할 충분한 공간이 생길 때까지
         우선 순위가 낮은 라이브 셋을 삭제
5:    else
6:       새로운 라이브 셋을 저장하지 않고 알고리즘 종료
7: end if
8: 새로운 라이브 셋을 라이브 페이지 캐시에 저장
9: 라이브 셋의 데드 페이지 수에 따라 내림차순 정렬
```

각각의 라이브 셋의 헤더가 저장하는 데드 페이지 수에 따라서 내림차순으로 정렬한다.

3.2.5 업데이트 정책

기존에 라이브 페이지 캐시에 저장되어있던 라이브 페이지가 업데이트 된다면 해당 정보를 수정해 주어야 한다. 업데이트된 정보를 라이브 캐시에 저장해 둔다면 컴퓨터 전원이 나갈 경우 해당 정보를 소실할 수 있기 때문에 업데이트 발생 시 해당 정보를 다른 블록에 저장한 후 라이브 페이지 캐시에 저장된 해당 정보들을 삭제해야 한다.

3.3 자세한 방법

제안하는 방법에서는 읽기 작업이 수행된 플래시 메모리의 라이브 페이지를 라이브 페이지 캐시에 저장함으로써 가비지 컬렉션 시 발생하는 읽기 비용을 줄이는 것이 그 목적이다. 플래시 메모리의 라이브 페이지가 읽기 작업이 발생되면 해당 페이지들과 블록의 정보를 라이브 페이지 캐시에 저장한다. 그림 2는 파일 시스템이 플래시 메모리의 라이브 페이지를 읽어올 때 읽어온 라이브 페이지와 라이브 페이지가 속하는 블록의 정보가 라이브 페이지 캐시 내에 저장되는 과정을 보여준다. 그림 2에서 D는 데드 페이지이고, F는 프리 페이지, L은 라이브 페이지이다.

라이브 페이지 캐시에 저장된 라이브 페이지는 쓰기 작업이 발생되고 가비지 컬렉션이 필요할 때까지 라이브 페이지 캐시에 남아있는다. 만약 쓰기 작업이 발생되어 플래시 메모리에 데이터가 쓰여질 수 있는 프리 페이지가 남아있는 적당한 블록이 없다면 가비지 컬렉션이 발생하게 된다. 3.1절에서 설명하였듯이 가비지 컬렉션이 발생하면 일반적으로 블록 내에 남아있는 라이브 페이지를 다른 페이지에 복사하여야 하는데 복사는 각 페이지 당 1번의 읽기와 1번의 쓰기가 실행한다. 하지만 라이브 페이지 캐시 방법에서는 라이브 페이지 캐시에 저장되어있는 라이브 페이지들에 한해서 각 페이지 당 1번의 읽기 작업을 생략할 수 있다. 그림 3은 읽기

그림 2. 읽기 발생 시 라이브 페이지 캐시 갱신

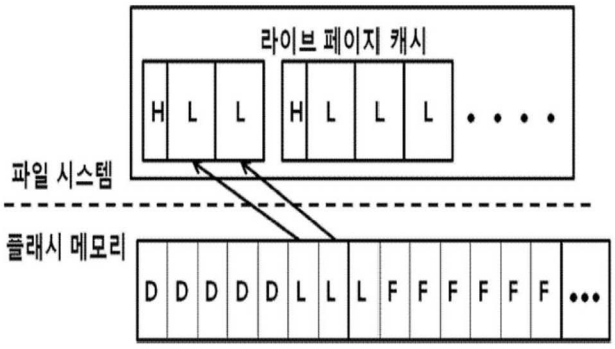


그림 3. 가비지 컬렉션 발생 시 복사 작업

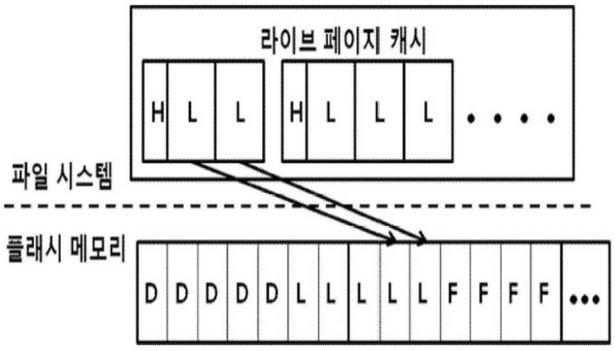
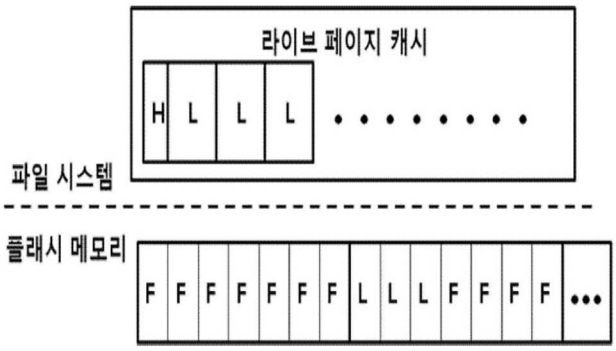


그림 4. 가비지 컬렉션 종료 후



작업을 생략하고 라이브 페이지 캐시에 저장되어 있는 라이브 페이지를 플래시 메모리에 복사하는 과정을 보여준다.

가비지 컬렉션이 발생하여 해당 블록의 라이브 페이지들을 다른 블록에 복사하였다면 그 블록은 삭제(erase)되어야 한다. 또한 라이브 페이지 캐시에 저장되어 있던 헤더와 라이브 페이지들도 삭제되어야 한다. 그림 4에서 라이브 페이지 캐시의 라이브 셋이 삭제되고 기존에 라이브 페이지가 저장 되어있던 블록이 지우기 작업이 수행되어 프리 페이지로 이루어진 블록이 생성됨을 확인할 수 있다.

3.4 이론적 성능 분석

지금까지 라이브 페이지 캐시 방법에 대해서 설명하였다. 3.4 절에서는 라이브 페이지 캐시 방법을 이용해

서 가비지 컬렉션 시 소용되는 비용이 얼마나 절감되는지를 이론적으로 분석해본다. 표 1에 나타난 읽기, 쓰기, 지우기 시간을 바탕으로 3.2 절의 라이브 페이지 캐시 정책에 해당되는 블록의 가비지 컬렉션 시 소용되는 시간을 일반적인 가비지 컬렉션 방법과 비교하여 본다.

Rt를 페이지 읽기에 소요되는 시간, Wt를 페이지 쓰기에 소요되는 시간, Et를 블록 지우기에 소요되는 시간, Lc를 블록 내 라이브 페이지의 수라 할 때 일반적으로 많은 파일 시스템에서 하나의 블록에 대한 가비지 컬렉션에 소요되는 시간은 식 (1)과 같다.

$$Lc * (Rt + Wt) + Et \quad (1)$$

라이브 페이지 캐시 방법에서는 이전에 읽기가 발생한 블록을 라이브 페이지 캐시에 저장한 후 가비지 컬렉션을 수행하므로 가비지 컬렉션 발생 시 라이브 페이지 캐시에 저장되어 있는 라이브 페이지에 대한 읽기 비용을 절약할 수 있고 식 (2)와 같다.

$$Lc * (Rt' + Wt) + Et \quad (2)$$

Rt'은 이전에 읽기 작업이 수행되지 않은 블록 내의 라이브 페이지로 Rt'은 블록 내의 모든 라이브 페이지인 Rt보다 작다.

기존의 가비지 컬렉션 방법 대비 라이브 페이지 캐시 방법의 시간을 계산해 보면 식 (3)과 같다.

$$(Lc * (Rt' + Wt) + Et) / (Lc * (Rt + Wt) + Et) \quad (3)$$

분자가 작을수록 식 (3)의 값이 작아지므로 Rt'가 0일 때, 즉 블록 내의 모든 라이브 페이지가 읽기 작업이 수행되어 라이브 페이지 캐시에 저장되어 있을 때 최고의 효율을 보인다.

예를 들어 블록 내에는 총 64개의 페이지가 있고, 라이브 페이지 캐시 정책의 데드 페이지 정책에 따라 75%가 데드 페이지라 하고 나머지 25%인 16개의 페이지가 라이브 페이지, 그 모든 라이브 페이지가 라이브 페이지 캐시에 저장되어 있다고 생각해 본다. 식 (1)과 (2)를 표 1에 나타난 페이지 읽기, 쓰기, 블록 지우기 시간을 넣어 계산을 해보면 기존의 가비지 컬렉션 방법은 6789.6μs의 시간이 소요되는 것에 비해 라이브 페이지 캐시 방법을 적용해 보면 5544.8μs의 시간이 소요된다. 이는 기존의 방법에 비해 81.6% 정도의 시간이 소요되는 것으로 라이브 페이지 방법이 기존의 가비지 컬렉션 방법에 비해 18.4% 정도의 시간을 절약할 수 있다는 것을 보여준다.

하지만 1절에서 설명했던 바와 같이 플래시 메모리의 지우기 횟수는 한정되어 있기 때문에 읽어온 라이브 페이지가 속하는 블록에 대해서 지우기를 수행하는 라이브 페이지 캐시 방법은 데드 페이지의 수가 가장 많은 블록을 지우는 기존의 가비지 컬렉션 방법에 비해서 웨어레벨링 측면에서 그 효율이 떨어질 수 있다.

4. 결 론

본 논문에서는 플래시 메모리를 기반으로 한 우선순위가 높은 실시간 프로세스 수행 중 발생하는 가비지 컬렉션의 시간적 효율성을 높이기 위한 라이브 페이지 캐시 방법을 제시하였다. 그 방법은 메인 메모리의 일부분에 읽기 작업이 발생한 페이지를 저장해둠으로써 가비지 컬렉션 발생 시 읽기 비용을 줄이는 것이다. 또한 이론적 분석을 통해 최대 얼마만큼의 시간적 비용을 절약할 수 있는지 알아 보았다.

본 논문에서 제안하는 라이브 페이지 캐시 방법은 읽기 작업이 수행된 특정 블록을 가비지 컬렉션 하므로 그리디(greedy) 방법을 바탕으로 데드 페이지가 가장 많은 블록을 가비지 컬렉션하는 기존의 방법과 비교하였을 때 웨어레벨링 측면에서는 효율성이 떨어질 수 있다.

차후에는 라이프 페이지 캐시 방법에 압축을 적용하여 페이지 정보를 압축하여 메인 메모리에 저장하는 방법을 연구해 볼 계획이다. 더 나아가 기존의 방법과 웨어레벨링 측면에서 효율이 뒤쳐지지 않는 그러한 방법을 찾아보고자 한다.

참고 문헌

[1] Dongwon Kang, Dawoon Jung, Jeong-Uk Kang, Jin-Soo Kim, “μ-Tree : An Ordered Index Structure for NAND Flash Memory”, EMSOFT’07, 2007

[2] Li-Pin Chang, Tei-Wei Kuo, “A Real-Time Garbage Collection Mechanism for Flash-Memory Storage Systems in Embedded Systems”, ACM Transaction on Embedded Computing Systems, Vol3, pages 837-863, 2004

[3] Wookey, Aleph One Ltd, Embedded Debian, “YAFFS A NAND-flash file system”

[4] Chin-Hsien Wu, Tei-Wei Kuo, Li Ping Chang, “An Efficient B-Tree Layer Implementation for Flash-Memory Storage Systems”, ACM Transactions on Embedded Computing Systems, Vol 6, Article 19, 2007

[5] David Woodhouse, Red Hat, Inc., “JFFS : The Journalling Flash File System”

[6] Artem B. Bityutskiy, “JFFS3 design issues”, 2005

[7] Eran Gal, Siven Toledo, “Algorithms and Data Structures for Flash Memories”, ACM Computing Surveys, Vol37, No.2, pp. 138-163, 2005

[8] Lee S.W, Park D.J, Chung T.S, Lee D.H, Park S.W, Song H.J, “A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation”, ACM Transactions on Embedded Computing Systems, Vol 6, No.3, Article 18, 2007