SAF: 디스크 탐색 시간 향상을 위한 파일 시스템 내 스왑 공간 할당 기법

안우현* · 김보곤** · 김병규*** · 오재원****

SAF: A Scheme of Swap Space Allocation in File Systems to Reduce Disk Seek Time

Woo Hyun Ahn* · Bo-Gon Kim** · Byung-Gyu Kim*** · Jaewon Oh****

본 연구는 2011년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음

요 약

최근 고성능 컴퓨터에서 메모리 사용량이 매우 큰 프로그램과 파일 접근을 많이 하는 프로그램을 동시에 실행하고 있다. 많은 메모리의 사용은 디스크의 스왑 공간에 대해 디스크 접근을 빈번히 발생시키고, 파일 접근은 디스크의 파일 시스템 파티션으로 디스크 접근을 야기한다. 이런 두 종류의 프로그램을 동시에 실행하면 스왑 공간과 파일 시스템 파티션 간에 디스크 탐색이 빈번히 발생할 수 있다. 이런 문제를 해결하기 위해 본 논문은 SAF 기법을 제안한다. 이 기법은 파일 시스템 파티션에 새롭게 여러 개의 스왑 공간을 배치하고, 이들 공간에 페이지-아웃되는 페이지들을 저장한다. 즉, 페이지들은 가장 최근에 접근한 파일의 디스크 위치에 근접한 스왑 공간에 저장된다. 이 스왑 공간이 기존의 스왑 공간에 비해 최근 접근된 파일의 위치로부터 가까운 거리에 있기 때문에 파일 접근 후 발생하는 디스크 탐색의 시간을 크게 줄일 수 있다. 성능 검증을 위해 BSD기반의 FreeBSD 6.2 운영체제에 SAF를 구현하였고, 5개의 벤치마크를 실행하여 성능을 측정하였다. 성능 측정 결과 SAF는 FreeBSD에 비해 벤치마크의 실행 시간을 약 14%~42% 감소시켰다.

ABSTRACT

In recent computer systems with high-performance, users execute programs needing large memory and programs intensively accessing files simultaneously. Such a large memory requirement makes virtual memory systems access swap spaces in disk, and intensive file accesses require file systems to access file system partitions in disk. Executing the two kinds of programs at once incurs large disk seeks between swap spaces and file system partitions frequently. To solve the problem, this paper proposes a new scheme called SAF to create several swap spaces in a file system partition, where pages to be paged out are stored. When a page is paged out, the scheme stores the page to one of the swap spaces close to a disk location where the most recently accessed file is located. The chosen swap space in the file system partition is closer to the disk location than the traditional swap space, so that our scheme can reduce the large disk seek time spent to move to the traditional swap space in paging out a page. The experiment of our scheme implemented in FreeBSD 6.2 shows that SAF reduces the execution time of several benchmarks over FreeBSD ranging from 14% to 42%.

키워드

운영체제, 파일 시스템, 가상 메모리 시스템, 디스크, 스왑 공간

Key word

operating system, file system, virtual memory system, disk, swap space

* 정회원: 광운대학교
** 정회원: 삼성전자
*** 정회원: 선문대학교
**** 정회원: 가톨릭대학교(교신저자, jwoh@catholic.ac.kr)

접수일자 : 2011. 02. 15
심사완료일자 : 2011. 05. 04

Ⅰ. 서 론

최근 하드웨어 발전으로 하드 디스크 성능은 크게 개선되었다. 이런 디스크 성능 향상은 디스크 탐색(seek)과 회전 지연(rotational delay)으로 구성된 디스크 접근 시간을 줄였으나, 디스크 접근 시간은 CPU 처리 시간과 메모리 접근 시간에 비해 여전히 크다. 이런 디스크 접근은 파일 시스템을 접근하는 응용 프로그램의 실행 성능을 저하시킬 수 있다. 한편 메모리를 많이 사용하는 프로그램이 실행될 때 메모리가 부족할 수 있다. 이때 가상 메모리 시스템은 메모리에 있는 페이지를 디스크 스왑(swap) 공간으로 저장하는 페이지-아웃(page-out)을 수행하거나 스왑 공간에 있는 페이지를 메모리로 읽어오는 페이지-인(page-in)을 수행한다. 페이지-인이나 페이지-아웃이 빈번히 발생하면 디스크 접근 횟수가 증가하여 응용 프로그램의 성능을 저하시킨다.

파일 시스템과 가상 메모리 시스템에서 디스크 접근 시간을 감소시키기 위한 정책들이 연구되었다. 리눅스와 BSD 계열의 파일 시스템인 FFS(Fast File System)[1,2]에서는 한 파일의 여러 블록을 묶어서 한 번에 디스크에 저장하는 클러스터링 기법이 있다. LFS 파일 시스템[3]은 새로 생성되거나 변경된 여러 파일들을 모은 후 빈 공간에 한 번의 디스크 쓰기로 저장함으로써 파일 쓰기를 위한 디스크 I/O 횟수를 줄였다. 가상 메모리 시스템에서는 메모리에 있는 페이지들을 효과적으로 교체하여 디스크 I/O를 감소시켰다[4 -6]. 페이지 클러스터링 기법[7,8]은 페이지-아웃되는 페이지들을 묶어서 한 번에 디스크로 저장하여 디스크 접근을 최소화하였다.

하드웨어 발전으로 인해 고성능 컴퓨터의 사용자는 메모리를 많이 사용하는 시뮬레이터, 게임 등의 메모리 접근 집중 프로그램을 실행하면서 동시에 파일 접근을 많이 하는 소프트웨어 개발 툴, 멀티미디어 활용 툴 등의 파일 접근 집중 프로그램을 실행하기도 한다. 이때 메모리 접근 집중 프로그램이 메모리를 많이 사용해서 메모리가 부족할 수 있으며, 이로 인해 가상 메모리 시스템은 스왑(swap) 공간에 대해 페이지-아웃과 페이지-인 동작을 빈번히 발생시킬 수 있다. 반면에 파일 시스템은 파일 접근 집중 프로그램이 요청하는 파일 접근을 위해 디스크의 파일 시스템 공간(또는 파티션)을 접근한다. 이런 두 종류의 프로그램을 동시에 수행하면 스왑 공간과 파일 시스템 파티션 사이에 디스크 탐색이 빈번히 발생할 수 있다. 이는 파일 시스템과 가상 메모리 시스템의 성능을 저하시키며, 나아가 프로그램의 성능을 감소시킨다.

본 논문은 파일 접근 집중 프로그램과 메모리 접근 집중 프로그램이 동시에 수행될 때 스왑 공간과 파일 시스템 영역 간에 빈번하게 발생하는 디스크 탐색을 줄이는 SAF 기법을 제안한다. 이 기법은 기존의 스왑 공간 외에 별도로 파일 시스템 파티션에 여러 개의 스왑 공간을 배치한다.

페이지 아웃되는 페이지가 디스크로 저장될 때 가장 최근에 접근한 파일의 디스크 위치와 인접한 스왑 공간에 페이지를 저장한다. 파일을 접근한 후 디스크 헤드는 그 파일이 있는 디스크 위치 또는 그와 인접한 위치에 놓여진다. 이 위치와 인접한, 파일 시스템 파티션에 있는 스왑 공간에 페이지-아웃될 페이지를 저장하면 디스크 헤더는 최근 접근된 파일의 위치로부터 가까운 거리만을 탐색하면 된다. 따라서 파일 접근 후 페이지를 저장할 때 SAF는 기존의 스왑 영역에 저장하는 방법에 비해 디스크 탐색 시간을 크게 줄일 수 있다.

성능 검증을 위해 SAF를 FreeBSD 6.2 커널에 구현하였다. FreeBSD는 Unix, Linux 등 상업용 운영체제의 파일 시스템으로 널리 활용 중인 Fast File System(FFS)을 사용한다. 파일 시스템과 가상 메모리 시스템의 성능 측정을 위해 여러 벤치마크를 실행하였다. 실험 결과 SAF는 FreeBSD에 비해 벤치마크의 실행 시간을 약 14%～42% 단축시켰다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 FreeBSD 커널의 파일 시스템과 가상 메모리 시스템을 서술하고, Ⅲ장에서 연구 동기를 설명한다. Ⅳ장에서는 SAF의 개념, 동작 원리 및 구현을 기술한다. Ⅴ장에서 실험 결과를 분석하고, Ⅵ장에서 관련 연구를 기술한다. 마지막으로 Ⅶ장에서 결론을 도출하여 정리한다.

Ⅱ. 배경 지식

많은 상업용 운영체제의 파일 시스템은 FreeBSD처럼 BSD 계열 운영체제의 파일 시스템인 FFS[1,2]를 기반으로 설계되었으며, FFS는 리눅스 파일 시스템 ext2의 선조(ascendent) 시스템이기도 하다. FFS는 한 디렉토리의 파일들 및 한 파일의 블록(block)들을 참조할 때 발생할 수 있는 디스크 탐색을 줄이기 위해 디스크를 실린더 그룹 단위로 나눈다. 실린더 그룹은 디스크 탐색 시간을 무시할 수 있는 인접한 실린더들의 묶음으로 구성되며, 동일 디렉토리의 파일들과 이것과 관련된 inode 등의 메타데이터를 같은 실린더 그룹에 할당한다. 아울러 한 파일에 포함되는 여러 블록들과 이 파일의 inode들을 동일 실린더 그룹에 저장한다. 그래서 동일 디렉토리의 파일들을 순차적으로 참조하거나 한 파일의 블록들을 순차적으로 참조할 때 해당 실린더 그룹 내에서 서로 인접한 파일 블록들을 접근하기 때문에 디스크 탐색이 발생하지 않는다.

FreeBSD의 가상 메모리 시스템[2,7]은 페이지 부재(page fault) 횟수의 감소를 위해 메모리에 적재된 페이지들 중 적절한 페이지를 선택하여 교체시킨다. 이 페이지 교체는 빈 페이지가 부족할 때 또는 주기적으로 실행되는 페이지 데몬(page daemon)에 의해 수행된다. 그런데 페이지 교체 시 대상 페이지가 더티(dirty 또는 변경) 페이지인 경우 페이지-아웃이 수행되어 디스크 스왑 공간으로 저장된다. 스왑 공간은 전체 디스크 공간에서 디스크 주소가 가장 낮은 부분, 즉 가장 앞쪽 부분에 생성된다. 페이지 아웃이 수행되면 스왑 공간에서 빈 공간을 할당하고, 스왑 블록(swap block)에 페이지 아웃될 페이지의 정보를 기록한다. 마지막으로 스왑 공간에 페이지를 저장한다. 이때 스왑 블록은 스왑 공간에 저장된 페이지의 디스크 주소를 기록해 두는 메모리 블록이다. 페이지를 스왑 공간에서 해제할 때 스왑 블록에서 해당 페이지의 정보를 변경한다.

FreeBSD의 파일 시스템과 가상 메모리 시스템은 각각 파일 시스템의 공간과 스왑 공간을 독립적으로 관리한다. 파일 시스템은 실린더 그룹 맵(cylinder group map)을 통해서 해당 실린더 그룹에 포함된 모든 파일 블록들의 할당 정보를 관리한다. 이 실린더 그룹 맵은 비트맵(bitmap)으로 구성되며, 비트맵의 각 비트는 실린더 그룹에 있는 블록의 할당 상태를 나타낸다. 한 개의 블록을 실린더 그룹에 할당할 때 실린더 그룹 맵의 비트맵에서 빈 공간을 검색하여 할당한다. 아울러 실린더 그룹 맵은 디스크에 저장되는, 파일 시스템의 메타데이터(metadata)이다. 가상 메모리 시스템은 스왑 공간에 있는 모든 블록을 라딕스 트리(radix-tree)[7]를 사용하여 관리한다. 라딕스 트리는 32비트 비트맵을 가진 노드(node)들을 사용하며, 비트맵의 각 비트는 스왑 공간에 있는 4KB 크기의 블록을 지시한다.

Ⅲ. 연구동기

일반적으로 컴퓨팅 시스템에서 파일 접근 집중 프로그램과 메모리 접근 집중 프로그램이 동시에 수행되는 경우가 적지 않다. 리눅스와 FreeBSD 등과 같은 상용 운영체제는 일반적으로 스왑 공간을 파일 시스템을 위한 파티션이 아닌 별도의 파티션에 생성하여 관리한다. 이런 방식으로 스왑 공간을 관리하는 FreeBSD 6.2 운영체제에서 파일 접근 집중 프로그램과 메모리 접근 응용 프로그램을 동시에 수행할 때 디스크 탐색 시간의 분석을 위해 디스크 헤드(disk head)의 움직임을 관찰하였다. 실험 환경은 인텔 펜티엄 CPU 3.00GHz, 512MB의 메모리, Seagate 160GB 디스크가 탑재된 컴퓨터를 사용하였고, 스왑 공간은 시스템 기본 설정인 931MB를 따랐다. 또한 파일 접근 집중 프로그램인 Postmark[9]와 메모리 접근 집중 프로그램인 SOR[10]을 동시에 실행하였다.

그림 1은 두 개의 응용 프로그램을 동시에 실행할 때 발생하는 디스크 접근의 패턴을 나타낸다. 가로축은 디스크 접근 순서를 나타내며, 전체 디스크 접근 중에 500,000~510,000번째 범위의 디스크 접근을 나타낸다. 세로축은 스왑 공간과 파일 시스템 영역을 포함한 전체 디스크의 섹터(sector) 번호를 나타낸다. 이 중에서 스왑 공간은 섹터 번호 1,000,000에서 2,000,000까지의 디스크 영역에 속하고, 스왑 공간의 앞부분은 루트(root) 파티션이다. 나머지는 파일 시스템 영역에 포함된다. 실험 결과는 두 개의 프로그램을 동시에 실행하면 스왑 공간과 파일 시스템 영역을 빈번하게 번갈아 접근하고 있음을 보여준다.

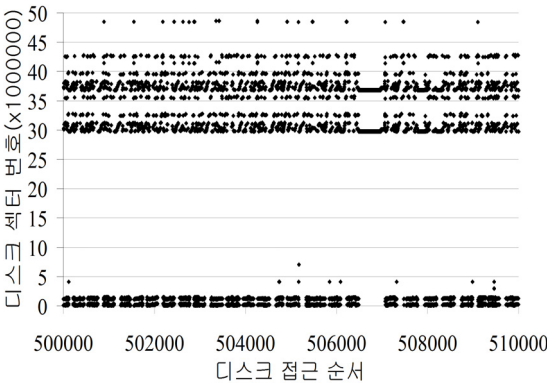

그림 1. 디스크 헤드 움직임
Fig. 1 Disk head movement

이는 스왑 공간으로 페이지를 저장하는 디스크 접근과 파일 시스템 영역으로 파일 데이터를 저장하는 디스크 접근이 번갈아서 일어나기 때문이다. 이들 두 디스크 영역이 디스크 상에서 거리가 멀고 빈번히 번갈아 접근하게 되면 큰 디스크 탐색 시간이 자주 발생한다. 따라서 이런 빈번한 디스크 탐색 시간은 파일 시스템과 가상 메모리 시스템의 성능을 저하시키며, 이는 응용 프로그램의 성능을 크게 감소시킨다.

Ⅳ. SAF 기법

4.1. 기본 개념

파일 접근 집중 프로그램과 메모리 접근 집중 프로그램이 동시에 수행될 때 스왑 공간과 파일 시스템 영역 간에 빈번한 디스크 탐색이 발생할 수 있다. 이런 문제를 해결하기 위해 본 논문은 페이지-아웃되는 페이지를 기존 스왑 공간에 저장하는 대신 파일 시스템 영역에 저장하는 SAF 기법을 제안한다. 이 기법은 기존 스왑 공간과 별도로 파일 시스템 영역 내 실린더 그룹에 스왑 공간을 새롭게 배치하고, 이 스왑 공간에 페이지-아웃되는 페이지를 저장한다. 실린더 그룹에 있는 스왑 공간을 cgSwap 공간이라고 정의한다. 더티 페이지가 페이지 교체로 인해 디스크로 저장될 때 가장 최근에 접근한 파일이 위치한 실린더 그룹에 있는 cgSwap 공간에 페이지를 저장한다. 디스크 헤드는 파일을 접근한 후에 접근했던 파일의 위치 또는 그와 인접한 위치에 놓여진다. 하지만 시스템 부팅 후 파일 시스템 영역에 있는 파일을 접근한 적이 없다면 페이지-아웃되는 페이지를 기존의 스왑 공간에 저장한다.

디스크 헤드가 위치한 실린더 그룹의 cgSwap 공간에 페이지-아웃될 페이지를 저장하면 디스크 헤드는 최근 접근된 파일의 위치로부터 거의 무시할 수 있는 거리만을 탐색하면 된다. 따라서 파일 접근 후에 기존의 스왑 영역에 저장하는 것에 비해 디스크 탐색 시간을 크게 줄일 수 있다.

그림 2는 SAF가 파일 시스템의 디스크 영역에 어떻게 페이지를 저장하는지 보여준다. Fi는 파일, Pi는 페이지를 나타낸다. 우선 초기 상태에 한 블록으로 구성된 네 개의 파일 F1, F2, F3, F4가 디스크에 저장되어 있다. cgSwap은 두 개의 블록으로 구성되어 있고, 각 실린더 그룹의 뒷부분에 할당되었다고 가정한다. 또한, 시스템 부팅 후에 아직 파일 시스템 영역에서 파일 접근이 일어나지 않았다고 가정한다.

그림 2(a)는 시스템 부팅 후 처음으로 페이지-아웃이 일어날 때의 동작 과정을 나타낸다. 두 개의 페이지 P1, P2가 페이지 아웃을 요청할 때 가상 메모리 시스템은 기존 스왑 공간에 페이지를 저장한다. 그림 2(b)는 파일 입출력이 일어나고 있을 때 페이지가 저장되는 동작을 보여준다. 실린더 그룹 CG10에 있는 파일 F1을 갱신한 후에, 페이지 P3을 페이지-아웃하면 최근 파일 접근이 발생한 CG10의 cgSwap에 페이지 P3을 저장한다. 그림 2(c)는 실린더 그룹 CG11로 파일 입출력 동작이 이동한 상황을 보여준다. CG11에 존재하는 파일 F4를 접근한 후, 페이지 P4를 저장하고자 한다. 이때 최근 파일 접근이 발생한 CG11의 cgSwap 공간에 P4 페이지를 저장한다. 그림 2(d)처럼 다시 CG11에서 파일 F3을 접근하고 페이지 P5를 페이지-아웃하면 그 페이지를 CG11에 저장한다.

SAF 기법은 페이지를 저장하기 위해 파일 시스템 영역 내 각 실린더 그룹에 연속적인 공간을 할당하며, 그 공간을 cgSwap 영역이라고 정의한다. 실린더 그룹에 cgSwap 영역을 생성하는 시기는 다음과 같다. 파일 시스템을 생성할 초기에 정적으로 생성하는 대신 각 실린더 그룹에 페이지가 처음 저장될 때 동적으로 생성한다. cgSwap의 크기가 너무 크면 파일 시스템을 위한 공간이 부족할 수 있다.

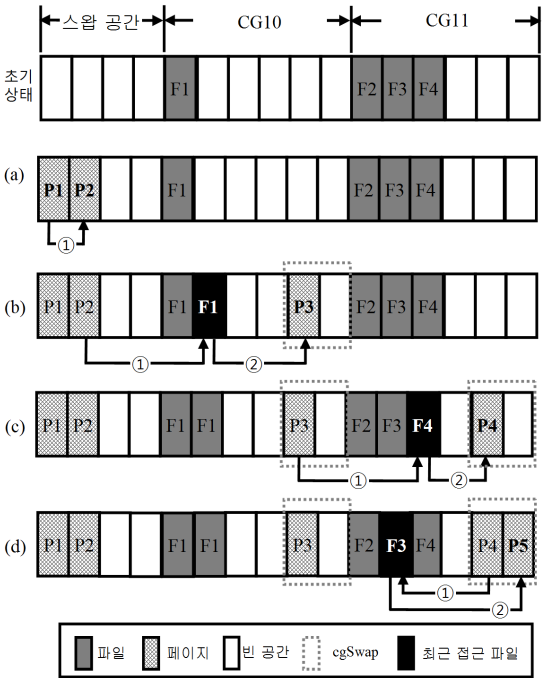

그림 2. SAF의 기본 동작 과정
Fig. 2 Basic operations of SAF

반면에 cgSwap 공간의 크기가 너무 작으면 cgSwap 공간을 빈번하게 생성해서 시스템 부하가 발생한다. 따라서 본 논문에서는 실험을 통해 가장 성능이 좋은 32MB를 기본 확장 공간 크기로 사용한다.

4.2.단일 파일 접근에 대한 다중 페이지 저장

메모리 사용의 요구가 크게 증가하여 최근 한 번의 파일 접근이 발생한 후에 한 개가 아닌 많은 페이지들이 짧은 시간에 페이지-아웃될 수 있다. 가장 최근에 파일 접근이 일어난 실린더 그룹 Ci에 cgSwap 공간이 있지만 페이지를 저장할 충분한 공간이 없고, 더 이상 cgSwap 공간을 생성하지 못할 수 있다. 이때는 Ci에 인접한 다른 실린더 그룹 Cj에서 페이지를 저장할 공간을 찾아서 저장하게 된다. 하지만, Cj 내 cgSwap 공간에 페이지를 저장하는 도중에, 메모리 접근 집중 프로그램이 할당받은 메모리를 해제함으로써 최근 파일 접근이 발생한 실린더 그룹 Ci의 cgSwap 공간에 빈 공간이 생길 수 있다.

만일 Ci의 이 빈 공간에 후속 페이지를 저장한다면 방금 전 페이지를 저장했던 실린더 그룹 Cj에서 Ci로 디스크 탐색이 발생한다. 더욱이 Ci의 이 빈 공간에 페이지를 저장한 후 더 이상 Ci에 페이지를 저장할 공간이 없다면 그 다음 페이지를 인접한 다른 실린더 그룹의 cgSwap 공간에 저장하게 된다. 이런 동작이 빈번히 발생하면 실린더 그룹 간에 디스크 탐색이 많이 발생하여 성능이 저하될 수 있다.

한 번의 파일 접근 후에 많은 페이지를 저장할 때 야기되는 빈번한 디스크 탐색을 줄이기 위해 SAF 기법은 효율적인 빈 공간 할당 정책을 수행한다. 우선 최근 파일 접근이 발생한 실린더 그룹의 cgSwap 공간에서 빈 공간을 검색한다. 만일 빈 공간이 있으면 페이지를 저장한다. 그렇지 않으면 cgSwap 공간이 할당된 시점이 최근인 것부터 오래된 순으로 순차적으로 cgSwap 공간을 검색한다.

이는 가장 최근에 할당된 cgSwap 공간에 빈 공간이 가장 많을 가능성이 높기 때문이다. 이처럼 빈 공간이 많은 cgSwap 공간에 후속 페이지들을 저장하면 이들 페이지를 여러 실린더 그룹에 있는 cgSwap 공간에 분산하여 저장하지 않아도 된다. 따라서 빈번히 발생할 수 있는 디스크 탐색을 최소화할 수 있다. 하지만 만일 최근에 할당된 cgSwap 공간에 빈 공간이 부족하면 cgSwap의 할당 시점이 최근인 것부터 오래된 순서로 cgSwap 공간을 검색한다.

그림 3은 SAF의 단일 파일 접근 후에 다중 페이지를 저장할 때 빈 공간을 할당하는 과정을 보여준다. 최근 파일 접근이 실린더 그룹 CG10에 발생하였으며, 페이지를 저장할 빈 공간이 없다고 가정한다. 또한 6개의 cgSwap 공간이 CG10, CG11, CG12에 흩어져 위치하며, 1→2→3→4→5→6의 순서대로 cgSwap 공간이 할당되었다고 하자. 실린더 그룹 10에 빈 공간이 없기 때문에 가장 최근에 할당된 CG12의 cgSwap 6에서 빈 공간을 검색하여 페이지를 저장한다.

cgSwap 6에 여러 페이지를 저장하는 도중에 cgSwap 1, 2, 3에 빈 공간이 생기더라도 cgSwap 6에 이 페이지들을 저장한다. 만일 cgSwap 6에 빈 공간이 부족하게 되면 할당된 시점이 최근인 것부터 오래된 순서인 5→4의 순으로 cgSwap 공간을 검색한다. 만약 CG12 내 모든 cgSwap 공간의 빈 공간이 고갈되면 할당된 시점이 최근인 것부터 오래된 순서, 즉 3→2→1의 순서로 cgSwap 공간을 검색한다.

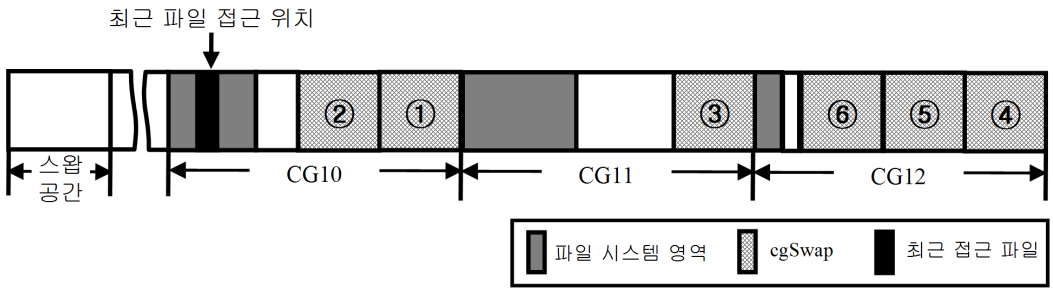

그림 3. cgSwap 공간 할당
Fig. 3 Allocation of cgSwap space

4.3. cgSwap 공간의 할당

페이지를 저장할 실린더 그룹에 cgSwap 공간이 할당되지 않았거나 할당되어 있더라도 더 이상 페이지를 저장할 공간이 없을 수 있다. 이런 상황에서는 파일 시스템 영역에서 cgSwap 공간을 할당할 실린더 그룹을 결정해야 한다. 우선 cgSwap 공간을 할당할 실린더 그룹을 결정하기 위해 최근 파일 접근이 발생한 실린더 그룹에 cgSwap 공간만큼의 연속적인 빈 공간이 있는지 검사한다. 만일 있다면 해당 실린더 그룹에 cgSwap 공간을 할당한다.

이는 페이지를 저장할 때 디스크 탐색 시간을 최대한 줄일 수 있는 위치가 최근 파일 접근이 발생한 실린더 그룹이기 때문이다. 이런 할당 정책에 따라 실린더 그룹에 한 개 이상의 cgSwap 공간이 할당될 수 있다. 만약 cgSwap 공간만큼의 빈 공간이 없으면 실린더 그룹 번호를 증가시켜가며 파일 시스템 파티션의 끝까지 각 실린더 그룹에 cgSwap 공간만큼의 빈 공간이 있는지를 검사한다. 하지만, 파티션 끝까지 검사했지만 cgSwap 공간을 할당하는데 실패할 경우에는 기존 스왑 공간을 이용한다.

페이지를 저장할 실린더 그룹이 결정되었다면 그 실린더 그룹 내에서 cgSwap 공간을 위한 연속적인 빈 공간을 할당한다. 이를 위해 실린더 그룹의 뒷부분부터 검색한다. 그 이유는 다음과 같다. 실린더 그룹 기반의 파일 시스템은 실린더 그룹 내에서 빈 공간을 얻기 위해 실린더 그룹의 앞에서부터 검색하는 최초 적합 블록 할당(first-fit block allocation) 기법을 사용한다. 이러한 블록 할당 기법으로 인해 실린더 그룹의 앞부분 공간은 대부분 할당되는 반면에, 뒷부분에 연속적인 큰 빈 공간이 있을 가능성이 높다. 기존 연구[11]에 따르면 디스크의 빈 공간의 절반이 실린더 그룹 뒷부분의 20%에 위치해 있음을 보고하고 있다. 그렇기 때문에 파일 시스템에서 사용 빈도가 낮은 실린더 그룹의 뒷부분부터 검색함으로써 cgSwap 공간을 위한 검색을 빠르게 수행할 수 있다. 아울러 파일 시스템의 사용 빈도가 낮은 영역을 활용할 수 있다.

4.4. cgSwap 공간을 관리하는 정보

SAF 기법은 FreeBSD가 스왑 공간을 관리할 때 사용하는 라딕스 트리를 사용하여 cgSwap 공간 내에 있는 빈 공간을 관리한다. 실린더 그룹에 cgSwap 공간이 할당되면 그 공간을 위한 라딕스 트리를 생성한다. 이 라딕스 트리는 기존 스왑 공간의 라딕스 트리와 병합되지 않고 cgSwap 공간별로 독립적으로 유지된다. 실린더 그룹에 페이지를 저장할 때 이 실린더 그룹의 cgSwap 공간을 위해 관리되는 라딕스 트리를 통해 페이지를 저장할 빈 공간을 검색한다.

그림 4는 cgSwap 공간을 관리하기 위한 자료 구조를 보여준다. 각 cgSwap 공간별로 한 개의 라딕스 트리를 둔다. 한 개의 실린더 그룹에 있는 여러 cgSwap 공간들의 라딕스 트리를 관리하기 위해 각 실린더 그룹별로 라딕스 트리 리스트(list)를 둔다. 아울러 모든 실린더 그룹들의 라딕스 트리 리스트를 관리하기 위해 cgSwap 리스트를 둔다. cgSwap 리스트의 각 엔트리(entry)는 실린더 그룹의 라딕스 트리 리스트를 지시한다.

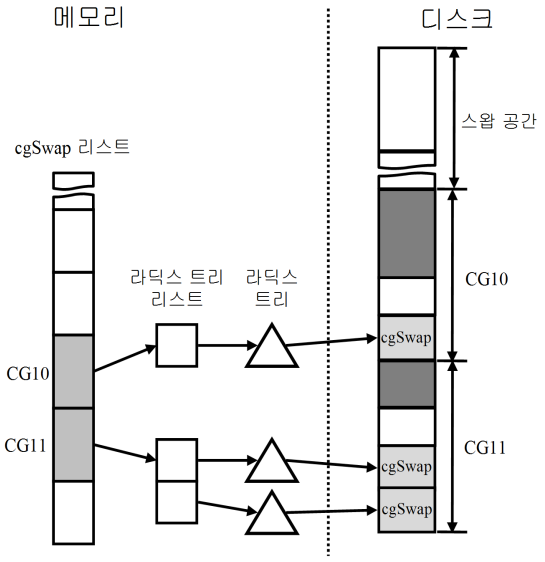

그림 4. cgSwap 공간 관리를 위한 자료 구조
Fig. 4 Data structure for managing cgSwap space

실린더 그룹에 cgSwap 공간을 할당한 후에는 파일 시스템이 그 공간에 파일 데이터를 저장하지 못하도록 해야 한다. 그렇지 않으면 cgSwap 공간에 저장된 페이지가 삭제되거나 미리 할당된 빈 공간만큼 페이지들을 저장할 수 없게 된다. 이를 위해 SAF 기법은 cgSwap 공간을 실린더 그룹에 할당할 때 이 실린더 그룹의 실린더 그룹 맵을 변경한다. 즉, 실린더 그룹에 cgSwap 공간을 할당하면 실린더 그룹 맵에서 cgSwap 공간에 대응하는 블록들의 비트를 할당 상태로 지정한다. 아울러 이 cgSwap 공간을 위한 라딕스 트리를 생성하여 페이지를 저장할 빈 공간의 관리를 시작한다. 결국 파일 시스템은 cgSwap 공간을 이미 사용 중인 상태로 인식하지만 SAF 기법은 페이지를 저장하기 위해 cgSwap의 라딕스 트리를 검색하여 빈 공간을 사용한다.

시스템이 재부팅되면 일반적으로 가상 메모리 시스템에서 관리되는 모든 메타데이터와 스왑 공간에 저장된 모든 페이지는 소멸된다. 마찬가지로 시스템이 재부팅되면 SAF 기법이 메모리에서 관리하는 cgSwap 리스트, 라딕스 트리 리스트 및 라딕스 트리는 소멸한다. 하지만 cgSwap 공간의 할당 상태를 나타내는 실린더 그룹 맵은 디스크에 저장되기 때문에 자동적으로 해제되지 않는다. 그래서 재부팅되면 실린더 그룹에서 맵 정보를 수정하여 할당된 cgSwap 공간을 해제해야 한다. 이런 해제 동작을 시스템 부팅 과정에 시도할 수 있다. 하지만, 부팅 시에 모든 실린더 그룹을 검색하여 부팅 전에 할당된 cgSwap 공간을 검색해야 한다. 또한 검색되면 해당 실린더 그룹의 실린더 그룹 맵을 변경한 후에 디스크로 다시 저장해야 한다. 이런 동작은 CPU 부하와 디스크 접근 횟수를 크게 증가시켜서 부팅 시간을 매우 증가시킬 수 있다.

SAF 기법은 시스템 부팅 시간이 증가하지 않도록 cgSwap 공간의 해제 동작을 부팅 과정에 수행하지 않는다. 대신 부팅 후 각 실린더 그룹별로 페이지 아웃이 처음 발생할 때 해당 실린더 그룹에 있는 기존 cgSwap 공간을 새롭게 할당할 cgSwap 공간으로 재사용한다. 만일 실린더 그룹에 다수의 cgSwap 공간이 있다면 실린더 그룹에서 마지막에 위치한 한 개를 제외한 나머지 cgSwap 공간을 모두 해제한다. 이 해제는 실린더 그룹 맵에서 cgSwap 공간에 대응하는 블록들을 빈 공간 상태로 변경함으로써 이루어진다. 이때 변경된 실린더 그룹 맵을 다시 디스크로 저장한다. 아울러 새로운 cgSwap 공간을 위해 초기화된 라딕스 트리를 생성한다. 따라서 시스템 부팅 시에 모든 실린더 그룹에 있는 cgSwap 공간들을 검색하여 해제하는 방법에 비해 부팅 시간 및 디스크 접근을 줄일 수 있다.

V. 실험

5.1 실험 환경

SAF 기법의 성능 검증을 위해 Intel Pentium D CPU 3.00GHz, DDR2 512MB 메모리, Seagate ST3160812AS 160GB HDD가 탑재된 컴퓨터를 사용하였고, FreeBSD 6.2 버전의 운영체제를 설치하였다. 파일 블록 크기는 16KB이고, 스왑 파티션의 크기는 931MB로써 시스템 기본 설정을 따랐다. 실험에 사용된 파일 접근 집중 프로그램과 메모리 접근 집중 프로그램은 다음과 같다.

Postmark[9]: 파일 시스템의 성능을 측정하는 대표적인 파일 접근 집중 벤치마크(benchmark) 프로그램이며, 작은 파일들의 읽기, 쓰기, 추가 및 삭제 동작을 수행한다. 초기에 200개의 디렉토리를 생성하고, 디렉토리당 25,000개의 파일을 대상으로 순차적으로 쓰기 동작을 수행한다. 그 후 트랜잭션(transaction) 단계에서 랜덤하게

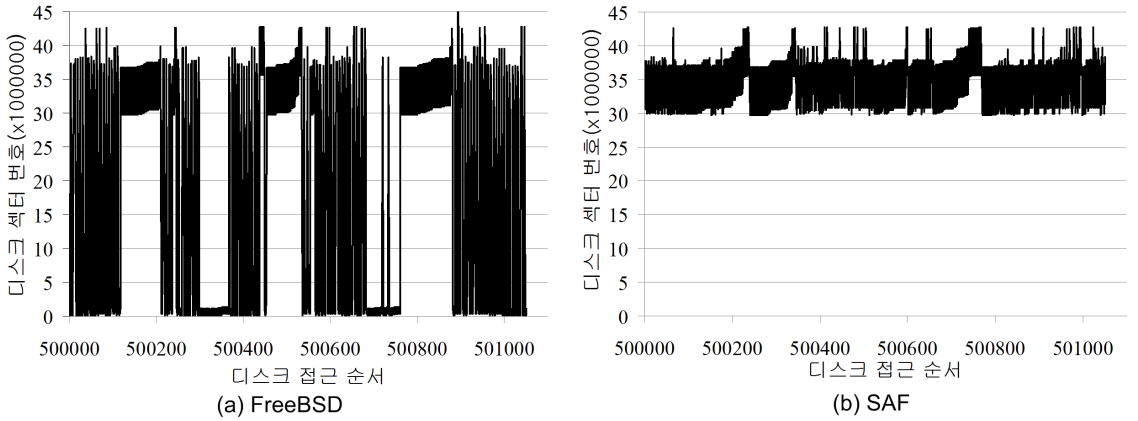

그림 5. Postmark 및 SOR 실행 시의 디스크 헤드 움직임
Fig. 5 Disk head movement in the execution of Postmark and SOR

디렉토리를 선택하여 파일 읽기, 쓰기, 추가, 삭제를 한다. 이때, 파일 생성 단계와 트랜잭션 단계의 파일 쓰기 비율은 25,000 대 50,000 비율로 트랜잭션 단계에서 더 많은 파일 쓰기가 일어난다.

커널 컴파일: 커널의 컴파일 동작은 기존에 생성된 FreeBSD 커널 코드의 오브젝트 파일을 제거하고 커널의 C소스 파일에 대한 오브젝트 파일을 다시 생성하는 순서로 구성된다. 이 동작은 파일 접근 집중적이며, 커널 컴파일 중에 파일 시스템은 여러 실린더 그룹을 참조하는 패턴을 보인다.

NS2[12]: NS2는 TCP, 라우팅 프로토콜, 멀티캐스트 프로토콜 등 다양한 인터넷 프로토콜을 시뮬레이션 하는 도구이다. NS2는 메모리 접근 집중 프로그램이며, 페이지 아웃을 충분히 발생시킬 수 있도록 시뮬레이션 샘플 파일(wireless-pkt-demo.tcl)에서 노드의 개수를 490으로 설정하였다.

Scimark2[10]: Scimark2는 과학적 수치 컴퓨팅을 위한 벤치마크이며, 메모리 접근 집중 프로그램이다. 총 다섯 가지의 과학 연산을 통해 시스템 성능을 측정한다. 실험에서는 SOR 알고리즘을 수행하였다. SOR 입력 배열의 크기는 파일 접근 집중 프로그램의 실행 시간에 따라 5600x5600과 7000x7000 두 종류를 사용하였다. 이는 동시에 실행되는 파일 집중 프로그램의 실행 시간만큼 SOR가 함께 실행되도록 메모리의 사용량을 조정하기 위해서이다.

nbench 2.2.2[13]: BYTE 메거진의 BYTEmark 벤치마크 중에 Linux/Unix용 버전 2이다. 이 벤치마크는 시스템의 CPU, FPU, 메모리 시스템의 성능을 측정하기 위해 주로 사용된다. 실험에서는 메모리 접근 집중 프로그램인 LU 알고리즘을 수행하였다. LU의 입력 배열 크기는 6300x6300으로 설정하였다.

5.2 실험 결과

본 실험에서는 파일 접근 집중 벤치마크인 Postmark, 커널 컴파일에서 한 개를 선택하고, 메모리 접근 집중 프로그램인 NS2, SOR, LU 중에서 한 개를 택하여 동시에 실행하였다. 또한 다섯 개의 벤치마크를 동시에 실행하는 실험을 수행하였다. 성능 측정을 위해 두 개의 척도를 사용하였다. 첫째, 파일 접근 집중 벤치마크와 메모리 접근 집중 벤치마크의 동시 실행으로 야기되는 디스크 헤드의 동작을 분석하여 먼 거리의 디스크 탐색이 얼마나 빈번히 발생하는지를 관찰하였다. 둘째, 파일 접근 집중 벤치마크와 메모리 접근 집중 벤치마크를 동시에 실행시킨 후 모든 벤치마크들의 실행이 종료될 때까지 걸리는 시간을 사용하며, 이를 실행 시간이라고 정의한다.

그림 5는 FreeBSD와 SAF에서 Postmark 한 개와 5600x5600 배열 크기의 SOR 두 개를 동시에 수행했을 때의 디스크 헤드의 움직임을 나타낸다. 두 개의 SOR를 동시에 실행한 이유는 메모리 사용량을 증가시켜 페이지-아웃을 매우 빈번히 발생하도록 하기 위해서이다. 이

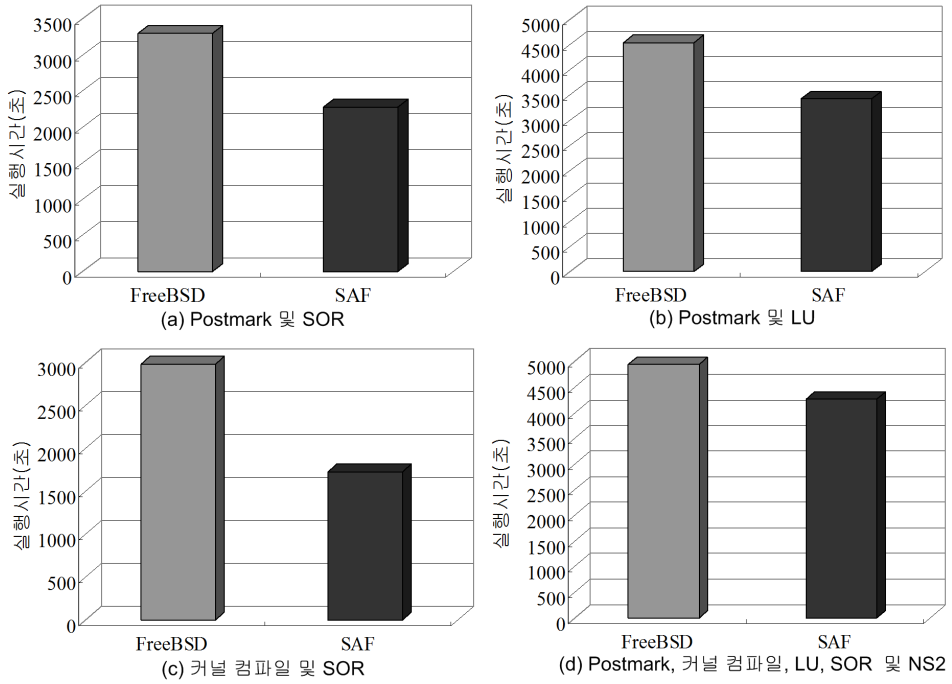

그림 6. 벤치마크의 실행 시간
Fig. 6 Execution time of several benchmarks

실험에서 파일 시스템에서 일어나는 파일 읽기와 쓰기, 가상 메모리 시스템에서 발생하는 페이지-인과 페이지-아웃에 대한 디스크 헤드의 접근 위치를 관찰하였다. 그림의 가로축은 500,000번째의 디스크 접근부터 1000개의 디스크 접근을 나타내며, 세로축은 접근된 디스크 섹터의 번호를 나타낸다. 이때 섹터 번호 1,000,000에서 2,000,000까지의 디스크 영역은 스왑 공간을 나타내고 나머지는 파일 시스템 영역을 나타낸다.

그림 5에서 가로축의 실선은 디스크 탐색의 발생을 나타낸다. 즉, 실선은 i번째 디스크 접근과 i+1번째 디스크 접근을 직선으로 연결한 것이다. FreeBSD에서 빈번히 발생하는 긴 실선이 SAF에서는 나타나지 않는다. 그 이유는 다음과 같다. FreeBSD에서는 Postmark이 접근하는 실린더 그룹과 이 실린더 그룹과 물리적으로 먼 거리에 있는 스왑 공간 사이를 디스크 헤드가 매우 빈번히 번갈아 움직인다. 반면에 SAF는 Postmark이 접근하는 실린더 그룹에 스왑 공간을 생성하여 페이지-아웃되는 페이지를 저장한다. 따라서 두 벤치마크가 동시에 실행되어 발생하는 긴 디스크 탐색의 횟수를 많이 감소시켰다. 이런 긴 디스크 탐색 횟수의 감소는 그림6(a)처럼 파일 시스템과 가상 메모리 시스템의 성능을 향상시켜서 벤치마크의 실행 시간을 FreeBSD에 비해 약 31% 단축시킬 수 있다.

그림6(b)는 Postmark 한 개와 LU 두 개를 동시에 수행할 때의 실행 시간을 나타낸다. SAF는 FreeBSD에 비해 실행 시간을 약 25% 감소시켰다. 이를 통해 SAF에 구현된, 페이지를 위한 공간 할당 기법이 FreeBSD의 가상 메모리 시스템에서 사용되는 할당 기법보다 디스크 접근 시간을 향상시켰음을 알 수 있다. 이 성능 개선은 두 종류의 접근 집중 프로그램이 동시에 실행되어 스왑 공간으로 페이지 접근이 매우 활발할 때 디스크 헤드가 기존 스왑 공간으로 이동하는 대신 파일 시스템 파티션 내 스왑 공간으로 이동하도록 하여 먼 거리 디스크 탐색의 횟수를 크게 감소했기 때문이다.

Postmark이 아닌 다른 파일 접근 집중 프로그램을 수행할 때에도 SAF 기법이 우수함을 검증하기 위해 커널

컴파일 한 개와 SOR 두 개를 동시에 수행시켰다. 그림 6(c)는 커널 컴파일 한 개와 5600x5600 배열 크기의 SOR 두 개를 동시에 수행시켰을 때의 실행 시간을 나타낸다. 실행 결과 SAF는 FreeBSD에 비해 실행 시간을 약 42% 감소시켰다.

서로 다른 종류의 프로그램을 동시에 실행할 때 FreeBSD에서 긴 디스크 탐색이 심각하지 않아 SAF의 성능 개선이 크지 않을 수 있다. 이런 실험을 위해 그림 6(d)처럼 Postmark, 커널 컴파일, LU, 7000x7000 배열 크기의 SOR, NS2를 각 한 개씩 동시에 실행할 때의 실행 시간을 측정하였다. 실험 결과 FreeBSD에 비해 SAF는 실행 시간을 약 14% 감소시켰으나, 앞에 나타난 다른 실험의 성능 개선에 비해 작다.

Ⅵ. 관련연구

리눅스와 윈도우즈와 같은 상용 운영체제에서 사용하는 스왑 공간 할당 기법[14,15]은 크게 두 가지로 나눌 수 있다. 하나는 파일 시스템의 영역에 한 개의 대용량의 스왑 파일을 생성하고, 페이지-아웃되는 페이지를 이 파일에 저장한다. 이 방법의 장점은 스왑 공간을 구현하기에 용이하며, 필요할 때 언제든지 스왑 공간의 크기를 변경할 수 있는 확장성이 있다는 것이다. 하지만, 페이지 아웃이 발생할 때마다 가상 메모리 시스템이 파일 시스템의 코드를 사용하여 스왑 파일의 디스크 상의 위치를 검색해야 하기 때문에 CPU 부하를 증가시킬 수 있다. 또한 대용량의 스왑 파일로 인해 이 파일에서 페이지를 저장할 빈 공간을 검색하기 위해 디스크에 있는 흩어져 있는, 여러 개의 관련 파일 메타데이터를 접근할 필요가 있다. 이는 다수의 디스크 접근을 일으켜서 페이지-아웃 시간을 증가시킨다. 그래서 위와 다른 스왑 공간 할당 기법으로 리눅스에서는 페이지-아웃의 성능을 증가시키기 위해 스왑 파일 대신에 스왑 공간을 별도의 디스크 파티션으로 생성하도록 권고하고 있다.

리눅스 운영체제에서는 한 개의 디스크에 여러 개의 스왑 공간을 생성할 수 있다[15]. 파일 시스템 파티션과 스왑 공간 간의 디스크 탐색의 최소화를 고려하여 사용자는 시스템 설정 시에 여러 개의 스왑 공간을 생성한다. 사용자는 성능을 고려하여 이들 스왑 공간에 우선순위(priority)를 배정한다. 페이지-아웃할 페이지를 우선순위가 높은 스왑 공간부터 우선 저장한다. 이 스왑 공간에 빈 공간이 없으면 다음 우선순위의 스왑 공간에 페이지를 저장한다. 하지만 디스크 탐색을 줄이기 위해 운영체제가 자동적으로 여러 개의 스왑 공간을 최적의 디스크 위치에 생성하는 것이 아니라 사용자가 직관적으로 설정해야 한다. 만일 부적절하게 설정되면 스왑 공간과 파일 시스템 파티션 간의 디스크 탐색을 효과적으로 줄일 수 없다. 또한 파일 접근 집중 프로그램과 메모리 접근 집중 프로그램을 동시에 실행하면 선택된 스왑 공간으로부터 거리가 먼 파일 시스템의 파티션에서 파일들이 여전히 빈번이 접근될 수 있다. 이는 디스크 탐색을 빈번히 발생시킬 수 있다.

디스크 탐색 시간을 줄이기 위한 대표적인 기법으로 FFS 파일 시스템은 파일 블록을 생성하면 이 블록이 이 블록이 속하는 파일의 기존 블록들과 디스크에 연속적으로 위치하도록 저장하는 클러스터링(Clustering) 기법[2,8]을 사용한다. 이 기법은 연속적으로 배치된 파일의 블록들을 디스크로 쓸 때 메모리에 일시적으로 저장한 후 이 블록들을 단일 디스크 쓰기(single disk write)로 한 번에 저장한다. 그 결과 디스크의 대역폭의 활용도를 증가시켜 디스크 탐색 시간을 감소시킨다. 또한 LFS 파일 시스템[3]은 새로 생성되거나 변경된 여러 파일들을 메모리에 모은 후 빈 공간에 한 번의 디스크 쓰기로 저장한다. 따라서 디스크 쓰기 성능이 개선되어 탐색 시간이 감소한다. 그러나 디스크가 조각화되어 큰 빈 공간이 없을 경우에는 빈 공간을 확보하기 위해 CPU 부하가 매우 큰 클리닝(cleaning) 동작을 수행한다. 이 클리닝 부하로 인해 파일 시스템으로는 거의 사용되지 않는다.

다중 프로세스, 다중 사용자 환경의 컴퓨팅 시스템에서 디스크 탐색 시간을 줄이기 위해 FS2 파일 시스템[16]이 제안되었다. FS2는 디스크 접근이 자주 일어나는 여러 실린더 그룹의 묶음을 활발한 지역(hot region)이라 정의한다. 그리고 활발한 지역과 다른 지역과의 디스크 접근이 빈번하게 발생하면 다른 지역에 있는 데이터를 활발한 지역 내로 복사(replication)한다. 이를 통해 FS2는 디스크 접근 시간을 크게 개선시킨다. 그러나 FS2는 디스크 블록의 배치 상태에 따라 성능이 좌우되고, 다수의 복사본이 생성될 수 있어서 디스크 공간의 낭비가 발생할 수 있다.

Ⅶ. 결 론

본 논문은 파일 접근 집중 프로그램과 메모리 접근 집중 프로그램이 동시에 수행될 때 스왑 공간과 파일 시스템 파티션 간에 발생하는 디스크 탐색을 줄이는 SAF 기법을 제안하였다. 이 기법은 파일 시스템 파티션에 여러 개의 스왑 공간을 생성하고, 페이지 아웃되는 페이지를 가장 최근에 접근한 파일의 디스크 위치와 인접한 스왑 공간에 저장한다. 이런 페이지 저장은 파일 접근 후 페이지-아웃되는 페이지를 기존의 스왑 영역에 저장하는 방법에 비해 디스크 탐색 시간을 크게 줄여서 프로그램의 실행 시간을 크게 줄일 수 있었다. 이런 성능 향상을 확인하기 위해 SAF를 FreeBSD 6.2 커널에 구현하였다. SAF가 FreeBSD에 비해 벤치마크의 실행 시간을 약 14%~42% 단축시켰음을 보여 SAF가 운영체제의 가상 메모리 시스템뿐만 아니라 파일 시스템의 성능을 향상시킬 수 있음을 입증하였다.

참고문헌

[1] M. McKusick, W. Joy, S. Leffler and R. Fabry, “A Fast File System for UNIX,” ACM Transactions on Computer Systems, vol.2, no.3, pp.181–197, 1984.

[2] M. K. McKusick and G. V. Neville-Neil, The Design and Implementation of the FreeBSD Operating System, pp.183, Addison-Wesley, 2004.

[3] M. Rosenblum and J. Ousterhout, “The Design and Implementation of a Log-structured File System,” Proceedings of the 13th ACM symposium on Operating Systems Principles, pp.1-15, 1992.

[4] N. Megiddo and D. S. Modha, “ARC: A Self-Tuning, Low Overhead Replacement Cache,” Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp.115–130, 2003.

[5] S. Bansal and D. Modha, “CAR: Clock with Adaptive Replacement,” Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp.187-200, 2004.

[6] S. Jiang, F. Chen and X. Zhang, “CLOCK-Pro: An Effective Improvement of the CLOCK Replacement”, Proceedings of USENIX 2005 Annual Technical Conference, pp.323-336, 2005.

[7] M. Dillon, “Design elements of the FreeBSD VM system,” Web site: http://www.freebsd.org/doc/en/articles/vm-design/article.html.

[8] D. Black, J. Carter, G. Feinberg, R. MacDonald, S. Mangalat, E. Sheinbrood, J. V. Sciver and P. Wang, “OSF/1 Virtual Memory Improvements,” Proceedings of USENIX Mach Symposium, pp.87-103, 1991.

[9] J. Katcher, “PostMark: A New Filesystem Benchmark,” Network Appliance, Technical Report TR-30-22, 1997.

[10] Scimark2 benchmark home page. Web site: http://math.nist.gov/scimark2.

[11] K. Smith and M. Seltzer, “File Layout and File System Performance,” Harvard University TR-35-94, 1994.

[12] NS2 Simulator home page. Web site: http://ww.isi.edu/nsnam/ns/.

[13] nbench benchmark home page. Web site: http://www.tux.org/~mayer/linux/bmark.html.

[14] A. Siberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, 8th Ed., Wiley, 2008.

[15] L. Wirzenius, J. Oja, S. Stafford and A. Weeks, The Linux System Administrator’s Guide, Web site: http://tldp.org/LDP/sag/html/index.html.

[16] H. Huang, W. Hung and K. G. Shin, “FS2: Dynamic Data Replication in Free Disk Space for Improving Disk Performance and Energy Consumption,” Proceedings of the 20th ACM Symposium on Operating System Principles, pp.263-276, 2005.

저자소개

안우현(Woo Hyun Ahn)

1996년 경북대학교 전자공학과(학사).
1998년 KAIST 전기 및 전자공학과(석사).
2003년 KAIST 전자전산학과(박사).
2003년~2005년 삼성전자 기술총괄 소프트웨어 연구소 책임연구원.
2006년~현재 광운대학교 컴퓨터소프트웨어학과 조교수.
※관심분야 : 운영체제, 임베디드시스템, 시스템 소프트웨어.

김보곤(Bo-Gon Kim)

2008년 광운대학교 컴퓨터 소프트웨어학과(학사).
2010년 광운대학교 컴퓨터과학과(석사).
2010년~ 현재 삼성전자 무선사업부 S/W Platform 개발팀 연구원.
※관심분야 : 운영체제, 시스템소프트웨어.

김병규(Byung-Gyu Kim)

1995년 부산대학교 전기공학과(학사).
1998년 KAIST 전기및전자공학과 (석사).
2004년 KAIST 전기 및 전자공학과(박사).
2004년~2008년 한국전자통신연구원(ETRI) 선임연구원
2009년~현재 선문대학교 컴퓨터공학과 조교수.
IEEE/ACM/IEICE 정회원.
※관심분야 : 영상 및 비디오 신호처리, 비디오 압축 이론 및 고속화 기법, 임베디드 멀티미디어 기술

오재원(Jaewon Oh)

1997년 서울대학교 계산통계학과(학사).
1999년 서울대학교 전산과학과(석사).
2004년 서울대학교 전기컴퓨터공학부(박사).
2004년~2007년 삼성전자 기술총괄 소프트웨어 연구소 책임연구원.
2007년~2009년 가톨릭대학교 컴퓨터정보공학부 전임강사.
2009년~현재 가톨릭대학교 컴퓨터정보공학부 조교수.
※관심분야 : 소프트웨어 품질, 소프트웨어 공학, 모바일 SW플랫폼, 시스템소프트웨어.