# 내장형 시스템에서 JFFS2 플래쉬 파일 시스템에 적합한 페이지 캐쉬 구조

송형근○ 차호정
연세대학교 컴퓨터과학과
{hksong,hjcha}@cs.yonsei.ac.kr

# A Page Cache Mechanism for JFFS2 Flash File System in Embedded Systems

Hyungkeun Song○ Hojung Cha
Dept. of Computer Science, Yonsei University

요 약

본 논문은 JFFS2 플래쉬 파일 시스템에 적합한 페이지 캐쉬 구조를 제안한다. JFFS2 플래쉬 파일 시스템은 공간 활용을 높이기 위해 데이터를 압축 저장하므로 기존 리눅스의 페이지 캐쉬가 효과적으로 사용될 수 있다. 그러나 멀티미디어 파일과 같이 비압축과 순차읽기 특성을 보이는 데이터는 플래쉬 메모리의 빠른 읽기 속도와 낮은 캐쉬 적중률로 인해 기존 페이지 캐쉬는 문제점을 보인다. 본 논문에서는 JFFS2 플래쉬 파일 시스템에서 사용하는 리눅스의 페이지 캐쉬를 기술하고 문제점을 분석한다. 그리고 기존 연구에서 제시된 저전력 소모를 위한 페이지 캐쉬 구조에 기반하여 throughput 향상을 위한 페이지 캐쉬 사용 기법을 제시하고 평가한다.

## 1. 서론

내장형 시스템에서 하드 디스크를 대체할 저장 매체로 플래쉬 메모리가 널리 이용되고 있다. 플래쉬 메모리는 하드 디스크와 달리 임의 접근 속도가 빠르고, RAM과 비교해 비휘발성이며 전력소모가 적은 장점이 있다. 플래쉬 메모리를 보조 기억장치로 사용하면 기존 디스크 기반에서 효율적으로 동작하는 운영체제의 기능들이 바뀌어야 한다. 리눅스와 같은 범용 운영체제는 블록 장치에서 데이터 읽기 시 느린 속도를 개선하기 위해 버퍼 캐쉬와 페이지 캐쉬를 사용하는데, 다중 사용자 환경에서 캐쉬 사용은 사용자 메모리 공간으로 원하는 데이터를 빠르게 전송하는 장점을 보인다. 그러나 플래쉬 메모리는 읽기 속도가 RAM에 근접할 만큼 빠르므로 캐쉬 사용 효과가 감소하며[1], 캐쉬 적중률에 따른 데이터의 특성을 고려하면 기존 캐쉬 구조는 적합하지 않다. 특히, 내장형 시스템에서 중요한 요소인 저전력 소모를 위해 Douglis[2]는 개선된 버퍼 캐쉬 구조를 제안했는데, 빈번히 접근되는 데이터는 캐쉬를 사용하고, 나머지 데이터는 캐쉬를 사용하지 않음으로써 전력 소모량을 감소시켰다. 이와 같이 기존 캐쉬 구조는 플래쉬 메모리 기반에서 저전력 소모와 성능상의 문제점을 보이므로 적합한 구조로 개선이 필요하다.

리눅스에서 제공하는 JFFS2[3]플래쉬 파일 시스템은 효율적인 저장 공간 활용을 위해 데이터를 압축 저장하므로 페이지 캐쉬가 효과적으로 사용 될 수 있다. 그러나 멀티미디어 데이터와 같이 사용자 수준에서 압축된 데이터는 대부분 압축 저장되지 않으므로 캐쉬 사용은 비효율적이다. 기존 연구[4]에서는 저전력 소모를 위해 압축 데이터와 비압축 데이터를 분류하여 압축 데이터만 페이지 캐쉬를 사용하는 기법을 제안하였고 향상된 결과를 보였다. 본 논문에서는 이와 같은 구조를 확장하여 저전력 소모에 적합한 캐쉬 구조 기반에서 through-put 향상을 위한 효율적인 페이지 캐쉬 구조를 제안한다. 2장에서는 리눅스에서 제공되는 RAM 캐쉬 구조의 문제점과 개선방향을 기술한다. 3장에서는 개선된 JFFS2 플래쉬 파일 시스템을 설명한다. 4장에서는 실험 및 결과를 기술하고, 5장에서 결론을 맺는다.

• 본 연구는 정보통신연구진흥원에서 지원하는 정보통신기초기술연구 지원사업으로 수행하였슴 (과제번호 : C1-2003-A1-2000-0240)

## 2. 기존 캐쉬 구조의 문제점과 개선 방향

리눅스와 같은 범용 운영체제에서는 주기억 장치인 RAM을 특정 크기의 페이지 프레임으로 관리한다. 일반적으로 페이지 프레임 크기는 4Kbytes이며, 커널은 페이지 속성 정보를 유지하여 각각의 페이지 프레임을 관리한다. 캐쉬는 블록 단위로 입출력이 수행되는 버퍼 캐쉬와 페이지 단위로 입출력이 수행되는 페이지 캐쉬로 분류할 수 있다[5]. 페이지 캐쉬는 버퍼 캐쉬의 집합으로 구성되고, 페이지 캐쉬 공간이 부족하면 커널은 페이지 교체 정책에 의해 특정 페이지를 회생시킨다. 페이지 교체 정책은 리눅스 버전 2.4.x 이상에서 Page aging 기법을 이용한 LRU 방법을 사용 한다. 커널이 사용하는 페이지를 제외한 모든 페이지는 자유 페이지가 부족할 때, 페이지 교체 정책에 의해 회생된다.

JFFS2 파일 시스템은 버퍼 캐쉬 사용 없이 플래쉬 메모리에서 페이지 캐쉬로 직접 데이터를 전송하는데, 압축된 데이터는 페이지 캐쉬로 풀어서 전송하므로 페이지 캐쉬 사용은 반복적인 압축 풀기 작업이 필요 없는 장점이 있다. 그러나 멀티미디어 파일과 같이 비압축과 순차읽기 특성을 보이는 데이터는 페이지 캐쉬에서 적중률이 낮으며 기존 적중률 높은 페이지를 회생시키는 문제가 발생하므로 페이지 캐쉬 사용은 빈번한 페이지 입출력에 따른 성능 저하와 불필요한 메모리 접근에 따른 전력 소모량을 증가시킨다. 그러므로 기존 페이지 캐쉬 시스템에서 무조건적인 데이터 캐싱 구조를 개선하기 위해 다음과 같이 데이터 특성을 분류한다.

- 압축 데이터와 비압축 데이터 : 파일 시스템 수준에서 데이터를 압축 저장하는 경우, 플래쉬 메모리에는 압축 데이터와 비압축 데이터가 존재한다. 사용자 수준에서 압축 된 데이터는 플래쉬 메모리에 비압축 데이터로 존재하게 되고, 나머지 데이터는 대부분 압축 데이터로 존재하게 된다. 비압축 데이터는 빠른 읽기 속도를 보이므로 저 전력 소모를 위해 페이지 캐쉬를 사용하지 않는 것이 유리하다.
- 순차접근과 임의접근 : 데이터는 순차접근과 임의접근의 특성을 보인다. 대부분 크기가 작은 파일은 임의접근 방식을 보이며, 대용량 파일은 순차접근 방식을 보인다. 순차접근 방식을 보이는 대용량 파일은 대부분의 저장 공간과 수행시간을 차지한다. Conquest 파일시스템[6]은 이와 같은 분류 기법을 사용하여 성능 향상을 보였다.

## 3. 개선된 JFFS2 플래쉬 파일시스템

논문에서 제시하는 플래쉬 파일 시스템의 전체적인 구조는 그림1과 같다. 페이지 캐쉬 사용 여부를 결정하는 Caching Filter와 Caching Filter의 기능을 지원하기 위해 개선된 JFFS2가 하나의 파일 시스템을 구성하여 동작한다. 구현 환경은 Montavis-ta에서 내장형 시스템을 위해 제공되는 리눅스 커널 버전 2.4.18을 사용했다.

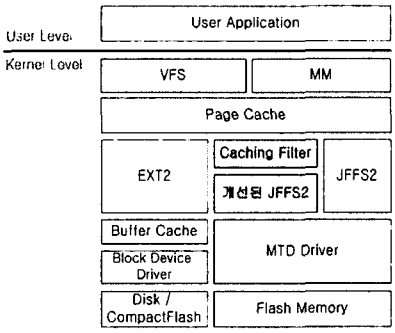


그림 2. 시스템 구조

### 3.1. Caching Filter

Caching Filter는 파일 읽기 시 페이지 캐쉬의 사용여부를 결정하는 기능을 수행한다. VFS 수준에서 파일 읽기 요구가 발생하면 JFFS2 파일 시스템은 데이터를 플래쉬 메모리로부터 주기억 장치로 전송하는데, Caching Filter는 데이터의 메타정보를 참조하여 주기억 장치의 사용자 메모리 또는 페이지 캐쉬로 전송하기 위한 결정을 한다.

캐쉬 결정 알고리즘은 압축 데이터와 비압축 데이터로 나뉜다., 그리고 압축 데이터와 비압축 데이터는 순차접근과 임의접근 특성을 보이는 데이터로 분류한다. 압축 데이터와 임의접근 특성을 보이는 데이터는 2장에서 언급한 바와 같이 페이지 캐쉬를 사용하고, 비압축 데이터와 순차접근 특성을 보이는 데이터는 페이지 캐쉬를 사용하지 않는 것이 바람직하다. 압축과 비압축 데이터 특성과 순차읽기와 임의읽기 특성을 반영하여 페이지 캐쉬 사용 여부를 결정하기 위해 표 2와 같이 분류하였다.

| 구분 | 압축 | 비 압축 |
|---|---|---|
| 임의 접근 | 사용 | PCB entry |
| 순차 접근 | PCB entry | 사용 않함 |

표 2. 페이지 캐쉬에 따른 데이터 분류

임의접근 특성을 보이는 압축 데이터는 페이지 캐쉬를 사용하지만 순차접근 특성을 보이는 비압축 데이터는 캐쉬를 사용하지 않는다. 순차접근 특성을 보이는 압축 데이터는 PCB entry에 의해 처리되는데 PCB entry는 최초 파일 읽기 시 데이터 정보를 기록하고 다음 접근 시 PCB entry에 해당 데이터가 존재하면 페이지 캐쉬를 사용한다. 이와 같은 방법은 순차접근 특성을 보이는 데이터에서 특정 부분에 빈번한 접근이 가능하므로 압축 처리에 따른 오버헤드를 최대한 줄이기 위해 사용된다. 비압축 데이터는 저전력 소모를 위해 페이지 캐쉬를 사용하지 않으나 빈번히 접근 되는 데이터는 페이지 캐쉬 사용으로 성능을 향상 시킨다. PCB entry 사용은 압축 데이터에서 사용한 방법과 달리 3단계 영역으로 구성하여 빈번히 접근될 가능성 높은 데이터를 분류 하였다. 그림 2는 PCB entry에서 압축

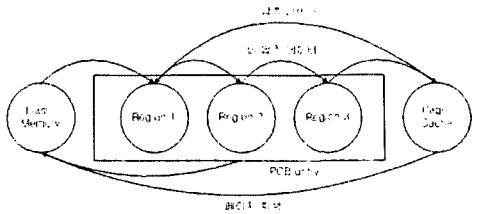

그림 2. PCB entry에서 데이터 이동 과정

데이터와 비압축 데이터가 처리되는 구조를 보인다. 압축 데이터는 Region 1에서 반복 읽기가 요구되면 Page Cache로 이동하지만, 비압축 데이터는 Region 1, 2, 3을 통해 이동한다. 그림 3은 구현된 PCB entry의 자료 구조를 보인다. 파일을 식별하기 위해 n개의 배열로 구성된 page buffer와 파일 내의 위치, 영역을 구별하기 위한 1bit의 index, 2bits의 region으로 구성된다. page buffer에 파일의 크기가 기록되어 index와 region 비트를 구성한다. 이와 같은 구조는 PCB entry 처리에 따른 오버헤드를 최대한 줄이기 위해 구현되었다.

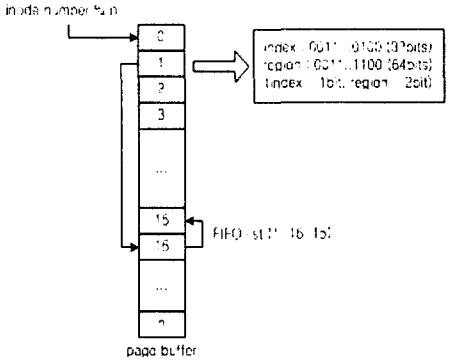


그림 3. PCB entry 구조

### 3.2. 시스템 구현

Caching Filter를 사용하기 위해 기존 JFFS2 파일 시스템을 개선하였다. 기존 JFFS2 파일 시스템은 사용자 프로그램이 read() 시스템 콜 호출시 커널 함수인 generic_file_read()를 호출한다. generic_file_read()는 리눅스에서 일반적인 파일 읽기를 위해 제공되는 함수로, read() 시스템 콜이 전달한 플래그에 따라 페이지 캐쉬를 사용하는 do_generic_file_read() 커널 함수와 사용자 메모리 공간으로 직접 데이터를 전송하는 generic_file_direct_IO() 커널 함수를 호출한다. 그러나 기존 JFFS2 파일 시스템은 generic_file_direct_IO()를 지원하지 않고, 커널 수준에서 페이지 캐쉬 사용 여부를 결정해야 하므로 무조건적인 페이지 캐쉬를 사용하는 do_generic_file_read() 함수의 페이지 캐쉬 사용 구조를 수정했다. 기존 do_generic_file_read() 함수는 JFFS2 파일 시스템의 jffs2_readpage 함수를 호출하여 페이지 캐쉬로 데이터를 읽고, 페이지 캐쉬에서 사용자 메모리 공간으로 데이터를 전송하나 수정된 do_generic_file_read() 함수는 Caching Filter에서 얻은 정보를 참조하여 페이지 캐쉬 혹은 사용자 메모리 공간으로 직접 데이터를 전송한다. 페이지 캐쉬 사용은 기존 jffs2_readpage를 이용하고 사용자 메모리 공간으로 데이터 전송은 do_generic_file_read()의 인자로 전달받은 buf의 주소값을 참조하여 플래쉬 메모리로부터 buf로 직접 데이터를 전송한다.

## 4. 실험 결과

실험은 Intel DBPXA250 개발 보드를 이용하여 측정했고, 시스템 구성은 표3과 같다. Intel DBPXA250 보드는 다양한 시스템 환경 설정이 가능하므로 코어 클럭 주파수를 400Mhz, SDRAM 주파수와 메모리 클럭 주파수를 100Mhz로 일정하게 설정하여 실험하였다. 그리고 플래쉬 메모리는 NOR 플래쉬를 사용하였다.

| 실험 환경 | |
|---|---|
| Processor | Intel PXA250 Applications processor |
| LCD | Sharp LM8V31 640x480 LCD panel |
| Memory | 64MB of SDRAM, 1MB SRAM |
| Flash | 32MB socketed Intel StrataFlash memory |

표 3. DBPXA250에서의 실험 환경

파일 시스템 성능 측정은 IOzone 벤치마크[7]를 사용하였고, 페이지 캐쉬 사용에 따른 파일 읽기와 사용하지 않을 때의 파일 읽기에서 throughput 차이를 관찰하였다. 그림 4, 5는 파일 읽기와 파일 다시 읽기에서 파일 시스템 수준의 압축, 비압축 파일에 대해 기존 JFFS2와 비교하여 보인다. 임계값과 PCB entry 크기는 1Mbytes로 설정하였다.

그림 4에서 비압축 파일 읽기와 임계값 이상의 압축파일 읽기는 개선된 JFFS2가 페이지 캐쉬를 사용하지 않음에 따른 RAM 접근횟수 감소로 높은 throughput을 보인다. 반면, 그림 5에서 파일 다시 읽기는 기존 시스템이 페이지 캐쉬를 사용하여 RAM에서 직접 데이터를 읽으므로 제안된 구조가 RAM보다 읽기 속도가 느린 플래쉬 메모리로부터 직접 데이터를 읽는 것에 비해 높은 throughput을 보인다. 특히 임계값 이상의 압축 파일에서 페이지 캐쉬를 사용하지 않으면 압축 풀기에 따른 오버헤드가 크므로 페이지 캐쉬 사용에 비해 약 5배 정도의 성능 감소를 보인다. 그러나 크기가 큰 파일은 순차읽기 특성을 보이므로 캐쉬 적중률이 낮고, 파일의 특정 위치에 빈번한 접근이 발생하면 Caching Filter에 의해 페이지 캐쉬가 사용되므로 적중률 높은 데이터는 페이지 캐쉬를 사용하고, 적중률 낮은 데이터는 페이지 캐쉬를 사용하지 않음으로써 성능 보상이 가능하다. 또한 비압축 파일은 기존 연구에서 보인바와 같이 저전력을 위해 페이지 캐쉬를 사용하지 않으므로 파일 다시 읽기에서 throughput이 기존 JFFS2에 비해 낮으나, Caching Filter에 의해 빈번히 접근되는 데이터가 분류되어 페이지 캐쉬를 사용함으로써 성능을 개선할 수 있다.

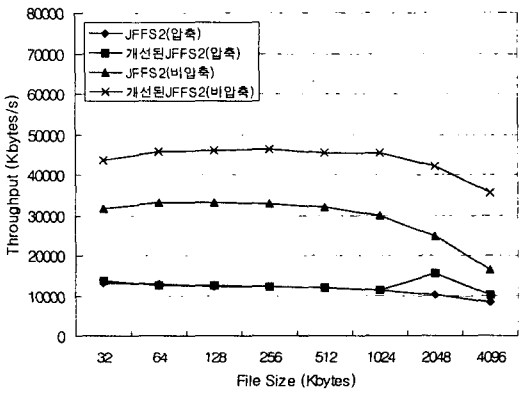


그림 4. 파일 읽기 throughput

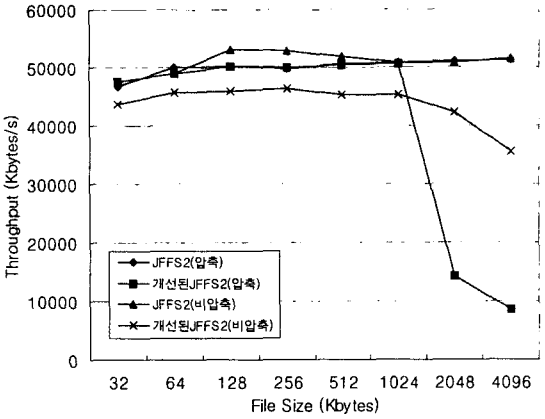


그림 5. 파일 다시 읽기 throughput

## 5. 결론

본 논문에서는 기존 페이지 캐쉬의 문제점을 분석하고 성능 향상을 위해 JFFS2 플래쉬 파일 시스템에 적합한 페이지 캐쉬 구조를 제안하고 구현하였다. 기본적으로 기존 연구에서 제시된 저전력 소모를 고려하였으며, 데이터를 특성에 따라 분류하여 선택적으로 페이지 캐쉬을 수행하는 기법을 사용하였다. 기존 JFFS2에서 사용되는 무조건적인 페이지 캐쉬 사용과 비교해 제안된 구조는 시스템 메모리 공간 절약과 적중률 높은 캐쉬 데이터를 보존할 수 있는 장점이 있다. 또한 RAM 접근 감소에 따른 성능이 향상됨을 실험을 통해 보였다. 본 논문에서는 간단한 실험을 통해 페이지 캐쉬 사용 여부에 따른 throughput 차이를 보였으나, 향후 과제로는 다양한 응용 프로그램을 수행하여 페이지 캐쉬 사용과 페이지 교체 정책에 따른 페이지 이동 상황을 모니터링 하여 실제 시스템을 분석하고, 저전력 소모를 위해 보다 향상된 알고리즘을 제시한다.

## 참고 문헌

[1] 박상호, 안우현, 박대연, 김정기, 박승민, "플래시 메모리를 위한 파일 시스템의 구현", 정보과학회논문지, 컴퓨팅의 실제 제7권 제5호, 2001, pp.402-415

[2] Fred Douglis, Frans Kaashoda, Brian Marsh, Ramon Caceres, Joshua A. Tauber, "Storage Alternatives for Mobile Computers", *USENIX 1994 Unix Operating Systems Design and Implementation Proceedings*, 1994, pp.25-37

[3] David Woodhouse, "JFFS : The Journalling Flash File System", *Ottawa Linux Symposium*, 2001.

[4] 송형근, 차호정, "저전력 시스템을 위한 선택적 페이지 캐쉬 사용 기법", 한국정보과학회 2003년 춘계학술발표대회 논문집, 2003년 4월.

[5] Daniel P.Bovet, Marco Cesati, *Understanding the LINUX KERNEL*, O'Reilly, December 2002.

[6] An-I A. Wang, Peter Reiher, and Gerald J.Popek, Geoffrey H. Kuenning, "Conquest: Better Performance Throug -h A Disk/Persistent-RAM Hybrid File System", *USENIX 2002 Annual Conference.*

[7] IOzone Filesystem Benchmark, URL : http://www.iozone .org