컨테이너 기반의 가상머신 시스템에서 메모리 자원 아이솔레이션을 위한 동적 메모리 사용량 측정 기법*

김효진○, 노삼혁
홍익대학교 정보컴퓨터공학부
{kimhj○, noh}@cs.hongik.ac.kr

Dynamic Memory Measurement Scheme to Support Memory Resource Isolation for Container-based Virtual Machines

Hyojin Kim○, Sam H. Noh
School of Information & Computer Engineering, Hongik University

요 약

시스템 가상머신 환경은 높은 하드웨어 효율성과 높은 보안을 요구하는 시스템에서 그 사용이 점차 늘어나고 있다. 최근 많이 알려진 하이퍼바이저 가상머신 시스템은 높은 아이솔레이션과 보안성을 보장하나 각 게스트 운영체제 별로 운영체제 이미지를 가지기 때문에 하드웨어 효율성이 떨어지는 반면, 컨테이너 기반 가상머신 시스템은 운영체제 이미지의 공유로 인하여 높은 자원 효율성과 확장성을 가진다. 그러나 메모리 자원의 아이솔레이션에 대하여 취약점을 갖는 문제점을 안고 있다. 본 논문에서는 컨테이너 기반 가상머신 시스템에서 동적으로 각 가상머신별로 메모리 사용량 증가에 따른 페이지 히트율-곡선(Hit-Ratio-Curve)을 측정하였다. 이 곡선을 관찰해 보면 각 가상머신의 메모리 필요량을 알 수 있으며 이를 기반으로 메모리 자원을 할당하게 될 경우 효과적으로 메모리 자원의 아이솔레이션을 제공할 수 있게 된다. 본 논문에서는 대표적인 컨테이너 기반 가상머신인 리눅스 VServer가 적용되어 있는 리눅스 2.6.17 커널에 직접 구현하였으며, Lmbench 및 리눅스 커널 컴파일 등을 통하여 오버헤드를 측정하였고 1.6~7.2%의 적은 오버헤드로 이를 측정할 수 있음을 확인하였다.

1. 서 론

최근 시스템 가상머신은 높은 하드웨어 효율성과 높은 보안을 요구하는 시스템에서 그 사용이 점차 늘어나고 있다. 시스템 가상머신은 가상화된 완전한 운영체제 환경으로서, 시스템의 자원을 가상화하고 이들에 대한 인터페이스를 제공하여 하나의 머신 안에서 여러 개 혹은 여러 가지의 운영체제를 사용자에게 제공한다. 최근 개발되고 있는 시스템 가상머신으로는 VMware[1], Xen[2] 등의 하이퍼바이저 가상머신과 VServer[3] 등의 컨테이너 기반 가상머신이 있다.

가상머신이 활용되는 환경 중에서 가상 사설 서버(VPS)나 플래닛랩과 같이 다수의 서로 다른 유저 그룹들에게 서로 독립된 어플리케이션 환경을 제공하는 시스템에서는 높은 하드웨어 효율성과 아이솔레이션을 동시에 제공해 주어야 한다[4]. 그러나 이 두 가지를 동시에 성취하는 것은 쉽지 않은 일이다. 아이솔레이션은 시스템 안의 개체들이 서로 영향을 주지 않도록 하는 것을 뜻하는데 곧, 어떤 개체에서의 활동이 다른 개체의 권리를 침해해서는 안 된다는 것이다. 아이솔레이션은 시스템 자원 효율성과 트레이드-오프의 관계에 있다.

하이퍼바이저 가상머신 시스템은 각 게스트 운영체제 별로 독립된 운영체제 이미지를 가진다. 이를 통하여 높은 아이솔레이션을 이루지만, 전체적인 하드웨어 효율성은 떨어지게 된다. 반면, 컨테이너 기반 가상머신은 호스트 운영체제와 게스트 운영체제가 모두 하나의 동일한 운영체제 이미지를 공유한다. 따라서 하나의 가상머신의 생성에 소모되는 자원이 적기 때문에 높은 하드웨어 활용도를 보이고 범위성이 뛰어나지만 리소스 아이솔레이션에 대해서는 취약점을 보인다. 특히 메모리 자원의 경우 커널 내부에서 공유하는 자료 구조가 많기 때문에 오버헤드가 적은 적절한 정책 없이는 아이솔레이션을 달성하기가 어려우며 이것은 컨테이너 기반 가상머신의 문제점으로 지적되어 왔다[2].

본 논문에서는 컨테이너 기반 가상머신의 메모리 자원 아이솔레이션 문제를 해결하기 위하여, 첫 번째 단계로서 리눅스 VServer에 히트율-곡선(Hit-Ratio-Curve)을 구현하고 이를 통해 수행시간 동안 동적으로 각 가상머신의 실제 메모리 필요량을 측정하였으며, 이에 따른 오버헤드를 Lmbench 및 리눅스 커널 컴파일 등을 수행하여 측정하였다.

리눅스 VServer에서는 자원 사용 통계를 위해서 워터마크 페이지 수를 계수한다. 그러나 이것은 총 사용 페이지 개수 량을 나타낼 뿐 실제 페이지 필요량을 반영하지는 않는다. 게다가 가상 머신에서 사용하고 있는 페이지를 반환할 때 어떤 페이지를 선택해야 하는지에 대한 근거를 전혀 제시하지 못하기 때문에 가상머신마다 페이지 관리를 위한 리스트를 별도로 관리해야 하고, 이 오버헤드는 납득될만한 수준으로 충분히 적어야 한다.

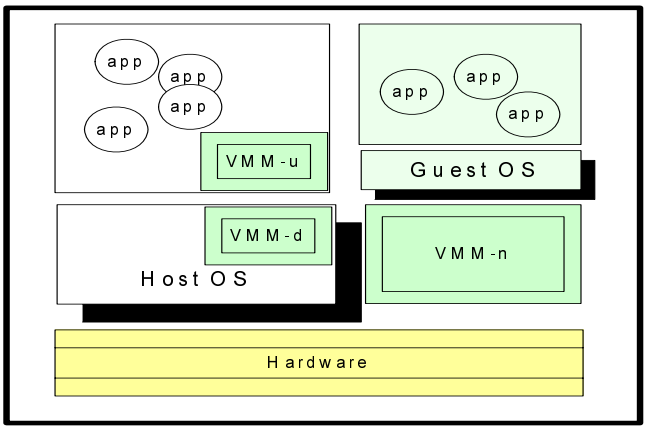
* 이 논문은 2005년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임 (KRF-2005-213-D00075)

이 이후의 논문은 다음과 같이 구성되어 있다. 제 2장에서는 배경 지식 및 관련 연구로서 하이퍼바이저 가상머신과 컨테이너 기반 가상머신에 대해서 살펴보고, 제 3장에서는 컨테이너 기반 가상머신의 메모리 자원 아이솔레이션을 위한 실제 메모리 필요량 측정 기법의 설계 및 구현에 대해 설명한 후 제 4장에서 실험 결과를 제시한다. 마지막으로 제 5장에서 논문의 결론을 내리고 향후 연구에 대하여 언급하며 논문을 마무리 한다.

2. 배경 지식 및 관련 연구

2.1 하이퍼바이저 가상머신과 컨테이너 기반 가상머신

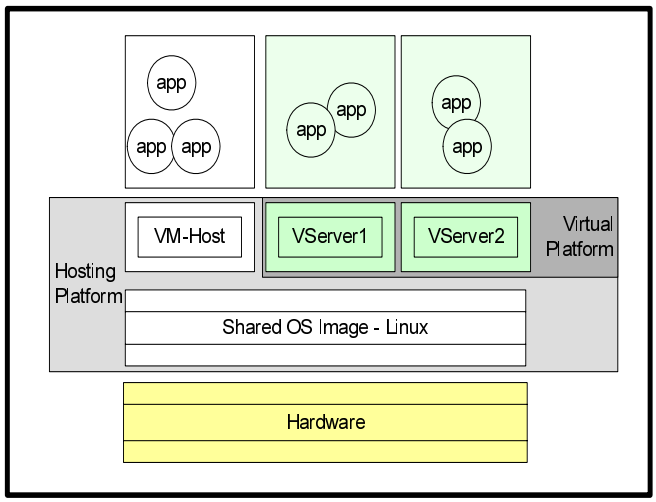
최근 가상머신으로 가장 많이 알려져 있는 것은 하이퍼바이저 가상머신으로서 vmware[1], Xen[2] 등이 이에 속한다. 일반적으로 하이퍼바이저 가상머신은 하나의 머신 위에 서로 다른 운영체제 환경을 제공하기 위하여 설계되었다. 하이퍼바이저 가상머신의 보편적인 구조는 <그림 1>과 같다. 운영체제의 가상화를 위한 소프트웨어를 가상머신 모니터(VMM)라고 하며 <그림 1>과 같은 구조에서는 가상 머신 모니터가 가상화된 운영체제보다 높은 권한을 갖기 때문에 하이퍼바이저라고 한다. 이 구조에서는 소프트웨어 설치만으로 한 머신 안에 여러 가지 운영체제를 사용할 수 있기 때문에 최근에 특히 개인용 컴퓨터 사용자들과 프로그램 개발자들에게 각광을 받고 있다. 하이퍼바이저 가상머신은 가상화된 운영체제들이 서로 연관성을 갖지 않는다는 가정을 가지고 설계되기 때문에 각각의 운영체제 이미지를 가지면서 공유를 하지 않음으로써 태생적으로 완전한 아이솔레이션을 제공한다.



<그림 1> 하이퍼바이저 가상머신의 구조

플래닛랩[5, 6]이나 가상 사설 서버와 같이 높은 정도의 아이솔레이션과 효율성을 동시에 요구하면서 게스트 운영체제들과 호스트 운영체제가 서로 다른 운영체제일 필요가 없는 경우에는 리소스 컨테이너의 개념을 따라 설계된 컨테이너 기반 가상머신을 사용하는 것이 효과적이다. 리소스 컨테이너[7]는 하나의 애플리케이션이 여러 개의 프로세스로 구성되어 있을 때, 각 프로세스 별로 자원을 할당하기 보다는 프로세스들의 논리적인 집합체를 자원의 할당 개체로 삼는 것이 바람직하기 때문에, 이러한 집합체의 추상화로서 제안되었다. 이 리소스 컨테이너의 개념을 가상머신에 적용한 컨테이너 기반 가상머신은 가상머신이 리소스 컨테이너가 되며 운영체제 환경을 제공한다. 즉, 하나의 운영체제 이미지를 가상화된 운영체제들이 공유하면서 시스템 자원에 대한 공유를 하게 된다. 그러나 자원의 아이솔레이션에 있어서는 적절한 정책이 필요하고 특히 메모리 자원의 경우 커널의 내부 자료구조가 복잡하게 공유되기 때문에 적은 오버헤드로 효과적인 아이솔레이션을 제공하는 것은 어려운 문제이다. 컨테이너 기반 가상머신으로는 Solaris 10 [8], Virtuozzo [9], Linux VServer [3] 등이 있는데 Solaris 10과 Virtuozzo는 제품화된 가상머신이고 VServer는 오픈소스 프로젝트이다.

본 논문은 컨테이너 기반 가상머신 중 VServer를 위주로 연구하였다. VServer구조는 <그림 2>와 같으며, 이러한 구조는 개별적인 운영체제 환경을 제공하면서도 시스템 자원을 최대한 공유하기 때문에 높은 효율성을 갖는다. VServer의 높은 확장성은 시스템 제공자에게 큰 매력으로 작용할 수 있다.



<그림 2> 컨테이너 기반 가상머신 VServer의 구조

2.2 가상머신의 메모리 자원 아이솔레이션 및 관리 기법

지금까지 살펴본 것과 같이 하이퍼바이저 가상머신에서는 구조적으로 아이솔레이션을 제공한다. 이는 메모리 자원 아이솔레이션에 대해서도 동일한데 하이퍼바이저 가상머신은 하나의 가상머신이 생성될 때 고정된 값으로 메모리양을 예약 할당하며, 이 값은 한 머신 안에서 생성할 수 있는 가상머신의 개수에 영향을 미치게 된다. 각 가상머신은 그 안에 별도의 페이지 테이블을 가지면서 스스로 메모리 자원을 관리한다. 이를 위해서 물리 주소, 가상 주소 외에 머신 주소의 세 개의 메모리 주소 계층 구조를 가진다. 이 경우 주소 계층을 추가적으로 운용하는데 필요한 공간적 오버헤드 및 번역 오버헤드가 들어가기 때문에 효율적인 면이 떨어진다.

컨테이너 기반 가상머신에서는 고정된 값으로 처음부터 메모리를 할당하는 것이 아니라 실행하는 동안에 각 가상머신들이 경쟁을 통하여 메모리 자원을 획득한다. 이는 최대한 가벼운 가상화 계층으로 자원 효율을 최대화 하고자 하기 때문이다. 그러나 그렇기 때문에 현재 리눅스 VServer에서의 메모리 자원의 아이솔레이션은 잘 이뤄지지 않고 있다. VServer에서는 기존 리눅스에서처럼 프로세스마다 메모리를 할당한다. 즉, 할당에

있어서 가상머신 차원에서의 관점으로 메모리를 관리하지 않고 있기 때문에 메모리를 많이 사용하는 프로세스가 포함되어 있는 가상머신에 계속해서 더 많은 메모리를 할당할 수 있다. 그러다가 더 이상 가용한 메모리가 없는 상태가 되면 시스템을 감시하고 있던 와치독 데몬이 현재 메모리를 가장 많이 사용하고 있는 가상머신을 소멸시키고 자원을 강제로 반납시킨다. 그러나 이러한 방식은 특정 가상머신에게 지속적으로 충분한 메모리를 할당하지 못할 가능성이 있을 뿐 만 아니라 와치독 데몬에 의해 소멸된 가상머신에게도 공평하게 관리가 되었다고 할 수 없다.

일반적인 서버 시스템의 자원 아이솔레이션은 Sullivan과 Seltzer에 의해 연구되었는데 이는 로터리 스케쥴링 방식을 확장한 개념이다[4]. 이를 통해 CPU나 디스크 대역폭에 대하여는 효과를 거둘 수 있었으나 메모리 자원을 로터리 스케쥴링으로 할당하기에는 커널 내부의 자료구조가 복잡하여 효율적이지 못했다. 차선책으로 명시적으로 자원을 요구하는 커널 프로세스에 대해서만 대응하도록 하였는데 이는 컨테이너 기반 가상머신이 유저 모드에 속하게 되기 때문에 적용하기에 미흡하다.

3. CvHRC 측정 기법

3.1 가상머신의 메모리 필요량 측정

본 논문에서는 컨테이너 기반 가상머신에서 메모리 자원 아이솔레이션을 위하여 각 가상머신마다 동적으로 메모리 사용량을 측정하고 이 측정값을 추후 메모리 자원 관리에 사용할 수 있도록 메모리 필요량을 측정하는 기법을 제안한다. 메모리의 사용량을 관찰하여 메모리를 관리하는 기법으로서 일차적으로 고려할 수 있는 것이 워킹 셋의 크기를 측정하는 것이다. 워킹 셋은 일정한 기간인 워킹 셋 윈도우 동안 참조된 페이지의 개수로 정의되는데, 이것은 페이지가 참조된 특성을 반영하지는 못하기 때문에 정확한 메모리 사용량의 기준이 될 수 없다. 현재 리눅스 VServer에서는 VServer가 생성되고 소멸되는 시점 전체에 걸쳐서 자원 사용 통계를 위하여 워터마크 페이지 수를 계수한다. 그러나 이것은 위에서 언급한 바와 같이 총 사용 페이지 개수량을 나타낼 뿐 페이지가 참조되고 있는 특성을 반영하지 못하기 때문에 실제 페이지 필요량을 파악하는데 사용할 수 있는 값은 아니다. 게다가 페이지 반환이 필요한 시점에서 가상머신에서 사용하던 페이지를 반환할 때 어떤 페이지를 선택해야 하는지에 대한 근거를 전혀 제시하지 못하기 때문에 페이지의 참조 특성을 반영할 수 있는 페이지 필요량 측정 기법이 필요하다.

한편, 리눅스 커널에서는 메모리 관리 정책에서 반환할 페이지를 선택하기 위하여 액티브 리스트와 인액티브 리스트로 구성되어 있는 단순화된 LRU 리스트에서 페이지들을 관리한다. 액티브 리스트에는 비교적 최근에 참조된 물리 페이지들이 등록되어 있으며 페이지 반환이 필요할 때에 후보가 될 수 있는 페이지들은 인액티브 리스트로 옮겨진다. 인액티브 리스트에서 최종적으로 반환할 페이지가 결정되고 페이지 반환 절차를 거치게 된다. 그런데 리눅스 커널의 이 액티브 리스트와 인액티브 리스트에서는 가상머신 혹은 프로세스 단위로 페이지를 관리하는 것이 아니라 페이지의 소유자 정보를 모르고 관리하기 때문에 가상머신을 단위로 하여 메모리 자원 관리를 할 수 없다. 따라서 가상머신마다 반환할 페이지를 선택하기 위한 별도의 페이지 관리 리스트가 필요하며 이 리스트는 커널의 페이지 관리 리스트에 부가적이기 때문에 이것을 관리하기 위한 오버헤드는 납득될 만한 수준으로 충분히 적어야 한다.

기존 연구로서 동적으로 자원 사용량을 측정하기 위하여 미스율-곡선을 사용한 경우들이 있는데, Kim 등[10]은 파일 시스템에서 버퍼 캐쉬의 블록 관리를 위하여 이 기법을 사용하였으며, Zhou 등[11]은 프로세스의 메모리 사용량을 측정하기 위하여 하드웨어적 방법과 운영체제 수정을 통한 구현으로 이 기법을 사용하였다. 본 논문에서는 Zhou 등[11]의 운영체제 수정을 통한 소프트웨어적 구현을 컨테이너 기반 가상머신 환경에 적합하게 재설계하고 구현하였다. 컨테이너 기반 가상머신을 위한 메모리 사용량 측정 기법인 CvHRC(**H**it-**R**atio-**C**urve for **C**ontainer-based **v**irtual machine)와 Zhou 등의 소프트웨어적 구현의 차이에 대해서는 다음 절에서 자세하게 설명한다.

3.2 CvHRC의 설계

본 연구에서는 히트율-곡선을 사용하고 있으며 컨테이너 기반 가상머신을 위한 히트율-곡선이라 하여 CvHRC라고 명명하였다. 히트율-곡선은 페이지 개수의 증가에 따른 페이지 히트율의 곡선으로 정의한다. 즉, 더 많은 페이지를 사용하면 더 좋은 성능을 기대하게 되는데 추가 할당에 따른 실제 성능 향상을 나타내는 것이다. Zhou 등[11]이 측정한 미스율-곡선은 히트율-곡선과 같은 원리를 가지고 있고 히트율과 미스율은 *히트율+미스율=100*의 관계를 가진다.

이 곡선을 관찰하다 보면 어느 페이지 수 이상을 지나게 되면 곡선 접선의 기울기가 완만해지는 지점을 발견할 수 있다. 접선의 기울기가 완만하다는 것은 더 많은 페이지를 가지고 있더라도 히트율의 이득, 즉 성능의 향상이 크게 달라지지 않았다고 할 수 있다. 우리는 이 지점에서의 페이지 양을 가상머신의 페이지 필요량이라고 간주한다. 그러면 각 가상머신들에서 이 페이지 필요량까지만 메모리를 할당하게 되면 각 가상 머신들은 필요량을 보장받으면서 다른 가상머신에게는 최소한의 영향을 미치게 되어 메모리 자원의 아이솔레이션을 달성할 수 있게 될 것이다.

Zhou 등[11]의 미스율-곡선은 한 프로세스에 대해서 그 프로세스가 소유하는 페이지의 개수 증가에 따른 페이지 미스율을 가상주소글 기반으로 측정하였다. 가상주소를 가지고 측정하는 이유는 물리주소를 가지고 측정하게 되면 스왑에 대한 처리를 할 수 없고, 리눅스 커널의 메모리 관리자가 해당 페이지를 해제시켜버린 경우 내부 자료구조가 훼손될 수 있기 때문이다. 미스율/히트율-곡선은 페이지 수에 대한 효용을 나타내는 리스트를 관리할 때 페이지 수가 필요량 이상 넘어가게 되면 페이지 반환의 희생양이 되기 때문에 리스트는 이 성질을 나타낼 수 있어야 한다. 이를 위하여 가장 일반적으로 사용되고 있는 LRU 기법을 리스트에 구현하였다. 그리고 이 리스트를 재조정하기 위한 방법은 자주 사용되는 페이지 부분은 일정 기간마다 스캐닝을 통해 최근 참조된 페이지를 MRU로 옮기고 히트 카운트를 증가시키며, 자주 참조되지 않는 페이지들을 위해서는 스캐닝 오버헤드를 줄이기 위하여 페이지 폴트 핸들러 루틴에서 위조 페이지 폴트를 발생시켜 처리한다.

본 논문의 CvHRC에서도 이에 해당하는 내용은 위의 설계를 동일하게 차용하였다. 그러나 Zhou 등[11]의 설계는 프로세스를 기반으로 이루어졌기 때문에 컨테이너 기반 가상머신을 위해서는 추가적인 설계가 필요하다. CvHRC 기법 설계의 특징을 정리하면 다음과 같다.

가. 플래닛랩에서 관찰되는 특성들을 고려하였다.

나. 하나의 가상머신에 속하는 프로세스들 간의 페이지 공유에 대해 고려하였다.

다. 기존 운영체제의 운용 환경과 VServer의 운용 환경에 대한 차이를 고려하였다.

CvHRC 설계에서는 실제 환경을 반영하는 시나리오로서 플래닛랩에서 관찰되는 특성들을 고려하였다. 플래닛랩은 전 세계 25개국에 걸쳐 378개의 사이트에 775개의 노드를 가지고 있으며 지속적으로 확장하고 있는 거대한 실험용 네트워크이다. 플래닛랩에서는 분산 가상화를 위하여 VServer를 도입하였다. 플래닛랩에서의 자원 관리는 크게 신뢰 도메인인 사이트간의 자원 관리와 하나의 노드 내에서의 자원 관리 두 부분으로 나눌 수 있다. 사이트간의 자원 관리는 SHARP라는 메커니즘을 통해 이뤄진다[12]. 이에 반해 노드 내에서의 자원 관리는 한 노드 안에서 생성된 여러 개의 가상머신에게 자원을 분배하는 것이다. 본 연구는 이 자원 중 특별히 메모리 자원에 대해 관심을 가지고 있다.

플래닛랩에서는 현재 한 노드 당 15개까지의 가상머신이 생성될 수 있으며 각 가상머신에서 구동되는 프로세스들의 종류는 특별한 제약이 없이 다양하기는 하지만 대체로 생존 기간이 길다. 또한 한 가상머신 안에는 여러 개의 프로세스가 존재할 수 있지만, 하나의 프로세스가 여러 VServer에 걸쳐서 소유되지는 않는다.

페이지의 공유에 대해서는 한 가상머신 내에 있는 프로세스들 간의 페이지 공유를 고려하여 설계하였다. 페이지 공유는 기존 운영체제에서와 같이 프로세스 생성 시 COW 메커니즘이나 shmget을 통한 명시적인 메모리 공유 방식으로 발생할 수 있다. 이렇게 페이지 공유가 발생하게 되면 같은 물리 주소를 가지는 페이지들의 가상주소들을 리눅스 커널 2.6에서부터 지원하고 있는 리버스-매핑 메커니즘을 이용, 하나의 리스트로 연결하여 CvHRC의 LRU 리스트에서 하나의 개체로 관리한다. 한편 한 물리 페이지가 여러 가상머신에게 공유되는 것은 드문 경우로서 공유 라이브러리를 접근할 때 이외에는 발생하지 않기 때문에 이런 경우가 발생하면 가상머신 영역 외에서 기존 리눅스 메모리 관리자가 일반 프로세스의 페이지처럼 관리하도록 하였다.

또한, 기존 운영체제의 운용 환경과 VServer의 운용 환경에 대한 차이도 고려하였다. 기존 운영체제에서는 관리자가 완전한 루트 권한을 가지고 있다. 즉, 현재 운영 환경 설정에 대한 최고 권위 및 책임을 가지고 가장 적절한 환경을 설정하고자 노력한다. 그러나 VServer의 운영 환경에서는 사이트의 관리자가 VServer 내에서 루트 권한을 가지고 있지만 그것이 시스템 전체에 대한 루트 권한을 의미하는 것은 아니다. 전 시스템적인 관점에서 그 루트는 그 역시 가상화된 루트로서 일반 사용자에 불과하다. 따라서 Zhou 등[11]의 설계에서는 실시간으로 현재 프로세스들의 상태를 고려하여 /proc 파일시스템 인터페이스로써 미스율-곡선 기법의 구동 여부를 결정하였으나, VServer 환경에서는 커널 컴파일 시점에 결정하고 VServer의 생성/소멸과 동시에 자동으로 해당 자료구조들이 초기화되는 것이 적당하다.

4. 실험 및 결과

4.1 구현 및 실험 환경

실험에 사용된 시스템의 사양은<표 1>과 같다. 리눅스 커널의 버전은 개발 당시 가장 최근의 커널 버전이었으며 플래닛랩에서 사용하는 VServer에서 가장 최신 버전이다.

<표 1> 구현 및 실험 환경

CPU	1.86 GHz 듀얼 코어
CPU 캐쉬	2048KB
메모리	2GB
하드 디스크	삼성 S-ATA 160G 7200RPM
운영체제	Fedora Core 5 Linux 2.6.17 + VServer

4.2 실험 설계

본 연구에서 측정한 실험 결과는 공간 오버헤드와 실제로 프로그램을 수행시켰을 때의 수행 시간 오버헤드 및 CvHRC 곡선이다. CvHRC는 기존에 존재하지 않던 자료구조 및 이를 관리하는 함수들이 생겼기 때문에 오버헤드가 없을 수는 없다. 따라서 여기에서의 관건은 오버헤드가 가상머신 시스템에서 포용할 수 있을 정도로 적은가 하는 것이다.

이를 측정하기 위하여 Lmbench 3.0-a7과 커널 컴파일을 가상머신에서 가상머신의 수를 늘려가며 수행시켜보았다. Lmbench 벤치마크는 운영체제의 병목 현상을 일으킬 수 있는 부분들을 집중 공략하여 측정하는 마이크로 벤치마크의 집합이다[13]. 여기에서는 CvHRC의 오버헤드 및 성능을 실험하기 위해 파라미터를 두 가지로 설정하였다. 이후에 Lmbench-Mem이라고 호칭하는 부분은 메모리 관련 파라미터들을 'Yes'라고 설정하여 수행시킨 것이고, 설정 파라미터를 모두 'Yes'로 한 것은 Lmbench-All으로 지칭하겠다.

4.3 실험 결과

가. 공간 오버헤드

<표 2>는 하나의 가상머신 당 CvHRC를 측정하기 위해 필요한 공간 오버헤드를 정리한 것이다. 공간 오버헤드는 정적 공간 오버헤드와 동적 공간 오버헤드로 분류할 수

<표 2> 공간 오버헤드

	크기
정적 공간 오버헤드	169KB
동적 공간 오버헤드	40B x (스캔 리스트 크기 + ∑ 페이지공유 수)

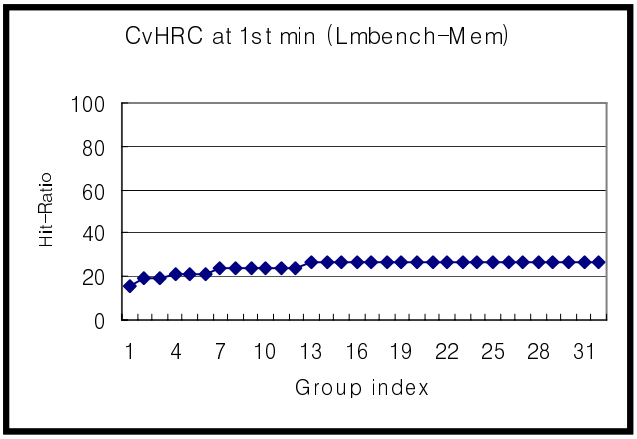
<표 3> VServer 개수의 변화에 따른 시간 오버헤드

가상머신 수	실행 프로그램	Linux+VServer 실행 시간	Linux+VServer +CvHRC 실행 시간	VServer 당 오버헤드
1	Lmbench-Mem	5' 28''	5' 45''	5.2%
	커널 컴파일	6' 27''	6' 33''	1.6%
2	Lmbench-Mem	10' 19''	10' 53''	5.5%
	커널 컴파일	8' 32''	8' 51''	5.7%
3	Lmbench-Mem	20' 43''	22' 12''	7.2%
	커널 컴파일	12' 16''	13' 08''	7.1%

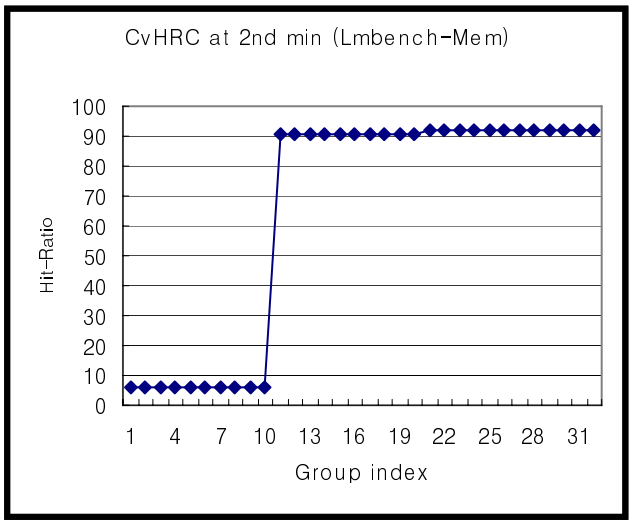
있다. 정적 공간오버헤드는 가상머신이 생성될 때 함께 생성되는 자료구조들로서 페이지들의 히트 계수기, 해쉬 테이블 등으로 구성되어 있다. 동적 공간 오버헤드는 가상머신 안에서 프로세스들이 수행되는 동안에 발생하는 자료구조들로서 LRU리스트의 노드들이다. 하나의 노드는 40B가 필요하며 스캔리스트와 페이지 공유 수로 결정된다. 만일 스캔 리스트의 크기가 8192라고 가정하면 한 가상머신마다 약 500KB가 필요하며 한 머신에 15개의 가상머신이 활성화되어 있다고 가정하면 약 7.5MB의 공간이 소요된다.

나. 수행시간 오버헤드

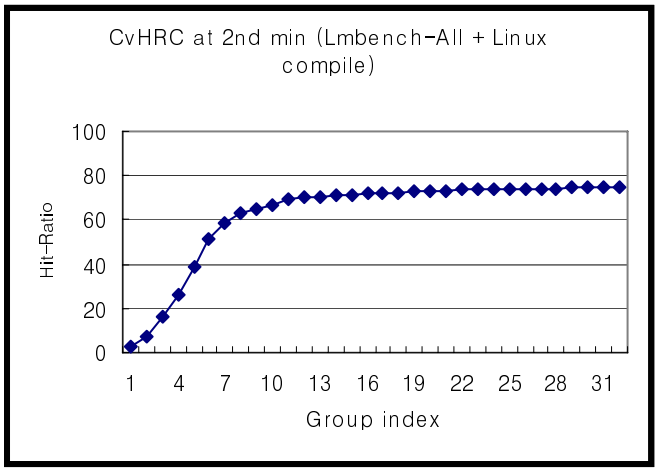
이 실험에서는 예비 실험을 통하여 스캔 리스트의 크기가 8192개일 때 입력 프로세스가 적절하게 관찰됨을 파악하고 이를 고려하여 스캔 리스트의 페이지 수를 8192개로 설정한 환경에서 실험을 수행하도록 하였다. <표 3>은 가상머신의 수를 변화시켜가면서 Lmbench-Mem 및 커널 컴파일을 가상머신 안에서 5회씩 수행시킨 시간의 평균이며, 정확한 측정을 위하여 매 세트의 실험이 끝날 때마다 호스트 플랫폼 전체를 재부팅 하였다. 이 결과 값을 CvHRC가 적용되어 있지 않은 플래닛랩의 Linux 2.6.17 + VServer 커널에서 동일하게 프로세스를 실행시킨 결과와 비교하였으며, 오버헤드는 가상머신 한 개당 발생하는 오버헤드를 계산한 것이다.



<그림 3> Lmbench-Mem 실행 1분 후 측정한 CvHRC



<그림 4> Lmbench-Mem 실행 2분 후 측정한 CvHRC



<그림 5> Lmbench-All + 리눅스 커널컴파일 실행 2분 후 측정한 CvHRC

커널 컴파일에 비해 Lmbench-Mem의 수행 시간이 가상머신 수의 증가에 따라 급격하게 증가하는 이유는 커널 컴파일보다 Lmbench-Mem이 메모리 필요량이 더 많기 때문이다. 메모리 자원에 대한 경쟁이 심해졌으나 아이솔레이션이 잘 이뤄지지 않았으므로 전체적으로 수행시간이 길어지게 되었다. 뿐 만 아니라 Lmbench-Mem을 수행 시킬 때에는 가상머신을 4개 이상 수행시키게 되면 종종 메모리 자원이 부족하여 쓰래싱 현상이 발생하거나 한 두 개의 가상머신이 메모리 자원을 오랫동안 할당받지 못해 진전을 내지 못하는 경우가 발생하였다. 커널 컴파일은 같은 현상이 가상머신 5개 이상의 환경에서 관찰되었다.

다. CvHRC 측정 결과

<그림 3>은 Lmbench-Mem을 실행시킨 다음 1분후에 측정된 CvHRC이다. 그룹 인덱스는 페이지들의 히트 수를적은 오버헤드로 관리하기 위하여 미리 설정된 크기의 그룹으로 묶었으며 이때의 인덱스를 가리킨다. <그림 4>는 Lmbench-Mem을 실행시킨 다음 2분후에 측정한 CvHRC이다. <그림 3>과 <그림 4>를 비교해보면 같은 프로세스이지만 시간의 흐름에 따라 메모리 사용 양식이 다양하게 나타날 수 있음을 관찰할 수 있다. 따라서 정확하게 메모리 사용량을 측정하기 위해서는 3.2절에서 설명한 것과 같이 실시간으로 적절한 간격을 두고 CvHRC의 LRU 리스트 스캔 및 히트율을 계산하는 것이 필요하다. <그림 5>는 Lmbench-All과 리눅스 컴파일을 하나의 가상머신 내에서 동시에 실행시키면서 2분 후에 얻은 곡선이다. 이 곡선은 히트율이 작은 값에서부터 큰 값까지 완만한 곡선을 그리고 있다.

5. 결론 및 향후 연구

이 논문은 플래닛랩에서 채택한 컨테이너 기반 가상머신인 VServer에서 메모리 자원의 효율성과 아이솔레이션을 동시에 제공하기 위한 첫 단계로서 각 가상머신의 실제 메모리 사용량을 측정하였다. 이를 위해 리눅스 VServer에 히트율-곡선(Hit-Ratio-Curve)을 구현하고 이에 따른 공간 오버헤드 및 수행시간 오버헤드를 Lmbench 및 리눅스 커널 컴파일 등을 수행하여 측량하였다.

향후 연구로서는 측정된 페이지 사용량을 바탕으로 하는 메모리 관리 기법을 제안할 것이다. 여기에서 해결해야 할 문제들은 페이지 필요량 지점의 기울기 결정 및 리눅스 커널의 기존 정책과의 관계 등이 있다. 효과적인 메모리 관리 기법을 통하여 가상 머신들은 필요량을 보장 받으면서 다른 가상머신에게는 최소한의 영향을 미치게 되어 메모리 자원의 아이솔레이션을 달성할 수 있게 될 것으로 예상한다.

[참고 문헌]

[1] C. Waldspurger, "A Memory Resource Management in VMware ESX Server," In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation,* 2002.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the Art of Virtualization," In *Proceedings of the 19th ACM symposium on Operating systems,* 2003.

[3] Linux VServers Project. http://linux-vserver.org/.

[4] D. Sullivan and M. Seltzer, "Isolation with Flexibility: A Resource Management Framework for Central Servers," In *Proceedings of the USENIX Auual Technical Conference,* 2000.

[5] PlanetLab project. http://www.planet-lab.org/.

[6] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experience Building PlanetLab," In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation,* 2006.

[7] G. Banga, P. Druschel and J. C. Mogul, "Resource Containers: A New Facility for Resource Management in Server Systems," In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation,* 1999.

[8] D. Price and A. Tucker, "Solaris zones: Operating system support for consolidating commercial workloads," In *Proceedings of the 18th USENIX LISA Conference,* 2004.

[9] SWSOFT. Virtuozzo Technology. http://www.sw-soft.com.

[10] J. Kim, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho and C. Kim, "A low-overhead high-performance unified buffer management scheme that exploits sequential and looping references," In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation,* 2000.

[11] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou and S. Kumar, "Dynamically Tracking Miss-Ratio-Curve for Memory Management," In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems,* 2004.

[12] Y. Fu, J. Chase, B. Chun, S. Schwab and A. Vahdat, "SHARP: An architecture for Secure Resource Peering," In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[13] L. McVoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis," In *Proceedings of the USENIX Annual Technical Conference,* 1996.