저전력 시스템을 위한 선택적 페이지 캐쉬 사용 기법

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

A Selective Usage of Page Cache towards Low-Power Systems

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

요 약

본 논문은 내장형 시스템에서 저전력 소모를 위한 선택적 페이지 캐쉬 사용 기법을 제안한다. 내장형 시스템의 저장매체로 널리 사용되고 있는 플래쉬 메모리는 데이터를 압축하여 저장하기 때문에 리눅스에서 사용되는 페이지 캐쉬가 효과적으로 동작한다. 하지만 플래쉬 메모리는 RAM 보다 전력 소모가 적기 때문에 페이지 캐쉬 사용에 따른 빈번한 RAM 접근 횟수는 전력 소모량을 증가시킨다. 따라서 저전력 시스템 운영을 위해서 페이지 캐쉬를 선택적으로 사용하는 것을 제안한다. 리눅스 운영체제상에서 구현된 시스템을 바탕으로 수행속도가 향상되고 전력 소모량이 감소함을 보인다.

1. 서론

플래쉬 메모리는 비휘발성이며 외부 충격에 강하고 전력소모가 적기 때문에 내장형 시스템에서 저장매체로 널리 이용되고 있다. 플래쉬 메모리는 회전식 자기 매체인 하드디스크와 달리 데이터를 읽는 속도가 RAM에 근접하며, 쓰기 속도가 읽기 속도보다 느리고 한번 데이터를 저장한 곳에 새로운 데이터를 저장 할 때에는 지움(Cleaning)과정[1]을 수행한 다음 저장해야 한다. 지움 과정은 여러 개의 블록들로 구성된 세그먼트 단위로 한번에 수행되며 지우는 시간은 쓰기 시간보다 느리고 지움 횟수도 제한되어 있다. 따라서, 기존 하드디스크 기반의 파일시스템은 플래쉬 메모리 특성에 맞게 바뀌어야 한다.

플래쉬 메모리를 위한 대표적인 파일 시스템으로 JFFS2[2]가 있다. JFFS2는 LFS기반의 파일 시스템이며 데이터 압축과 wear-leveling 기능을 제공한다. 데이터를 압축하여 저장하면 플래쉬 메모리 공간을 절약 할 수 있으나, 파일 읽기 시 매번 압축을 풀어야 하는 문제가 발생한다. 반면, 리눅스와 같은 운영체제에서 제공되고 있는 페이지 캐쉬를 사용하면 압축을 풀어야 할 필요가 없기 때문에 효과적이다. 페이지 캐쉬의 사용은 하드디스크와 같이 느린 응답속도를 나타내는 저장매체에서 효과적이다. 플래쉬 메모리의 경우, RAM과 유사한 읽기 속도를 갖고, 전력 소모도 적기 때문에 캐쉬의 효과가 감소하게 된다. Douglis[3]는 캐쉬의 크기가 커짐에 따라 나타나는 전력 소모량의 변화를 시뮬레이션 결과로 보여주었는데, 캐쉬 크기가 커짐에 따라 전력 소모량은 증가하지만 수행 속도에는 큰 영향을 미치지 못함을 알 수 있다. XIP(eXcute-In-Place)[2]는 저전력 소모를 위해 데이터를 RAM에서 읽지 않고 플래쉬 메모리에서 직접 읽는 기법이나 블록 단위의 데이터 읽기가 가능한 NAND 플래쉬에서는 적용하기가 부적합하다. 따라서 파일 시스템 변화와 함께 페이지 캐쉬의 사용도 플래쉬 메모리의 특성에 따라 바뀌어야 한다.

본 논문에서는 블록 단위로 데이터 읽기가 가능한 플래쉬 메모리에서 데이터 압축을 수행할 때 전력 소모량을 감소시키기 위한 선택적 페이지 캐쉬 사용 기법을 제안하고 리눅스에서 구현한다. 논문의 구성은 다음과 같다. 2장에서는 선택적 페이지 캐쉬 사용 제안과 구현을 기술한다. 3장에서 실험 결과를 기술하고, 4장에서 결론을 맺는다.

2. 선택적 페이지 캐싱

리눅스와 같은 범용 운영체제에서 read() 시스템 콜에 의해 파일 읽기를 하면 사용자 메모리 공간으로 데이터가 복사되기 전에 페이지 캐쉬로 복사된다. 페이지 캐쉬는 페이지 단위로 수행되며, 캐쉬 공간이 부족하면 페이지 교체 정책으로 LRU를 사용한다. 기존 시스템에서는 모든 파일 읽기에서 페이지 캐쉬를 사용한다[4]. 하지만, 본 논문에서는 특정 데이터에 대해서만 페이지 캐싱을 제안한다.

플래쉬 메모리에는 압축 데이터와 비압축 데이터 두 가지 특성을 갖는 데이터가 존재한다. 압축 데이터는 파일 시스템에 의해 압축 풀기를 수행해야 하기 때문에 사용자 메모리 공간으로 직접 복사가 불가능 하다. 반면에, 비압축 데이터는 직접 복사가 가능하다. 이러한 특성은 페이지 캐쉬를 선택적으로 사용 할 수 있게 한다. 압축 데이터는 매번 압축을 풀어야 하므로 수행 속도 저하와 전력 소모량을 증가 시킨다. 특히, 빈번하게 접근하는 데이터의 경우에는 수행 속도 지연으로 전체적인 시스템 성능이 저하된다. 따라서 페이지 캐쉬의 사용은 이 문제를 해결할 수 있다. 페이지 캐쉬는 논리적으로 파일시스템보다 상위 계층에 위치하기 때문에 압축이 풀린 상태로 데이터가 존재한다. 파일 읽기를 하면 데이터를 페이지 캐쉬에서 직접 복사 할 수 있으므로 압축풀기에 따른 지연시간과 전력 소모량을 줄일 수 있다. 비압축 데이터는 사용자 수준에서 이

미 압축된 데이터가 파일 시스템 수준에서 다시 압축할 때 압축되지 않는 데이터를 의미한다. 일반적으로 동영상과 같은 멀티미디어 데이터는 압축기술이 적용됐기 때문에 파일 시스템 수준에서 압축되지 않는다. 따라서, 페이지 캐쉬를 사용하지 않고 직접 사용자 메모리공간으로 복사할 수 있는데, 플래쉬 메모리의 빠른 읽기 특성으로 멀티미디어 데이터를 지연 없이 전송하는 것이 가능하다. 이와 같은 방법은 데이터 전송 지연에 따른 성능 저하를 최소화 하고, RAM 접근 횟수를 줄임으로써 전력 소모량을 감소시킨다. 텍스트와 같은 형태의 사용자 수준의 압축 파일도 비압축 데이터이고 압축을 풀기 위해 순차적으로 접근하는 특성 때문에 페이지 캐쉬를 사용하지 않는다.

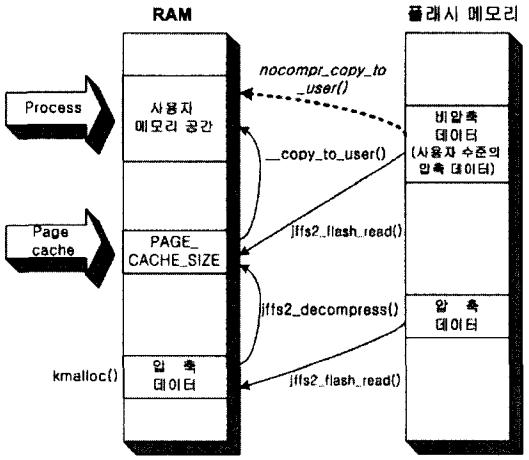

그림 1. 기존 페이지 캐쉬와 제안된 페이지 캐쉬 구조

그림 1 은 기존에 사용되는 페이지 캐쉬와 논문에서 제안하는 페이지 캐쉬 구조를 비교하여 보여준다. 기존 시스템에서는 압축 저장된 데이터를 jffs2_flash_read()를 통해 플래쉬 메모리에서 RAM 공간으로 복사한다. RAM에 복사된 압축 데이터는 jffs2_decompress()에 의해 압축이 풀리게 되고 페이지 캐쉬를 위해 빈 페이지를 할당 받은 후 저장된다. 캐쉬된 데이터는 사용자 응용프로그램 요구에 따라 __copy_to_user()에 의해 해당 페이지의 데이터를 사용자 메모리 공간으로 복사한다. 반면에 비압축 데이터는 페이지 캐쉬에 직접 복사한 후 사용자 메모리 공간으로 복사한다.

논문에서 제안하는 선택적 페이지 캐쉬는 비압축데이터에 대해서 페이지 캐쉬를 사용하지 않고 새로 구현된 nocompr_copy_to_user() 함수에 의해 플래쉬 메모리에서 사용자 메모리 공간으로 직접 복사한다. nocompr_copy_to_user() 함수는 사용자 메모리 주소 값을 인자로 갖고, 사용자 메모리 주소는 read() 시스템 콜에 의해서 전달된다. jffs2_flash_read() 함수에 의해 플래쉬 메모리에서 데이터를 읽게 되면 전달된 사용자 주소 공간의 시작 위치에서부터 데이터를 저장한다. 압축데이터와 유사하게 페이지 단위로 복사를 하고, 데이터 크기가 블록 크기 이하가 되면 해당 데이터만 복사하고 나머지 부분은 버리는 방법을 사용하였다. 그림 2 는 nocompr_copy_to_user() 함수 사용에 따른 파일 읽기 과정을 보여준다.

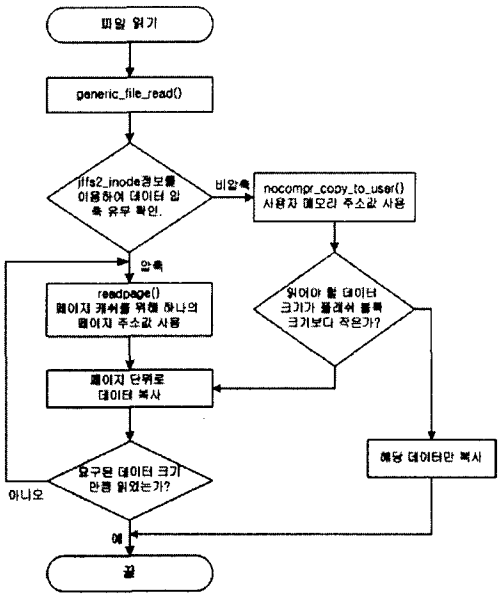

그림 2. 파일 읽기 과정

멀티미디어 데이터는 영상압축 기술이 적용되었기 때문에 파일 시스템 수준에서 압축하면 크기가 변하지 않는다. 하지만, MPEG과 같은 특정 파일의 경우 몇 바이트 정도만 변하는 형태로 여러 페이지에 분산되어 나타남을 실험을 통해 관찰 할 수 있었다. 기존 JFFS2에서는 데이터 크기가 변하지 않으면 원본데이터를 저장 하지만, 단지 몇 바이트 크기만 변해도 압축 데이터로 저장한다. 이와 같은 압축 형태는 저장 공간을 크게 절약할 수 없고, 데이터를 읽기 위해 압축 풀기에 따른 수행시간과 전력 소모량을 증가 시킨다. 따라서, 데이터가 압축되는 정도에 따라 압축여부를 결정하는 방법을 제안한다. 그림 3 은 구현된 압축 처리 과정을 보여준다.

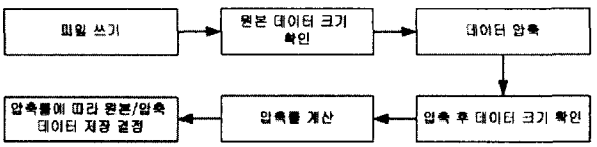

그림 3. 데이터 압축 수행 과정

압축률에 따른 데이터 저장을 결정함에 있어 기준이 되는 임계값을 설정해야 한다. 현재 구현에서 임계값은 원 데이터의 크기가 페이지 크기인 4Kbytes를 기준으로 50Bytes로 설정했다. 즉, 4Kbytes의 데이터를 압축 할 때, 압축 후 데이터의 차이가 50Bytes 미만의 값을 나타내면 압축을 하지 않는다. MPEG 파일의 경우 실험을 통해 많은 부분에서 압축 후 데이터 크기가 변하지 않거나, 1~50Bytes정도의 데이터 차이가 나타남을 관찰 할 수 있었고, 전체적인 크기에서도 1Mbytes당 수 Kbytes 미만의 데이터 차이를 보였다.

3. 실험

구현 환경은 리눅스 버젼 2.4.17과 JFFS2 플래시 파일 시스템을 사용하였고, Intel DBPXA250 개발 보드[5]에서 실험하였다. 개발 보드는 Intel PXA250 Applications Processor를 탑재하였고, 코어 클럭 주파수를 400Mhz로 설정하였다. 64MB 크기의 SDRAM과 플래쉬 메모리로 32MB Intel StrataFlash를 사용하였다. SDRAM 주파수와 메모리 클럭 주파수를 100Mhz로 설정하였다. 실험을 위한 데이터로는 MPEG 파일을 사용하였고, 선택적 페이지 캐쉬 사용과 파일 시스템 수준에서 선택적 데이터 압축 적용에 따른 수행 속도와 전력 소모량을 측정하였다.

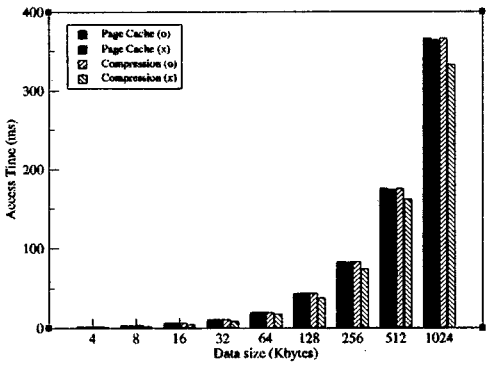

그림 4. 선택적 페이지 캐쉬 사용과 데이터 압축에 따른 읽기 속도

그림 4 는 MPEG 파일 읽기에서 데이터 크기에 따른 수행 속도를 보여준다. 페이지 캐쉬의 사용은 사용하지 않을 때와 비교해 RAM 접근 횟수가 증가되기 때문에 지연 시간이 생긴다. 1Mbytes 크기의 데이터를 읽을 때 RAM의 빠른 읽기 속도로 인해 그 차이가 적음을 볼 수 있지만, 데이터 크기가 계속 커짐에 따라 지연 속도는 비례해서 커진다. 사용자 수준에서 압축된 데이터를 파일 시스템 수준에서 압축하지 않고 저장하면 더 빠르다. 압축률이 낮은 데이터를 파일 시스템 수준에서 압축하면 해당 데이터에 대해 압축 풀기를 하기 때문에 지연 시간이 발생한다. 1Mbyte 읽기에서 약 10% 정도의 수행 속도를 향상 시켰고, 데이터 크기의 증가에 비례하여 압축률이 낮은 페이지가 발생할 가능성이 더 커지기 때문에 수행 속도 차이는 계속 커진다.

그림 5 는 논문에서 제안한 선택적 페이지 캐쉬와 압축률에 따른 데이터 압축을 적용했을 때 전력 소모량과 기존 시스템에서의 전력 소모량을 비교한 것이다. 전력 소모량 측정은 ScopeMeter Test Tool Kit인 Fluke 123을 사용하였으며, 1Mbytes 크기의 파일 읽기에 따른 전체 시스템 전력 소모량의 변화를 관찰하였다. 그림 5 의 진한 부분은 전력 소모량의 차이를 보여준다. 제안된 시스템은 기존시스템보다 RAM 접근횟수 감소와 압축 처리 지연이 없게 되어 전력 소모량이 감소한다. 330ms 이후에 기존 시스템은 파일 읽기를 계속 하지만, 제안된 시스템은 파일 읽기가 종료되어 전력 소모량의 차이가

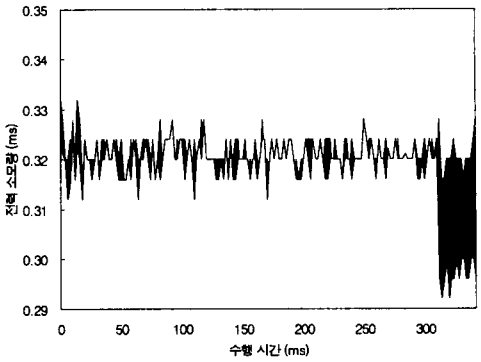

그림 5. 기존 시스템과 제안된 선택적 페이지 캐쉬와 데이터 압축을 적용했을 때의 전력 소모량

크게 나타난다. 전체적으로 기존 시스템 보다 약 10%의 전력 소모량이 감소됨을 알 수 있었다.

4. 결론

본 논문에서는 RAM과 플래쉬 메모리의 읽기 속도와 전력 소모량, 그리고 파일시스템 수준에서 압축을 고려한 선택적 페이지 캐쉬 사용을 제안하고 구현하였다. 비압축 데이터에 대해 페이지 캐쉬를 사용하지 않고, 페이지 캐쉬 사용을 줄이기 위해 사용자 수준에서 압축된 파일을 압축률에 따라 선택적인 압축을 제안하였다. 그 결과, 적은 크기의 저장 공간 낭비에 비해 데이터 읽기에서 저전력 소모와 수행 속도를 모두 향상시켰다.

실험 환경에서는 NOR 플래쉬를 사용했지만 대용량 멀티미디어 파일을 저장하는데 유리하고 블록 단위의 접근이 용이한 NAND 플래쉬에서 이와 같은 방법을 적용하여 저전력 소모를 기대할 수 있다.

참고 문헌

[1] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda, "A Flash-Memory Based File System", USENIX, pp.155-164, 1995

[2] David Woodhouse, "JFFS : The Journalling Flash File System", Proc. of Ottawa Linux Symposium and Technical Paper of RedHat Inc., 2001.

[3] Fred Douglis, Frans Kaashoda, Brian Marsh, Ramon Caceres, Joshua A. Tauber, "Storage Alternatives for Mobile Computers", Operationg Systems Design and Implementation, pp.25-37, 1994

[4] Daniel P.Bovet, Marco Cesati, "Understanding the LINUX KERNEL", O'Reilly, January 2001

[5] Intel DBPXA250 Development Platform for Intel Personal Internet Client Architecture, URL : http://www.intel.com/des-ign/pca/applicationsprocessors/manuals/278403-101.pdf, September 2002