MOC: 다중 오브젝트 클러스터링을 통한 BSD VM의 페이지-아웃 성능 향상

(MOC: A Multiple-Object Clustering Scheme for High Performance of Page-out in BSD VM)

양 종 철 † 안 우 현 †† 오 재 원 †††
(Jongcheol Yang) (Woo Hyun Ahn) (Jaewon Oh)

요 약 BSD 가상 메모리 시스템(BSD VM)은 페이지-아웃 시 디스크 I/O 횟수를 줄이기 위해 클러스터링 기법을 사용한다. 이 기법은 페이지-아웃 대상 페이지와 가상 메모리 공간에서 인접한 변경 페이지들을 그 대상 페이지와 함께 클러스터(그룹)를 만들어 한 번의 디스크 I/O로 디스크에 저장한다. 하지만 응용 프로그램이 가상 메모리 공간에서 서로 인접하지 않은 다수의 페이지들을 변경하면 클러스터들의 크기가 작아져 클러스터링의 효과가 감소된다. 이 문제점을 해결하기 위해 본 논문에서는 Multiple-Object Clustering(MOC) 기법을 제안한다. MOC는 클러스터별로 디스크 I/O를 하는 대신 여러 클러스터들을 모아 단일 디스크 쓰기로 페이지-아웃시킨다. 따라서 이 페이지-아웃 방식은 디스크 I/O 횟수를 감소시켜 시스템 성능을 크게 향상시킨다. MOC는 성능 검증을 위해 FreeBSD 6.2 운영체제 커널에서 구현되었다. NS2, Scimark2 SOR, nbench LU 벤치마크를 통한 MOC 성능 측정 결과 기존 BSD VM보다 MOC의 실행 시간이 9~45% 단축되었다.

키워드 : 가상 메모리, 페이지-아웃, 클러스터링, 스왑 디스크 I/O, BSD VM

Abstract The virtual memory system in 4.4 BSD operating systems exploits a clustering scheme to reduce disk I/Os in paging out (or flushing) modified pages that are intended to be replaced in order to make free rooms in memory. Upon the page out of a victim page, the scheme stores a cluster (or group) of modified pages contiguous with the victim in the virtual address space to swap disk at a single disk write. However, it fails to find large clusters of contiguous pages if applications change pages not adjacent with each other in the virtual address space. To address the problem, we propose a new clustering scheme called Multiple-Object Clustering (MOC), which together stores multiple clusters in the virtual address space at a single disk write instead of paging out the clusters to swap space at separate disk I/Os. This multiple-cluster transfer allows the virtual memory system to significantly decrease disk writes, thus improving the page-out performance. Our experiments in the FreeBSD 6.2 show that MOC improves the execution times of realistic benchmarks such as NS2, Scimark2 SOR, and nbench LU over the traditional clustering scheme ranging from 9 to 45%.

Key words : virtual memory, page-out, clustering, swap disk I/O, BSD VM

· 이 논문은 2009년도 광운대학교 교내 학술 연구비 지원에 의해 연구되었음

† 학생회원 : 광운대학교 컴퓨터과학과
shining-soul@hanmail.net
†† 정 회 원 : 광운대학교 컴퓨터과학과 교수
whahn@kw.ac.kr
††† 종신회원 : 가톨릭대학교 컴퓨터정보공학부 교수
jwoh@catholic.ac.kr
(Corresponding author임)

논문접수 : 2009년 7월 13일
심사완료 : 2009년 8월 10일

정보과학회논문지 : 시스템 및 이론 제36권 제6호(2009.12)

1. 서 론

최근 메모리의 대용량화로 과거보다 훨씬 더 많은 정보를 메모리에 적재(load)시켜 사용할 수 있게 되었다. 하지만 여러 응용프로그램을 동시에 실행하거나 작업량(workload)이 큰 프로세스를 실행할 때 여전히 메모리 부족 현상이 발생할 수 있다. 메모리가 부족할 경우 디스크 스왑(swap) 영역이 메모리의 일부로 사용되어 디스크 I/O가 발생하게 된다. 하드 디스크는 I/O 속도 면에서 메모리와의 격차가 매우 크기 때문에[1] 디스크 I/O 횟수가 증가하면 시스템 성능이 저하된다.

효율적인 메모리 관리를 위해 여러 페이지 교체(page replacement) 정책[2-6]들이 연구되어 왔다. 이 연구들은 페이지 부재(page fault) 횟수의 감소를 위해 메모리에 적재된 페이지들 중 적절한 페이지를 선택하여 교체시킨다. 이러한 기존 연구들은 페이지-인으로 발생하는 디스크 I/O 감소에만 초점을 두고 있다. 그런데 페이지 교체 시 교체 대상 페이지가 더티(dirty 또는 변경) 페이지인 경우 디스크 스왑 영역으로 페이지-아웃(page-out)이 수행된다. 그러나 기존 연구들은 페이지-아웃으로 발생하는 디스크 I/O에 대해서는 고려하지 않고 있다.

동일한 크기의 데이터를 디스크에 저장할 때 디스크 I/O 횟수가 적을수록 시스템 성능이 좋아진다[7]. 이것은 가상 메모리 시스템의 스왑 디스크 사용 시에도 해당된다. 따라서 가상 메모리 시스템의 성능 개선을 위해 페이지-아웃 과정에서 발생하는 디스크 I/O 횟수를 최소화할 필요가 있다. 이를 위해 4.4BSD[8] 가상 메모리 시스템(BSD VM)은 디스크 스왑 영역으로의 페이지-아웃 동작 시 클러스터링(clustering)[9] 기법을 사용한다. 클러스터링은 페이지-아웃을 위해 페이지-아웃 대상 페이지뿐만 아니라 이 페이지와 가상 메모리(virtual memory) 공간에서 연속적인 더티 페이지들을 함께 묶는 기법이다. 그리고 클러스터링을 통해 묶여진 페이지 그룹을 클러스터(cluster)라고 한다. 클러스터링을 사용하면 페이지 교체 시 한 번에 여러 개의 페이지들을 페이지-아웃시켜 I/O를 감소시키기 때문에 시스템 성능이 향상된다.

본 논문에서는 실험을 통하여 BSD VM의 클러스터링 기법을 사용할 때 전체 페이지-아웃 중 공간적 인접성이 작은 페이지에 대한 페이지-아웃 비율이 높은 것을 발견했다. 여기서 페이지의 공간적 인접성이란 이 페이지와 연속적인 페이지 번호를 가진 더티 페이지들이 메모리에 존재하는 정도이다. 공간적 인접성이 작으면 클러스터링 조건을 만족시키는 페이지 수가 적기 때문에 한 번에 디스크 I/O할 수 있는 데이터의 양이 줄어든다. 따라서 BSD VM의 클러스터링 기법이 효과적이지 못한 경우가 적지 않음을 알 수 있다.

본 논문에서는 BSD VM의 페이지 교체 정책을 개선한 Multiple-Object Clustering(MOC)을 제안한다. 기존 BSD VM은 클러스터링을 시도한 후 클러스터링된 페이지 수에 상관없이 한 번의 페이지-아웃 요청을 수행한다. 그러나 MOC는 여러 클러스터들을 버퍼에 모아 한 번에 페이지-아웃시키는 다중 클러스터링을 수행한다. MOC의 장점은 다중 클러스터링을 통해 페이지-아웃 요청 횟수를 감소시켜 시스템 성능을 향상시키는 것이다. 특히 시스템 상에 공간적 인접성이 작은 페이지들이 많을수록 기존보다 두드러진 성능을 보인다.

본 논문에서는 성능 검증을 위해 MOC를 FreeBSD [10] 6.2 커널에 구현하였다. FreeBSD는 4.4BSD를 기반으로 한 유닉스로써 현재 널리 사용되고 있는 운영체제이다. MOC가 구현된 FreeBSD에서 여러 벤치마크(benchmark) 프로그램을 실행한 결과 기존 BSD VM에 비해 MOC가 9~45%의 실행 시간 단축과 37~80%의 페이지-아웃 요청 횟수 감소를 보였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 BSD VM의 페이지-아웃 동작 과정에 대해 설명한다. 3장에서는 본 논문의 연구 목적 및 동기를 설명한다. 4장에서는 BSD VM의 페이지 교체 정책을 개선한 MOC의 동작 원리와 특징에 대한 설명을 하며, 5장에서는 실험과 분석 결과에 대해 설명한다. 6장에서는 관련 연구에 대해 설명하고, 마지막 7장에서는 결론을 도출하여 정리한다.

2. BSD VM의 페이지-아웃

BSD VM은 가상 메모리상의 페이지들을 전역적으로 관리하기 위해 네 개의 페이지 리스트를 사용한다. 네 개의 리스트는 활성 리스트(active list), 비활성 리스트(inactive list), 자유 리스트(free list), 그리고 캐시 리스트(cache list)이다. 활성 리스트는 최근에 활발히 참조된 페이지들을 관리하고, 그중에서 오랫동안 참조되지 않은 페이지들은 비활성 리스트의 끝(tail)으로 이동되어 관리된다. 캐시 리스트는 비활성 리스트처럼 한동안 접근되지 않은 페이지들을 관리하지만 오직 변경되지 않은(clean page) 페이지들만을 가진다. 그리고 비활성 리스트 및 캐시 리스트의 페이지가 접근되면 다시 활성 리스트로 이동한다. 자유 리스트는 빈 페이지를 관리하며, 빈 페이지 부족 시 캐시 리스트에서 자유 리스트로 페이지를 이동시켜 빈 페이지를 확보한다.

빈 페이지가 부족할 때 또는 주기적으로 실행되는 페이지 데몬(page daemon)은 비활성 리스트를 탐색(scanning)하여 페이지 교체를 수행한다[11]. 그 탐색은 LRU (Least Recently Used) 방식과 유사하게 페이지 교체를 시도하기 위해 비활성 리스트의 시작(head) 페이지

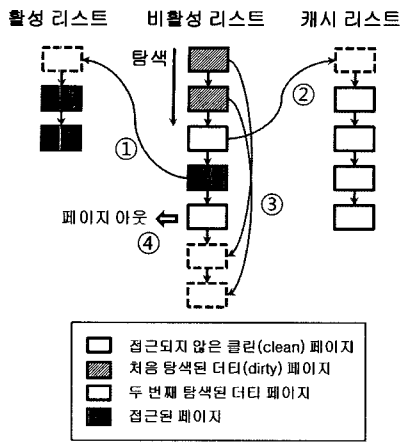

그림 1 BSD VM 비활성 리스트 탐색

부터 차례대로 수행된다. 그림 1은 비활성 리스트 탐색 시 각 페이지의 특성에 따른 다음 네 가지 작업을 나타낸다. 첫째, 해당 페이지가 비활성 리스트에 존재하는 동안 참조되었다면 활성 리스트로 옮긴다(그림 1-①). 둘째, 변경되지 않은 페이지를 캐시 리스트로 옮긴다(그림 1-②). 셋째, 더티 페이지이며 첫 번째로 탐색된 경우 리스트의 끝으로 보낸다(그림 1-③). 넷째, 두 번째로 탐색된 접근되지 않은 더티 페이지를 페이지-아웃시킨다(그림 1-④). 페이지 데몬은 비활성 리스트 탐색 과정에서 페이지-아웃으로 인한 과도한 디스크 I/O가 파일 시스템 서비스를 방해하지 않도록 하기 위해 페이지-아웃 횟수를 제한하는 임계값을 정하여 사용한다. 이 임계값만큼 페이지-아웃하거나 리스트의 끝에 도달하면 리스트의 탐색을 종료한다.

페이지 데몬은 다음 두 가지 조건을 만족하는 페이지를 페이지-아웃 대상 페이지로 삼는다. 첫째는 비활성 리스트에 존재하는 페이지이고, 둘째는 두 번째로 탐색된 더티 페이지이다. 두 번째 조건은 더티 페이지를 곧바로 페이지-아웃시키는 대신 시계 알고리즘(clock algorithm)[12]과 유사하게 한 번 더 기회를 줌으로써 곧 접근될 페이지가 조기에 페이지-아웃되는 경우를 줄인다.

BSD VM은 디스크 I/O를 줄이기 위해 페이지-아웃 시 클러스터링 기법을 사용한다. 클러스터링 대상 페이지들은 익명(anonymous page) 페이지들이며, 익명 페이지는 bss, 스택(stack) 및 힙(heap) 메모리처럼 파일과 관련이 없는 페이지를 뜻한다. 다음 두 조건을 만족하는 페이지들이 페이지-아웃 대상 페이지와 함께 클러스터링된다.

Ⅰ. 페이지-아웃 대상 페이지와 동일 익명 오브젝트(anonymous object)에 속한 페이지

Ⅱ. 페이지-아웃 대상 페이지와 연속적인 페이지 번호를 가지며 비활성 리스트에 있는 더티 페이지

여기서 오브젝트란 가상 메모리에 사상(mapping)된 파일, 익명 페이지 또는 디바이스의 정보를 관리하는 객체이며 익명 오브젝트는[13] 익명 페이지들만을 갖는다. 페이지-아웃 대상 페이지에 대한 클러스터링 시 두 가지 클러스터링 조건을 만족하는 페이지들과 함께 클러스터를 구성한다. 클러스터는 페이지-아웃 단위가 되며 클러스터의 최대 크기는 시스템 설정 값인 16으로 제한된다. 즉, 페이지-아웃 시 최대 16개의 페이지를 한 번의 I/O로 디스크에 저장할 수 있다.

그림 2는 위의 두 가지 조건에 따른 클러스터링 동작에 대한 예를 나타낸다. 각 페이지의 알파벳은 오브젝트 ID이고 숫자는 오브젝트 내에서의 페이지 번호이다. 최초로 탐색되는 A_5가 두 번째로 탐색된 더티 페이지이므로 클러스터링을 시도한다. 페이지 A_5는 오브젝트 A에 속하고 페이지 번호가 5이다. 따라서 조건 I에 의해 오브젝트 A에 있는 페이지들을 검사한다. 이 때 조건 II에 따라 5번 페이지와 연속적인 페이지들 중 더티 페이지이면서 비활성 리스트에 있는 페이지들을 찾아낸다. A_3, A_4, A_6, 이렇게 세 개의 페이지가 조건을 만족하므로 A_5와 함께 클러스터를 구성하여 페이지-아웃이 요청되고 스왑 영역의 공간을 할당 받은 후 디스크로 저장한다.

조건 I, II를 통해 클러스터링함으로써 페이지-아웃뿐 아니라 미리 읽기를 통한 페이지-인(page in) 성능도 향상된다. 미리 읽기는 페이지 부재 시 요청된 페이지뿐 아니라 다른 여러 페이지들을 디스크로부터 한 번의 디스크 I/O로 메모리에 적재시키는 기법이다. BSD VM은 페이지-인 요청 시 부재 페이지와 스왑 영역에서 연속적으로 저장된 페이지들 중 일부를 최대 15개까지 미리 읽어 들인다. 이 페이지들은 동일 오브젝트에서 부재 페이지와 연속적인 페이지 번호를 가진 페이지들이다. 미리 읽은 페이지를 접근하면 디스크 접근 없이 메모리에서 참조되므로 페이지-인 횟수가 감소되어 디스크 I/O가 줄어든다.

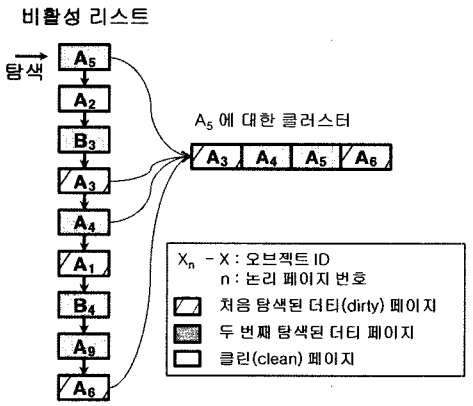

그림 2 BSD VM 클러스터링의 예

3. 연구 동기

우리는 실제 워크로드(workload)에서 BSD VM 클러스터링이 효과가 있는지 실험을 통해 분석하였다. 실험 환경으로 Pentium 4 3.0GHz CPU, 512MB DDR2 메모리, Seagate Barracuda 7200.9 160GB 7200RPM 하드 디스크를 사용하였고, 스왑 영역은 2GB로 설정하여 세 가지의 워크로드(NS2, Scimark2 SOR, nbebch LU)를 동시에 수행하였다.

그림 3은 세 가지 워크로드를 동시에 수행하는 동안 클러스터 크기(페이지-아웃당 페이지의 수)별로 페이지-아웃 횟수를 측정한 결과이다. 이 결과의 주요한 특징은 클러스터 크기가 1, 2 그리고 16인 경우가 페이지-아웃의 대부분을 차지한다는 것이다. 특히 클러스터 크기 1, 2의 경우 수치가 매우 높다. 그 이유는 다수의 페이지-아웃 대상 페이지가 공간적 인접성이 작기 때문이다. 따라서 클러스터 크기 1, 2의 디스크 I/O가 전체 I/O의 대부분을 차지한다. 작은 클러스터에 의한 디스크 I/O가 많다는 것은 기존 BSD VM의 클러스터링 정책이 개선될 여지가 있다는 사실을 나타낸다.

그림 3 동시에 세 개의 작업(NS2, Scimark2 SOR, nbench LU) 수행 시 BSD VM 클러스터링 결과

4. MOC(Multiple-Object Clustering)

기존 BSD VM은 클러스터링 조건을 만족하는 페이지 수가 소수일지라도 페이지-아웃을 수행하기 때문에 디스크 I/O가 많이 발생하여 비효율적이다. 이 장에서는 이 문제점을 해결하기 위해 여러 클러스터들을 버퍼에 모아 단일 쓰기로 저장하는 MOC(Multiple-Object Clustering) 기법을 설명한다.

4.1 기본 원리

본 절에서는 비활성 리스트에 연결된 페이지들의 배치 상태에 따라 MOC가 어떻게 동작하는지를 설명한다. MOC 동작에 대한 설명의 편의를 위해 비활성 리스트에는 더티 페이지만 존재한다고 가정한다.

그림 4부터 그림 6까지 사용하는 표기법은 다음과 같다. 첫째, 비활성 리스트 내의 각 페이지를 S와 R 중 하나로 구분한다. 임의의 페이지 p가 S(sequential)라면, p와 같은 오브젝트에 속하고 p와 연속적인 페이지가 비활성 리스트 내에 적어도 하나 존재한다. 한편 임의의 페이지 p가 R(random)이라면, 비활성 리스트 내에 p와 연속적인 페이지가 존재하지 않는다. 즉, S 페이지는 클러스터링 시 다른 페이지와 같이 묶여 크기 2 이상의 클러스터를 구성할 페이지이고, R 페이지는 클러스터 크기가 1이 되는 페이지이다. 둘째, 동일 오브젝트에 속하는 연속된 페이지의 집합을 식별하기 위해 X_i(X는 R 혹은 S, i는 자연수)를 사용한다. 페이지 집합이 S 페이지로 구성되면 X는 S이고, R 페이지 하나로 구성되면 X는 R이다. 그리고 i는 페이지 집합을 식별하기 위한 것이다. 셋째, X_i 페이지 집합에 속하는 하나의 페이지는 $X_{i,j}$(j는 자연수)로 표기하며 j는 페이지 집합 내의 페이지 순서를 나타낸다. 예를 들면 그림 4에서 페이지 ①인 $S_{1,4}$는 S_1 페이지 집합에 속하는 4번째 페이지가 된다. R 페이지들은 항상 크기가 1인 페이지 집합을 구성하므로 항상 1이 되는 j를 생략하여 R_i로 표기한다. 넷째, 임의의 페이지 $X_{i,j}$에 대한 클러스터링 결과로 생성된 클러스터의 크기를 $Cx_{i,j}$로 나타낸다. 다섯째, 버퍼와 관련한 표기법으로 T와 M이 있다. 각 페이지-아웃마다 여러 클러스터들을 모으기 위한 한 개의 임시 버퍼를 할당받아 사용하며 버퍼에 담을 수 있는 최대 페이지 개수를 M으로 표기한다. 이 값은 한 번에 페이지-아웃

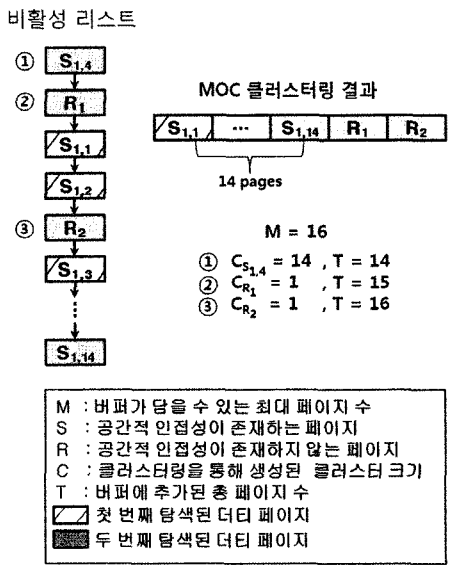

그림 4 MOC 클러스터링 수행 시 버퍼에 R 페이지만 존재하는 경우

할 수 있는 최대 페이지 수로써 16으로 고정된 시스템 설정 값을 사용한다. 그리고 버퍼 내에 채워진 총 페이지의 수를 나타내기 위해 T를 사용한다.

그림 4는 MOC로 S와 R 페이지들을 다중 클러스터링하는 과정을 나타낸다. 비활성 리스트에는 S_1에 속하는 14개의 연속적인 페이지들과 두 개의 R 페이지가 있다. 탐색이 시작되면 페이지 ①, ②, ③에 대해 각각 클러스터링 작업이 수행된다. 페이지 ①에 대한 $Cs_{1,4}$ 값은 14이고 버퍼에 여유 공간이 있으므로 페이지들을 버퍼에 추가시켜 T는 14가 된다. $Cs_{1,4}$이 14인 이유는 $S_{1,4}$에 대해 클러스터링할 때 이 페이지 번호와 연속적인 페이지를 앞뒤로 모두 찾기 때문이다. 페이지 ①에 대한 작업 후 버퍼에는 2개의 공간이 남고, 클러스터 크기가 1인 페이지 ②와 ③이 각각 클러스터링되어 버퍼의 남은 공간에 추가된다.

페이지 ①, ②, ③에 대한 세 번의 클러스터링 결과 클러스터들의 합이 버퍼 크기와 일치한다. MOC는 이와 같이 더 이상 버퍼에 여유 공간이 없으면 버퍼 내의 페이지들을 모두 묶어 한 번의 페이지-아웃 요청으로 처리한다. 반면 기존 BSD VM은 페이지 ①, ②, ③에 대한 클러스터링마다 페이지-아웃을 요청하므로 MOC보다 I/O 횟수가 2번 더 많다.

그림 5는 MOC로 R 페이지들만을 다중 클러스터링하는 과정을 나타낸다. 비활성 리스트는 최초 16개의 R 페이지와 각각 2개의 연속적인 페이지로 구성된 S_1, S_2를 갖고 있다. 탐색이 시작되면 M이 16인 버퍼에 R_1부터 R_{16}까지 16개의 R 페이지를 모으고, 그 페이지들을 한 번의 페이지-아웃으로 디스크에 저장한다. 반면 기존 BSD VM은 16번의 페이지-아웃 요청을 수행해야 한다. 결과적으로 MOC의 페이지-아웃 요청 횟수가 기존에 비해 1/16로 감소된다.

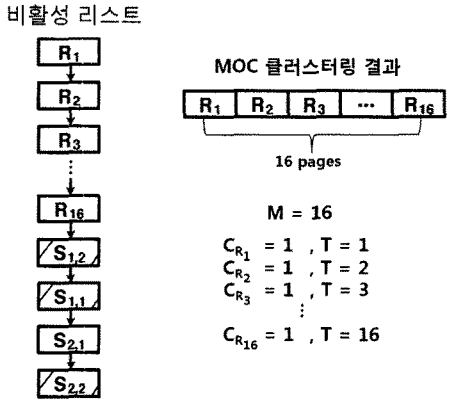

그림 5 MOC 클러스터링 수행 시 버퍼가 모두 채워지지 않는 경우

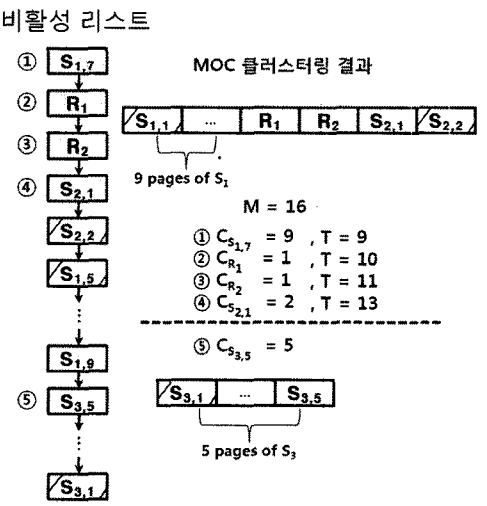

그림 6 MOC 클러스터링 수행 시 버퍼가 모두 채워지지 않는 경우

그림 6은 다중 클러스터링 과정에서 버퍼를 모두 채우지 못하고 페이지-아웃되는 경우를 나타낸다. 비활성 리스트는 각각 아홉 개, 두 개, 다섯 개의 연속적인 페이지로 구성된 S_1, S_2, S_3과 두 개의 R 페이지를 가진다. 먼저 페이지 ①부터 페이지 ④까지에 대해 각각 클러스터링이 수행된다. 각 C 값이 차례대로 9, 1, 1, 2이고 각 클러스터가 버퍼에 추가되어 T는 13이 된다. 그 후 페이지 ⑤에 대해 클러스터링을 시도하면 $Cs_{3,5}$는 5이므로 버퍼에 여유 공간이 부족하여 추가될 수 없다. 여유 공간이 부족하므로 이미 버퍼에 모아진 페이지들을 페이지-아웃하고 버퍼를 비운다. 그리고 비워진 버퍼에 $S_{3,5}$를 클러스터링하여 모은 페이지들을 채운 후 탐색을 계속한다. 비록 버퍼가 모두 채워지지 않은 상태에서 페이지-아웃이 요청됐지만 기존 BSD VM보다 효율적이다. BSD VM이 ④까지 검색하는 동안 4번의 페이지-아웃 요청을 발생시키는데 비해 MOC는 한 번의 페이지-아웃 요청만을 필요로 한다.

페이지-아웃 요청 시 버퍼 내에서 하나의 클러스터가 될 수 있는 페이지들이 여러 클러스터로 나뉘어 있을 수 있다. 그 이유는 비활성 리스트 탐색 도중에도 새로운 페이지가 리스트에 추가되어 사용될 수 있기 때문이다. 예를 들어 오브젝트 X의 페이지 5번~7번으로 구성된 클러스터 A가 버퍼에 추가된 이후 페이지 4번이 비활성 리스트에 추가될 수 있다. 그리고 페이지-아웃이 발생하기 전에 페이지 4번을 포함한 새로운 클러스터 B가 버퍼에 추가되면, 클러스터 A와 클러스터 B는 하나의 클러스터가 될 수 있음에도 불구하고 각각의 클러스터로써 버퍼에 존재한다.

MOC는 여러 클러스터로 나뉘어 쉬여 있는 연속적인 페이지들을 순서대로 배치하여 디스크에 연속적으로 저장할 수 있도록 하기 위해 오브젝트 정렬을 수행한다. 오브젝트 정렬의 목적은 미리 읽기 성능을 향상시키는 것이다. 미리 읽기 대상 페이지는 부재 페이지와 디스크 상에서 연속적으로 저장된 페이지들 중 클러스터링 조건을 만족하는 페이지들이다. 따라서 오브젝트 정렬을 수행하면 미리 읽기 조건을 만족시키는 페이지들이 많아져 미리 읽기 성능이 향상된다.

그림 7은 오브젝트 정렬의 예를 나타낸다. 오브젝트 A의 경우 페이지 번호 48부터 52까지의 연속적인 페이지들이 두 클러스터로 나뉘어 있다. 오브젝트 정렬을 통해 오브젝트 A의 두 클러스터가 연속하여 배치된다. 이때, 페이지 번호가 오름차순이 되도록 정렬된다. 오브젝트 정렬 후 페이지-아웃을 요청하면 오브젝트 A의 페이지들은 디스크 상에 연속적으로 저장된다. 차후에 49번 페이지에 대한 페이지 부재가 발생하면 48번부터 52번까지의 5개 페이지가 미리 읽기를 통해 메모리로 적재된다. 만약 오브젝트 정렬 없이 저장됐다면 48번, 49번, 두 개의 페이지만 메모리로 적재된다.

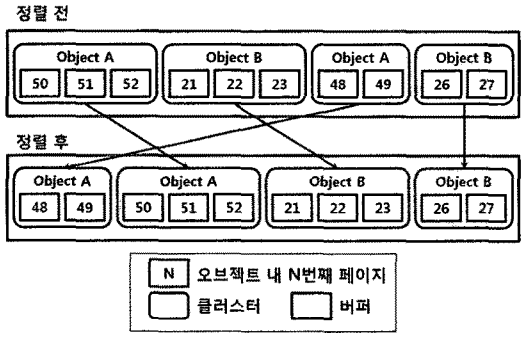

그림 7 오브젝트 정렬

MOC는 다음과 같은 장점들을 가진다. 첫째, 여러 클러스터들을 모아 단일 디스크 쓰기로 저장하기 때문에 디스크 I/O를 감소시켜 시스템 성능을 향상시킨다. 특히 공간적 인접성이 작은 페이지들이 많을수록 크게 성능이 향상된다. 둘째, MOC는 프로세스가 사용하는 메모리 페이지의 인접성이 작은 경우와 큰 경우에 관계없이 많은 페이지들을 모아 페이지-아웃할 수 있다. 따라서 수행되는 프로세스들의 메모리 사용 특성을 따로 분석하지 않아도 효율적인 페이지-아웃이 가능하다.

4.2 스왑 디스크 공간 할당

페이지-아웃이 요청되면 페이지들을 저장할 연속적인 빈 공간을 할당한다. MOC는 기존 BSD VM의 라딕스 트리(radix-tee)[14]를 사용한 공간 할당 방법을 그대로 사용한다. 라딕스 트리는 32비트 비트맵(bitmap)을 가진 노드들을 사용하여 스왑 영역의 상태를 관리한다. 스왑 영역의 연속적인 빈 공간은 라딕스 트리를 이용해 최초 적합(first-fit)으로 검색된다.

페이지-아웃 요청 시 요청된 크기의 연속적인 공간을 할당하는데 실패하면 크기를 줄여가며 할당을 시도한다. 할당에 성공하면 그 크기만큼의 페이지들만 디스크로 저장되고, 남아 있는 페이지들에 대해 공간 할당 시도를 반복한다. 결과적으로 최초에 페이지-아웃이 요청된 페이지들이 분할되어 저장될 수 있다. 하지만 스왑 영역의 사용률은 일반적으로 약 28%[15]이기 때문에 공간 할당이 실패하는 경우는 많지 않다.

BSD VM의 페이지-아웃시의 스왑 디스크 공간 할당 방법은 페이지-인과 밀접한 연관이 있다. 페이지-인의 미리 읽기는 페이지-아웃된 클러스터의 페이지들이 디스크 상에서 최대한 연속적으로 저장됨을 전제로 동작하기 때문이다. 따라서 MOC는 BSD VM의 스왑 디스크 공간 할당 방법을 그대로 사용하기 때문에 기존 페이지-인 효율을 저해하지 않는다.

MOC는 스왑 디스크 공간 할당 방법에 대한 의존도가 크지 않다. MOC가 BSD VM의 기존 스왑 공간 관리 기법을 사용하는 이유는 기존 페이지-인 정책을 그대로 사용함으로써 BSD VM과 페이지-아웃 I/O의 성능을 정확히 비교하기 위해서이다. MOC의 목표는 특정 스왑 공간 관리 기법을 기반으로 디스크 헤드의 움직임을 최소화하는 것이 아니라 디스크 I/O의 횟수를 감소시키는 것이다.

4.3 MOC 알고리즘

그림 8은 페이지 데몬이 비활성 리스트를 탐색할 때 더티 페이지에 대해 수행하는 MOC 동작 알고리즘을 나타낸다. 우선 페이지 데몬은 비활성 리스트를 탐색할 동안 허용되는 페이지-아웃의 수를 계산하여 임계값을 설정한다. 이 임계값을 설정하는 이유는 페이지 데몬의 과도한 페이지-아웃 요청으로 인해 발생하는 디스크 I/O가 파일 시스템에 대한 서비스를 방해할 수 있기 때문이다. 페이지-아웃의 임계값이 설정되면 비활성 리스트의 첫 페이지부터 순서대로 탐색한다.

탐색 대상 페이지가 더티 페이지인 경우의 처리는 페이지가 이미 탐색된 기록이 있는지의 유무에 따라 다르다. 만약 탐색 대상 페이지가 첫 번째로 탐색된 더티 페이지라면(단계 1) 페이지가 한 번 더 기회를 갖도록 하기 위해 리스트의 끝으로 옮기고 표시해둔다(단계 2). 그러나 탐색 대상 페이지가 두 번째로 탐색된 더티 페이지이고 페이지-아웃의 임계값이 0보다 크면(단계 3) 이 페이지가 속한 오브젝트를 기록한 후(단계 4) 오브젝트 타입에 따라 다르게 처리한다. MOC는 익명 페이지

```
// let m be the maximum number of pages that
   write buffer buf can contain
// let p be a current dirty page
// let pageout_limit be the maximum number of
   page-out allowed

1:  if p is not marked then {
2:    mark p and move p to end of inactive list
    }
3:  else if pageout_limit > 0 then {
4:    set target_object to an object including p
5:    if target_object is a vnode object then {
6:      execute page-out routine of BSD VM .
7:      decrement pageout_limit by one
      }
8:    else {
9:      set cluster to a cluster of dirty pages in
         target_object such that they are contiguous
         with p in virtual address space
10:     set c to size of cluster
11:     set t to total size of clusters in buf
12:     if c <= m - t then {
13:       put cluster into buf
        }
14:     else if c > m - t then {
15:       sort pages of each object in buf
           in ascending order
16:       store pages of buf to disk
17:       reset buf
18:       decrement pageout_limit by one
19:       put cluster into buf
        }
      }
    }
```

그림 8 MOC 알고리즘

만을 대상으로 한다. 따라서 디스크의 파일과 연관된 vnode 타입의 오브젝트에 속한 페이지에 대해서는(단계 5) 기존 BSD VM의 클러스터링을 통해 페이지-아웃을 수행하고(단계 6) 페이지-아웃 임계값을 감소시킨다(단계 7). 오브젝트 타입이 vnode가 아니라면(단계 8) 다중 클러스터링으로 처리한다.

다중 클러스터링 처리 과정은 페이지-아웃이 발생하는 경우와 발생하지 않는 경우 두 가지로 나뉜다. 우선 탐색 대상 페이지에 대해 클러스터링을 수행하여 클러스터를 만든다(단계 9). 그리고 이 클러스터의 크기(단계 10)와 버퍼의 여유 공간을 확인하여(단계 11) 클러스터가 버퍼에 추가될 수 있는지를 판단한다. 만약 클러스터 크기가 버퍼의 여유 공간보다 작거나 같으면(단계 12) 버퍼에 클러스터를 추가한다(단계 13). 그러나 버퍼의 여유 공간보다 클러스터 크기가 크면(단계 14) 소속된 오브젝트별로 버퍼내간을 확인하을 페이지 번호를 기준으로 오름차순으로 정렬한다(단계 15). 정렬이 끝나면 버퍼 내간을 확인하을 묶어 페이지-아웃을 요청하고(단계 16) 버퍼를 비운 후(단계 17) 페이지-아웃 임계값을 감소시킨다(단계 18). 그 후 버퍼의 여유 공간 부팸에 추추가되지 못했던 클러스터를 버퍼에 추가한다(단계 19). 만약 페이지-데몬이 비활성 리스트 탐색을 끝냈을 때 버퍼가 비어있지 않다면, 버퍼내의 페이지들을 묶어 페이지-아웃을 요청한 후 버퍼를 비운다.

5. 실 험

5.1 실험 환경

MOC의 성능 검증을 위해 Pentium 4 3.0GHz CPU, Seagate Barracuda 7200.9 160GB 7200RPM 하드 디스크로 구성된 시스템에서 실험했다. 메모리는 DDR2 512MB와 1GB 두 가지를 사용했다. 운영체제는 4.4BSD 가상 메모리 시스템(BSD VM)을 탑재한 FreeBSD 6.2를 사용하였고 스왑 영역은 2GB로 설정하였다. 그리고 BSD VM과의 성능 비교를 위해 MOC를 FreeBSD의 가상 메모리 시스템에 구현하였다. 실험에 사용한 응용 프로그램 및 설정들은 다음과 같다.

NS2[16] : 네트워크 시뮬레이터로써 유, 무선 환경의 TCP, 라우팅, Multicast 프로토콜 등에 대한 지원을 제공한다. 메모리 사용량을 증가시키기 위해 시뮬레이션 샘플 파일(wireless-pkt-demo.tcl)에서 노드 수를 700으로 수정하였고 프로그램 수행 시 메모리 사용량은 약 676MB이었다.

Scimark2[17] : 다섯 가지의 과학 계산을 통해 시스템 성능을 측정하는 벤치마크로써 Java와 C의 두 가지 버전이 있다. 실험에는 C 버전의 Scimark2에서 다섯 가지 테스트 목록 중 SOR을 사용했다. SOR의 입력 매트릭스(matrix) 크기는 8000×8000으로 설정했으며 프로그램 수행 시 메모리 사용량은 약 502MB이었다.

nbench 2.2.2[18] : CPU, FPU 및 메모리 시스템의 성능을 테스트하기 위한 벤치마크 프로그램이며 열 가지 테스트 목록을 수행한다. 실험에는 테스트 목록 중 LU를 사용하였으며 LU 입력 매트릭스 크기는 10000×10000으로 설정했다. 프로그램 수행 시 메모리 사용량은 약 764MB이었다.

5.2 실험 결과

본 논문에서는 512MB와 1GB 메모리에서 NS2, Scimark2 SOR, nbench LU를 개별적으로 실행하는 실험과 세 개의 벤치마크 모두를 동시에 실행하는 실험을 수행하였다. 또한 공간적 인접성이 성능에 미치는 영향을 분석하기 위해 마이크로벤치마크(micro-benchmark)를 개발하여 성능을 측정했다.

5.2.1 마이크로벤치마크

본 논문에서 사용한 마이크로벤치마크는 공간적 인접성에 따른 페이지-아웃의 성능을 분석하기 위해 클러스터 크기를 입력받아 페이지-아웃할 때마다 이 개수의 페이지들을 디스크로 저장하도록 구현하였다. 초기에

1.5 GB의 메모리를 할당한 후 메모리의 첫 페이지부터 클러스터 크기만큼의 페이지들을 변경한다. 그 다음에 클러스터 크기의 페이지 수만큼 건너뛴 후 다시 클러스터 크기만큼의 페이지들을 변경한다. 건너뛴 이유는 클러스터 크기별로 동일한 개수의 페이지를 변경하기 위해서이다. 이러한 과정들을 메모리의 마지막 페이지까지 반복적으로 수행하고 프로그램은 종료된다. 따라서 벤치마크가 수행되는 동안 페이지-아웃되는 클러스터들은 모두 동일한 크기가 된다.

그림 9는 512MB 메모리에서 벤치마크가 1, 2, 4, 6, 8, 10, 12, 14, 16의 클러스터 크기로 페이지 아웃할 때의 실행 시간을 나타낸다. 클러스터 크기가 8이하일 때 MOC가 BSD VM보다 성능이 개선되었다. 그 이유는 MOC가 한 개의 버퍼에 여러 개의 클러스터를 모아서 페이지 아웃하기 때문이다. 하지만, MOC에서 클러스터 크기가 8인 경우에 비해 6인 경우 성능이 감소하였다. 이것은 클러스터 크기가 6인 경우 한 번 페이지-아웃할 때 저장하는 페이지 수(12개)가 클러스터 크기가 8일 경우(16개)보다 작아서 전체적인 페이지-아웃 횟수가 증가했기 때문이다.

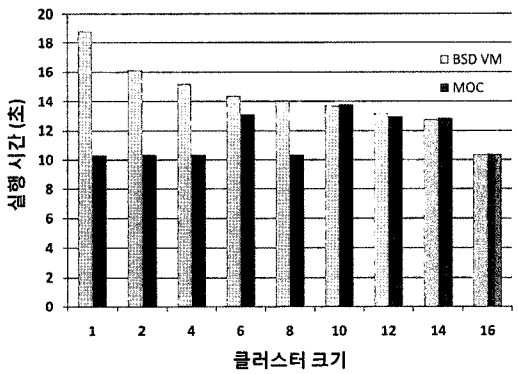

그림 9 마이크로벤치마크 실행 결과

클러스터 크기가 8보다 큰 경우에 MOC와 BSD VM의 성능이 비슷해지는 이유는 두 클러스터의 크기의 합이 MOC 버퍼의 여유 공간을 초과하여 다중 클러스터링이 가능하지 않기 때문이다. 결과적으로, 어떤 프로그램의 공간적 인접성이 평균 8이하면 MOC의 성능이 크게 개선되고, 8이상이면 BSD VM과 성능이 비슷함을 알 수 있다. 이 실험을 통해 BSD VM에 비해 성능이 뚜렷하게 개선되는 크기 8이하의 클러스터는 공간적 인접성이 작다고 할 수 있다.

5.2.2 NS2, Scimark2 SOR, nbench LU 개별 실행 결과

표 1의 (a)-①과 그림 10은 512MB 메모리에서 NS2를 실행할 때의 실행 시간과 페이지-인/아웃을 수행한 I/O 횟수를 나타낸다. MOC의 실행 시간이 BSD VM보다 약 16% 빨라졌고 페이지-아웃 횟수가 약 83% 감소했다. MOC는 페이지-아웃마다 요청되는 페이지들의 수가 BSD VM보다 많았다. 즉, 같은 수의 페이지를 페이지-아웃하기 위해 MOC가 BSD VM보다 적은 I/O 횟수를 요구했다. 이에 따라 MOC가 BSD VM에 비해 적은 페이지-아웃 횟수와 빠른 실행 시간을 보였다.

페이지-인 결과는 MOC의 페이지-인 횟수가 BSD VM과 거의 동일한 것을 볼 수 있다. 그 이유는 MOC가 페이지들을 단순하게 많이 모으는 것이 아니라 각 페이지의 공간적 인접성을 고려하기 때문이다. MOC는 BSD VM의 기존 페이지-인 기능을 그대로 사용하며, 기존 페이지-인은 부재 페이지와 공간적 인접성을 가진 페이지들을 디스크로부터 미리 읽는다. 결과적으로 MOC의 공간적 인접성을 고려한 다중 클러스터링은 BSD VM의 페이지-인 성능을 훼손하지 않으면서 페이지-아웃을 줄여 성능을 향상시킨다.

또한 그림 10에서 BSD VM보다 MOC의 페이지-아웃 횟수가 월등히 낮은 결과는 NS2가 사용하는 대다수 페이지들의 공간적 인접성이 작다는 사실을 의미한다. 페이지들의 공간적 인접성이 작으면 각 클러스터링 후의 클러스터 크기가 작으므로 MOC가 BSD VM보다 훨씬 많은 페이지들을 클러스터링하여 페이지-아웃할 수 있다. 따라서 메모리상에 공간적 인접성이 작은 페이

표 1 물리 메모리 크기, 클러스터링 방법에 따른 벤치마크 실행 시간(초)

벤치마크 \ 클러스터링 방법		(a) 512MB		(b) 1GB	
		BSD VM	MOC	BSD VM	MOC
①	NS2	729	608	397	397
②	SOR	651	356	5	5
③	LU	1994	1753	8	8
④	동시 실행	8398	6406	751	684

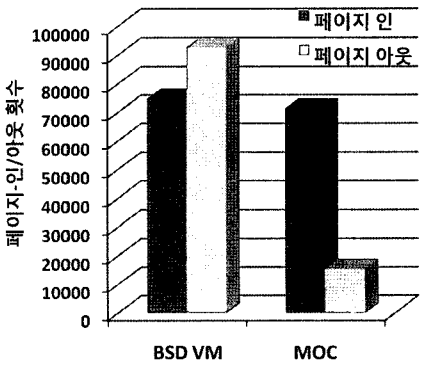

그림 10 NS2 실행 시 페이지-인/아웃 횟수(512MB)

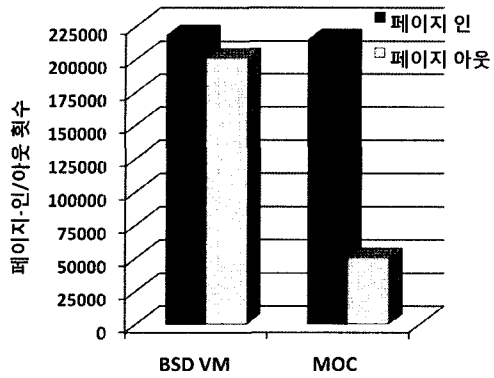

그림 11 Scimark2 SOR 실행 시 페이지-인/아웃 횟수 (512MB)

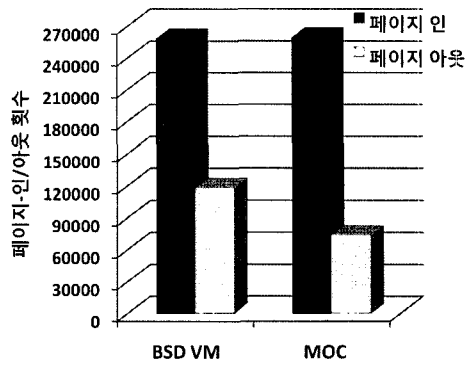

그림 12 nbench LU 실행 시 페이지-인/아웃 횟수(512MB)

지들이 많을수록 MOC와 BSD VM 사이의 페이지-아웃 횟수 격차가 커지게 된다.

표 1의 (a)-②와 그림 11은 512MB 메모리에서 Scimark2 SOR을 실행할 때의 실행 시간과 페이지-인/아웃을 수행한 I/O 횟수를 나타낸다. 이 실험의 경우 MOC가 기존 BSD VM에 비해 실행 시간이 약 45% 빨라졌고 페이지-아웃이 약 75% 감소했다. SOR의 실험 결과도 NS2와 마찬가지로 MOC와 BSD VM의 페이지-아웃 횟수 격차가 크다. 따라서 SOR이 실행되는 동안 사용하는 페이지들 대부분이 작은 공간적 인접성을 가진다는 사실을 의미한다.

SOR과 NS2 두 실험 모두 MOC의 페이지-아웃 횟수가 BSD VM보다 적지만 페이지-아웃 횟수 감소 비율은 다르다. SOR이 75%, NS2가 83%로써 SOR이 NS2보다 페이지-아웃 횟수 감소 비율이 더 낮다. 하지만 실행 시간 단축 비율은 SOR이 45%로 16%인 NS2보다 더 높다. 그 이유는 BSD VM으로 수행했을 때 실행 시간 대비 페이지-아웃 횟수 비율이 NS2와 SOR에서 서로 다르기 때문이다. 이 수치가 높은 프로세스일수록 MOC로 수행할 때의 실행 시간이 기존 BSD VM으로 수행할 때보다 높은 비율로 감소된다.

표 1의 (a)-③과 그림 12는 512MB 메모리에서 nbench LU를 실행할 때의 실행 시간과 페이지-인/아웃을 수행한 I/O 횟수를 나타낸다. 실험 결과 MOC가 BSD VM에 비해 실행 시간이 약 12% 빨라졌고 페이지-아웃 횟수가 약 37% 감소했다. 앞의 두 실험에 비해 MOC에 의한 페이지-아웃 감소 비율이 훨씬 낮다. 그 이유는 LU의 경우 비활성 리스트가 가진 페이지들의 공간적 인접성이 앞의 두 실험에 비해 비교적 크기 때문이다. 공간적 인접성이 크면 클러스터링 후의 클러스터 크기가 크다. 그래서 MOC가 BSD VM보다 많은 페이지를 클러스터링하지만 BSD VM에서 클러스터링된 페이지 수와 큰 차이는 나지 않게 된다. 또한 실행 시간 단축 비율도 세 실험 중 가장 낮은 결과를 보였다.

실험 결과들에 따르면 MOC의 성능이 어떤 특성의 프로그램을 수행하는가에 따라 다소 유동적이라는 것을 알 수 있다. NS2와 SOR처럼 공간적 인접성이 작은 페이지들이 많은 프로그램일수록 MOC의 성능이 BSD VM보다 높다. 또한 LU와 같이 공간적 인접성이 큰 페이지가 많은 프로그램에서도 MOC는 빠른 실행 시간과 낮은 페이지-아웃 횟수를 보인다. 즉, MOC는 공간적 인접성 패턴이 다른 여러 프로그램 상에서 개선된 성능을 보인다.

512MB에서와는 달리 1GB 메모리에서 세 벤치마크를 각각 실행시킨 결과 페이지-아웃이 전혀 발생하지 않았다. 이는 각 프로그램이 수행하는 동안 필요로 하는 메모리 양이 사용 가능한 물리 메모리 양보다 작기 때문이다. 이 실험의 경우 표 1의 (b)-①, (b)-②, (b)-③을 통해 알 수 있듯이 BSD VM과 MOC 두 경우 모두 같은 결과를 보였다.

5.2.3 NS2, Scimark2 SOR, nbench LU 동시 실행 결과

표 1의 (a)-④와 그림 13은 512MB 메모리에서 NS2, SOR, LU를 동시에 실행할 때의 실행 시간과 페이지-인/아웃을 수행한 I/O 횟수를 나타낸다. 실험 결과 MOC가 BSD VM에 비해 약 27% 빨라졌고 페이지-아웃 횟수가 약 69% 감소했다. 이 결과는 세 프로그램을 각각 실행했을 때 보인 12~45%의 실행 시간 감소와 37~83%의 페이지-아웃 횟수 감소의 중간 성능을 나타낸다. MOC의 성능 개선 이유는 그림 14의 페이지-아웃당 페이지의 수별 페이지-아웃 횟수 결과를 통해 볼 수 있다. BSD VM에서 페이지-아웃당 페이지의 수가 1, 2인 페이지-아웃이 전체 페이지-아웃 중 높은 비율을 차지하는 반면 MOC는 다중 클러스터링을 하기 때문에 페이지-아웃당 페이지의 수가 16인 페이지-아웃이 거의 대부분이다. 따라서 MOC가 BSD VM에서의 소수의 페이지들을 페이지-아웃하는 비효율적인 디스크 I/O를 줄여 성능을 향상시킴을 알 수 있다.

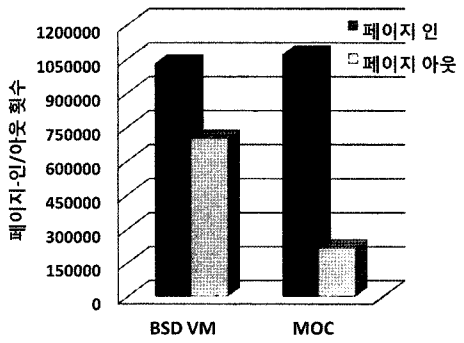

그림 13 NS2, Scimark2 SOR, nbench LU 동시 실행 시 페이지-인/아웃 횟수(512MB)

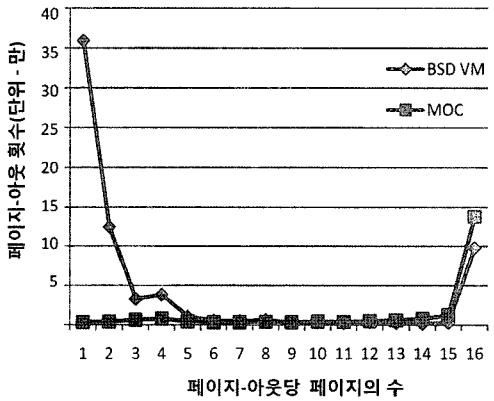

그림 14 NS2, Scimark2 SOR, nbench LU 동시 실행 시 클러스터링 결과(512MB)

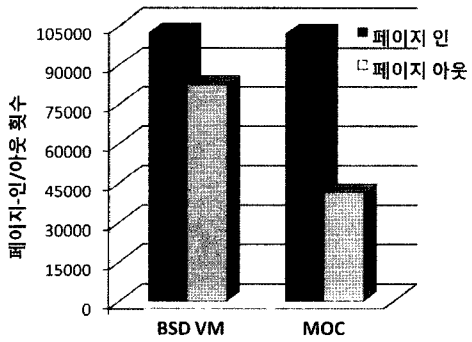

그림 15 NS2, Scimark2 SOR, nbench LU 동시 실행 시 페이지-인/아웃 횟수(1GB)

앞의 실험들에 비하면 MOC의 실행 시간 단축 비율이 지금까지의 실험 중 가장 낮게 나타났다. 또한 페이지-아웃 횟수를 보면 512MB 메모리에서 세 프로그램을 동시에 실행한 경우의 페이지-아웃 횟수보다 훨씬 작다. 그 이유는 물리 메모리의 크기가 512MB에서 1GB로 증가하여 페이지-아웃이 많이 발생하지 않았기 때문이다. 따라서 물리 메모리 크기에 비해 응용 프로그램의 메모리 사용량이 적을수록 MOC의 성능 개선 효과가 작다는 것을 알 수 있다.

표 2와 그림 16은 1GB 메모리에서 각 프로그램의 메모리 사용량을 높여 실행할 때의 실행 시간과 페이지-인/아웃을 수행한 I/O 횟수를 나타낸다. NS2는 노드의 수를 1000으로, Scimark2 SOR은 입력 매트릭스 크기를 10000×10000으로 변경하였고 nbench LU는 11000×11000으로 변경하였다. 실험 결과 MOC가 BSD VM에 비해 실행 시간이 약 20% 감소했고 페이지-아웃 횟수가 약 61% 감소했다. 메모리 사용량이 증가함에 따라 페이지-아웃되는 페이지가 많아져 MOC가 다중 클러스터링할 기회가 증가하여 성능 개선이 이루어졌다. 따라서 같은 크기의 물리 메모리를 사용할 때 메모리 사용률이 높은 프로그램일수록 MOC의 성능 개선 효과가 증가함을 알 수 있다.

MOC는 BSD VM보다 많은 페이지들을 한 번에 페이지-아웃하기 때문에 가까운 미래에 다시 변경될 가능성이 있는 페이지를 페이지-아웃할 수도 있다. 이러한 페이지가 많으면 페이지-아웃 횟수가 증가하고 이에 따라 디스크 I/O가 증가하여 성능이 저하될 수도 있다. 하지만 페이지-아웃된 전체 페이지 수를 나타낸 표 3을 통해 이 문제가 발생하지 않고 있음을 알 수 있다. 이 페이지-아웃된 전체 페이지 수는 동일한 페이지가 N번

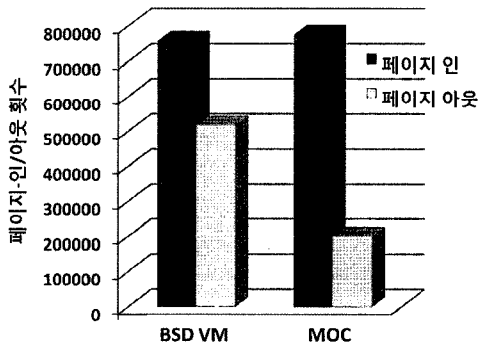

그림 16 NS2, Scimark2 SOR, nbench LU의 각 메모리 사용량 변경 후 동시 실행 시 페이지-인/아웃 횟수(1GB)

표 2 NS2, Scimark2 SOR, nbench LU의 각 메모리 사용량 변경 후 동시 실행 시간(1GB)

클러스터링 방법	BSD VM	MOC
실행 시간(초)	6635	5312

표 3 물리 메모리 크기, 클러스터링 방법에 따라 벤치마크 실행 시 페이지-아웃된 페이지 수

클러스터링 방법 / 프로그램	512MB		1GB	
	BSD VM	MOC	BSD VM	MOC
NS2	243,265	242,107	0	0
SOR	756,459	766,719	0	0
LU	917,923	919,423	0	0
동시 실행	2,821,329	2,891,037	582,411	577,944

페이지-아웃되면 N개의 페이지 수로 누적시킨 결과이다. BSD VM과 MOC의 페이지-아웃된 페이지 수의 차이가 평균 0.5%로 거의 동일하다. 그 이유는 FreeBSD의 활성 리스트와 비활성 리스트를 사용한 2단계(2-level) 페이지 관리 기법[19]이 짧은 시간 안에 활발하게 접근되는 페이지(hot page)와 그렇지 않은 페이지(cold page)들을 잘 구분하기 때문이다.

6. 관련 연구

페이지 교체 시, 교체될 대상 페이지 선정 정책은 시스템 성능에 영향을 미치는 주요 원인 중 하나이다. 따라서 오랫동안 효율적인 페이지 교체를 위한 여러 알고리즘들이 연구되어왔다. 기존 페이지 교체 방법들은 크게 3가지로 나뉜다. 첫째는 최근성(recency)을 고려하는 방법이고[2,5,6,20,21], 둘째는 참조 빈도(frequency)를 고려하는 방법이며[22], 마지막은 두 가지 모두를 고려한 방법이다[3,4,23]. 그러나 이 교체 방법들은 교체 대상 페이지의 선택 방법만을 제시하였을 뿐 대상 페이지가 페이지-아웃될 경우 발생할 디스크 I/O의 횟수는 고려하지 않았다. 또한 기존 연구의 대부분은 실제 운영체제의 가상 메모리 시스템에 구현하지 않고 시뮬레이션을 통해 실험하였다. 따라서 각 페이지 교체 기법이 실제 시스템에서 페이지-아웃에 미치는 영향에 대한 분석이 이루어지지 않았다. 반면에 MOC는 페이지 교체 동안 페이지 아웃될 페이지를 저장하기 위해 요구되는 I/O 횟수의 최적화를 고려하였다. 리눅스 커널 2.6은 대한 분석과 유사하게 전역적인 두 개의 LRU 리스트를 사용하여 페이지들을 관리한다. 하나는 최근에 접근된 페이지들을 관리하는 활성 리스트이고, 다른 하나는 한동안 접근되지 않은 페이지들을 관리하는 비활성 리스트이다. 페이지 교체 동작에서 리눅스와 대한 분석의 주요 차이점은 한 번의 페이지-아웃 요청 시 전달되는 페이지의 수이다. 대한 분석은 클러스터링 조건을 만족하는 다수의 페이지들을 한 번의 디스크 I/O로 페이지-아웃시킨다. 그러나 리눅스는 각각의 교체 대상 페이지에 대해 페이지-아웃을 요청하므로 I/O 효율이 대한 분석에 비해 떨어지는 단점을 가지고 있다. 또한 대한 분석은 LRU 리스트를 통한 시간적 요소뿐 아니라 클러스터링을 통해 가상 메모리상의 공간적 인접성을 고려한다. 그러나 리눅스는 페이지-아웃 대상 페이지 선정을 위해 단지 비활성 리스트에서 가장 오랫동안 접근되지 않은 더티 페이지들을 모은다. 즉, 시간적 요소를 제외하면 페이지-아웃되는 페이지들은 서로 연관성이 없다. MOC는 공간적 인접성을 고려한 다중 클러스터링을 통해 다수의 페이지를 페이지-아웃하기 때문에 리눅스보다 많은 페이지를 단일 디스크 쓰기로 저장한다.

스왑 압축(Swap Compression) 기법[24]은 페이지 크기가 4KB인 가상 메모리 시스템에서 페이지들을 압축하여 4KB 크기의 압축 캐시 버퍼에 모은 후 한 번의 디스크 I/O를 통해 다수의 페이지를 디스크에 저장한다. 스왑 압축 기법의 문제점은 기존보다 성능이 저하되는 경우가 생길 수 있다는 것이다. 만약 압축 효과가 거의 없는 페이지들이 많으면 압축 캐시 버퍼는 대부분 하나의 페이지만 포함하게 된다. 따라서 압축률이 낮은 페이지가 대다수인 프로세스의 경우, 압축과 해제로 인한 부하 때문에 오히려 성능이 저하된다. 스왑 압축 기법은 한 번의 I/O로 다수의 페이지를 내보낼 수 있다는 점에서 클러스터링과 유사하다. 그러나 MOC는 성능 개선 효과가 수행되는 프로그램의 공간적 인접성 패턴에 따라 다소 유동적이긴 하지만 스왑 압축 기법과는 달리 오버헤드가 적어 기존 시스템의 성능을 저하시키지 않는다.

7. 결 론

본 논문은 페이지-아웃 과정에서 BSD VM의 클러스터링 정책이 가지는 문제점을 제시하고 이를 개선한 MOC를 제안하였다. 실험을 통해 MOC가 BSD VM에 비해 9~45%의 실행 시간 단축과 37~83%의 페이지-아웃 요청 횟수 감소 결과를 보여 기존 BSD VM 클러스터링 정책보다 우수함을 입증하였다. 특히 사용되는 페이지들의 공간적 인접성이 작은 프로세스의 경우 MOC 기법이 탁월한 성능 개선 효과가 있음을 보였다. 아울러 MOC는 기존 BSD VM의 페이지-인 성능을 저해하지 않으면서도 페이지-아웃 요청을 줄여 시스템 성능을 개선시킨다. 또한 MOC는 비활성 리스트 내 페이지들의 공간적 인접성이 유동적이더라도 적절히 클러스터링함으로써 성능을 높일 수 있다. 그리고 MOC는 구현이 간단하다. 따라서 MOC는 BSD VM의 오브젝트-페이지 관계처럼 페이지가 계층적 구조에 속하는 시스템에 큰 어려움 없이 적용될 수 있을 것으로 예상한다.

참 고 문 헌

[1] Bryant and O'Hallaron, *Computer Systems: A Pro-*

grammer's Perspective, pp.447-458, Prentice Hall, 2003.

[2] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *ACM SIGMETRICS Conference*, pp.31-42, 2002.

[3] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," *2nd USENIX Conference on File and Storage Technologies (FAST 03)*, pp.115-130, 2003.

[4] S. Bansal and D.Modha, "CAR: Clock with Adaptive Replacement," *3rd USENIX Conference on File and Storage Technologies (FAST 04)*, pp.187-200, 2004.

[5] Y. Smaragdakis, S. Kaplan, and P. Wilson, "EELRU: simple and effective adaptive page replacement," *ACM SIGMETRICS Conference*, pp.122-133, 1999.

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

[7] Alma Riska, James Larkby-Lahet and Eric Riedel, "Evaluating Block-level Optimization Through the IO Path," *USENIX Annual Technical Conference*, pp.247-260, 2007.

[8] M. K. McKusick, K. Bostic, M. Karels, and J. Quarterman, *The Design and Implementation of the 4.4BSD Operating System*, Addison Wesley, 1996.

[9] Black D, Carter J, Feinberg G, MacDonald R, Sciver JV, Wang P, Mangalat S, and Sheinbrood E, "OSF/1 virtual memory improvements," *USENIX Mach Symposium*, pp.87-104, 1991.

[10] FreeBSD. FreeBSD home page. Web site: http://www.freebsd.org.

[11] M. K. McKusick and G. V. Neville-Neil, *The Design and Implementation of the FreeBSD Operating System*, p.190, Addison-Wesley, 2004.

[12] F. J. Corbato, "A Paging Experiment with the Multics System," MIT Project MAC Report MAC-M-384, 1968.

[13] M. K. McKusick and G. V. Neville-Neil, *The Design and Implementation of the FreeBSD Operating System*, p.616, Addison-Wesley, 2004.

[14] M. K. McKusick and G. V. Neville-Neil, *The Design and Implementation of the FreeBSD Operating System*, p.179, Addison-Wesley, 2004.

[15] P. Domingues, P. Marques, L. Silva, "Resource usage of Windows computer laboratories," *ICPP 2005 Workshops*, pp.469-476, 2005.

[16] NS2 home page. Web site: http://www.isi.edu/nsnam/ns.

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

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

[19] O. Babaoglu and D. Ferrari, "Two-level replacement decision in paging stores," *IEEE Transactions on Computers*, vol.C-32, no.12, pp.1151-1159, 1983.

[20] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," *ACM SIGMOD Conference*, pp.297-306, 1993.

[21] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," *VLDB Conference*, pp.297-306, 1994.

[22] L.B. Sokolinsky, "LFU-K: An Effective Buffer Management Replacement Algorithm," *DASFAA 2004*, pp.670-681, 2004.

[23] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Transactions on Computers*, vol.50, no.12, pp.1352-1361, 2001.

[24] R. Cervera, T. Cortes, and Y. Becerra, "Improving Application Performance through Swap Compression," *USENIX Annual Technical Conference*, pp.207-218, 1999.

양 종 철

2008년 광운대학교 컴퓨터 소프트웨어학과(학사). 2008년~현재 광운대학교 일반대학원 컴퓨터과학과 석사과정. 관심분야는 운영체제, 임베디드

안 우 현

1996년 경북대학교 전자공학과(학사). 1998년 KAIST 전기 및 전자공학과(석사). 2003년 KAIST 전자전산학과(박사). 2003년~2005년 삼성전자 기술총괄 소프트웨어연구소 책임 연구원. 2006년~현재 광운대학교 컴퓨터 소프트웨어학과 조교수. 관심분야는 운영체제, 임베디드시스템, 시스템소프트웨어

오 재 원

1997년 서울대학교 계산통계학과(학사) 1999년 서울대학교 전산과학과(석사). 2004년 서울대학교 전기컴퓨터공학부(박사) 2004년~2007년 삼성전자 기술총괄 소프트웨어연구소 책임 연구원. 2007년~2009년 가톨릭대학교 컴퓨터정보공학부 전임강사. 2009년~현재 가톨릭대학교 컴퓨터정보공학부 조교수 관심분야는 소프트웨어 품질, 소프트웨어 공학, 모바일 SW 플랫폼, 시스템 소프트웨어