제43회 정기총회 및 동계학술발표회

2016.12.21(수)~12.23(금)
강원도 보광휘닉스파크

목 차

제43회 정기총회 및 동계학술발표회 후원

골드
GWCVB
네이버

실버
삼성SDS

브론즈
SK Telecom

제43회 정기총회 및 동계학술발표회 Program Book
인 쇄 2016년 12월 16일
발 행 2016년 12월 20일
발행인 홍충선
편집인 김지홍
발행처 사단법인 한국정보과학회
http://www.kiise.or.kr
서울시 서초구 방배로 76
(방배동, 머리재빌딩 401호)
Tel. 1588-2728
Fax. 02-521-1352
E-mail. kiise@kiise.or.kr
인쇄처 한림원(주)
가 격 비매품

P2.1-43 Juliet Test Suite를 활용한 LLVM Clang 정적 분석기의 평가
박영관・진홍주・전주라・표창우(홍익대)
P2.1-44 Mobius IoT 플랫폼을 연계한 강의 환경 관리 시스템
이여울・손민석・송기환・이철웅・노병희(아주대)
P2.1-45 Unity 게임 엔진의 Fungus Asset을 이용한 스토리 시각화
이재현, 정윤경(성균관대)
P2.1-46 WESP: 스피치 연습을 위한 웹 어플리케이션
한나영・서지은・BeliseFundi・김명(이화여대)
P2.1-47 리눅스 KVM 상 가상머신의 메모리 공유 분석
안승민・이민호・엄영익(성균관대)
P2.1-48 비선점형 혼합 임계 실시간 시스템에서의 이월 작업에 대한 분석
윤지아, 이진규(성균관대)
P2.1-49 영・유아 안전사고 방지 시스템
우동혁・최종필(한국산기대)
P2.1-50 SSD 내부의 쓰기 버퍼 활용률 분석
이태형・이민호・엄영익(성균관대)
P2.1-51 아두이노-라즈베리 파이 기반의 미세먼지 측정 애플리케이션 개발 및 분석
장세훈(한국외국인학교)
P2.1-52 '오브젝트 풀링'기법을 이용한 가비지 콜렉터에 대한 최적화
양준석(선린인터넷고), 강미영(고려대)
P2.1-53 x86 아키텍쳐와 x64 아키텍쳐 혼합을 이용한 코드 난독화
이태양(한국정보기술연구원)
P2.1-54 순열 엔트로피를 이용한 한국프로야구 팀의 순위 변동
조찬우・김휘재・조찬형・하석규(용인고), 조환규(부산대)
P2.1-55 융합교육을 적용한 화학 원소 기호 스마트 러닝 웹앱
이재익・김태규(서울상문고)

P2.2 국방소프트웨어	12.21(수) 12:30~14:00, 볼룸

▷평가위원 : 진현욱(건국대)

P2.2-01 항공기 실시간 데이터 관리 기법 사례 분석
권기봉・신상면(한국항공우주산업)
P2.2-02 보안성 확보를 위한 무기체계 내장형 소프트웨어 개발 프로세스 개선 연구
이원철・김강현・이승현(고려대)
P2.2-03 국방 무기체계를 위한 컴포넌트 기반 소프트웨어 개발 방법
남송현・고정환・권철희(LIG넥스원)
P2.2-04 무기체계의 임베디드 안전필수 SW 개발 생명주기 연구
이승현・김강현・이원철(고려대)
P2.2-05 차량 보조 시스템을 위한 안개 제거
이건혁・최광남(중앙대)
P2.2-06 데이터 특징 분석 기반 TPR*-tree 성능 향상 방안
홍재기・전승우・김점수・홍봉희(부산대)

P2.3 사물인터넷I	12.21(수) 12:30~14:00, 볼룸

▷평가위원 : 이규택(KEIT), 손영성(ETRI)

P2.3-01 상황 인지에 기반한 추천 정보의 해석 방법
Syed Imran Ali・이승룡(경희대)

리눅스 KVM 상 가상머신의 메모리 공유 분석

안승민[1], 이민호[2], 엄영익[3]
[1]성균관대학교 경제대학
[2]성균관대학교 대학원 전자전기컴퓨터공학과
[3]성균관대학교 소프트웨어대학
{smahn9123, minhozx, yieom}@skku.edu

Analysis of Memory Share on Linux KVM based Virtual Machines

Seungmin Ahn[1], Minho Lee[2], and Young Ik Eom[3]
[1]College of Economics, Sungkyunkwan University
[2]Dept. of Electrical and Computer Engineering, Sungkyunkwan University Graduate
[3]College of Software, Sungkyunkwan University

요 약

가상화 환경에서 메모리 자원의 효율적 사용은 가상머신의 시스템 성능에 큰 영향을 미치게 된다. 가상머신 메모리의 절약을 위한 대표적 커널 모듈인 리눅스 KSM은 호스트의 페이지를 병합으로써 메모리 공유를 지원한다. 가상머신의 메모리 공유는 가상머신 내부 공유와 가상머신 간 공유라는 두 방식으로 발생한다. 이 논문은 대중적으로 사용되는 Ubuntu, Windows 7, Windows 10이 KVM의 가상머신으로 동작하는 동안 KSM으로 발생하는 두 방식의 메모리 공유를 측정하여 비교 및 분석한다. 또한 같은 운영체제 간의 메모리 공유가 다른 운영체제 간의 메모리 공유보다 크게 발생함을 실험으로써 확인한다.

1. 서 론

컴퓨터시스템에서 메모리는 시스템 성능에 큰 영향을 미치는 핵심적인 자원이다. 메모리를 효율적으로 사용할수록 더 많은 프로세스를 가동하거나, 페이지 캐시를 통해 스토리지에 접근하는 입출력 오버헤드를 감소시킬 수 있기 때문이다[1]. 특히 가상화 환경은 중복 입출력 스택 구조를 가지기 때문에 가상화 시스템의 메모리를 효율적으로 사용하기 위한 연구가 활발하다[1-3]. 예를 들어, 리눅스의 KSM(Kernel Same-page Merging) 모듈은 가상머신들의 페이지를 스캔하여 같은 내용을 가지는 페이지를 병합함으로써 호스트에서의 메모리 공유를 가능하게 한다.

최근 가상화 기술이 적용된 클라우드 컴퓨팅 환경의 서버에서 로드 밸런싱(Load Balancing)은 가상머신들의 CPU, 메모리, 네트워크 통신량 등의 자원 사용량을 계산하여 가상머신을 호스트 간 이주시킴으로써 최적화된 배치를 가능하게 하는 기술이다[4]. 가상머신의 메모리 공유율을 로드 밸런싱 기술에 반영함으로써 가상화 서버는 최소한의 메모리를 통해 여러 가상머신들을 운용할 수 있다. 이를 위해 가상머신의 메모리 사용 패턴을 분석할 필요가 있다.

가상머신의 메모리 공유는 가상머신 내부와 가상머신 간의 공유 두 가지로 나눌 수 있다[5]. 본 논문에서는 KVM 기반 가상머신의 메모리 공유가 어떻게 발생하는지 확인하고 두 가지 형태의 메모리 공유의 비중을 측정하여 비교 및 분석한다. 본 논문의 구성은 다음과 같다. 2장에서 리눅스 KSM 모듈과 메모리 공유의 두 가지 방식을 설명한다. 3장에서 KVM 기반 가상머신의 메모리 공유의 일반적 패턴과 가상머신 간 메모리 공유를 분석한다. 4장에서는 결론 및 향후 연구 방향을 제시한다.

이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.R0126-15-1082, (ICBMS-총괄) ICBMS(IoT, 클라우드, 빅데이터, 모바일, 정보보호) 핵심 기술 개발 사업 총괄 및 엑사스케일급 클라우드 스토리지 기술 개발)

2. 배경지식

2.1 KSM

리눅스 커널 2.6.32 버전부터 KSM이라는 페이지 공유 모듈이 지원된다[6]. KSM은 주기적으로 시스템의 메모리에 할당된 페이지를 스캔하여 내용이 일치하는 페이지들을 찾아 하나의 페이지로 병합한다. 이렇게 함으로써 시스템은 메모리를 효율적으로 사용할 수 있다. 스캔 및 병합의 대상이 되는 페이지의 관리를 위해 두 종류의 레드블랙 트리의 자료구조가 사용된다.

- 안정 트리(Stable Tree): 페이지 병합으로 인해 공유된 페이지들이 저장된다.
- 불안정 트리(Unstable Tree): 공유 가능성이 낮거나 공유될 가능성이 있는 후보 페이지들이 저장된다.

KSM은 스캔하는 페이지마다 먼저 같은 내용의 페이지가 안정 트리에 저장되었는지 검사하고, 저장되었다면 그곳으로 페이지를 병합한다. 또한, 페이지의 내용이 안정적인지 확인하기 위한 32비트 체크섬(Checksum) 값

이 존재한다. 체크섬이 지난 검사와 불일치하면 다음 스캔 주기로 미루며 일치한다면 불안정 트리의 페이지들과 비교하거나 불안정 트리에 저장한다. KSM은 MADV_MERGEABLE 옵션이 지정된 *madvise* 시스템 호출로 표시한 가상 메모리 영역에 매핑되는 호스트 메모리 영역에 대해서만 스캔과 병합을 진행한다. KVM 가상머신은 이 기능을 사용하므로 메모리 검사의 대상이 된다. 본 논문의 실험에서 호스트의 기타 프로세스는 *madvise*를 통해 KSM을 호출하지 않았기에 오로지 가상화 환경만의 메모리 공유를 측정하는데 용이하였다.

2.2 가상화 시스템의 두 가지 메모리 공유 방식

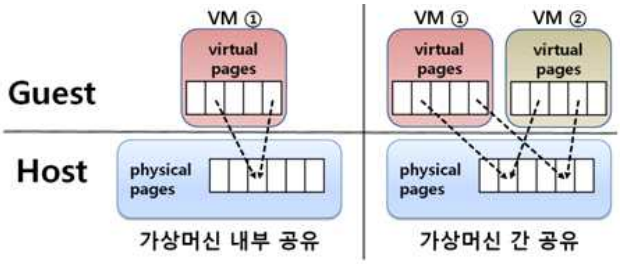

그림 1 가상머신 내부 공유 및 가상머신 간 공유

모든 가상머신의 메모리 공유는 그림 1에 나타난 두 가지 형태의 메모리 공유 방식 중 하나에 해당한다[5].

① 가상머신 내부 공유(VM-internal Share): 한 가상머신 내부의 서로 다른 가상 페이지들이 호스트의 같은 메모리 페이지로 매핑된다.
② 가상머신 간 공유(Inter-VM Share): 서로 다른 가상머신의 가상 페이지들이 호스트의 같은 메모리 페이지로 매핑된다.

가상화 환경에서 메모리 공유를 측정하려면 두 가지 방식을 분리하여 접근해야 한다. 같은 호스트에 가동되는 운영체제의 조합에 따라 메모리 공유 정도의 차이가 발생할 가능성이 높기 때문이다. 예를 들어 같은 계열의 운영체제인 Windows 7과 Windows 10은 메모리 공유 가능성이 높은 라이브러리가 다수 있지만, 리눅스 Ubuntu의 라이브러리와는 그렇지 않기 때문이다. 호스트에서 확인되는 모든 페이지 공유는 두 방식을 합산한 것이므로, 다음의 간단한 공식을 사용 가능하다.

$$S_{Total} = S_{Internal} + S_{Inter},\quad S_{Internal} = \sum_{i=1}^{n} s_i$$

- S_{Total}: 호스트가 파악하는 절약된 메모리의 총량
- $S_{Internal}$: 각 가상머신 내부 공유로 절약한 메모리 총량
- S_{Inter}: 가상머신 간 공유를 통해 절약한 메모리 총량
- s_i: 가상머신 i가 내부 공유로 절약한 메모리의 양

3. 메모리 공유 측정

3.1 실험 환경

/sys/kernel/mm/ksm 디렉터리에서 KSM의 메모리 스캔 속도를 조절할 수 있다. pages_to_scan 파일의 값을 100으로, 그리고 sleep_millisecs 파일의 값을 200으로 지정하여 KSM이 200 밀리 초마다 100개의 페이지를 스캔하도록 설정했다. 동일한 디렉터리에 위치한 다음 4개의 파일은 현재 KSM의 스캔 및 병합 과정에서 기록되고 있는 관련 변수를 실시간으로 표시한다. 가상머신의 메모리 공유를 측정하는 이후의 모든 실험은 4개의 변수의 변동 추이를 실시간으로 기록하는 쉘 스크립트(shell script)를 가동하는 방식으로 진행하였다.

표 1 실험 환경

CPU	Intel Core i7-6700 CPU @3.40GHz x8	
Memory	Host / Guest	8 GB / 2 GB of DRAM
OS	Host	Ubuntu 14.04 LTS 64-bit
	Guest 1	Ubuntu 14.04 LTS 64-bit
	Guest 2	Windows 7 64-bit
	Guest 3	Windows 10 64-bit

- pages_shared: 안정 트리에 속한 페이지의 개수
- pages_unshared: 불안정 트리에 속한 페이지의 개수
- pages_volatile: 잦은 내용 변동으로 인해, 페이지 병합을 위한 비교연산이 불가능한 페이지의 개수
- pages_sharing: 페이지 병합으로 인해 메모리에서 해제된 (즉 현재 절약되고 있는) 페이지의 개수

3.2 가상머신의 내부 메모리 공유 측정

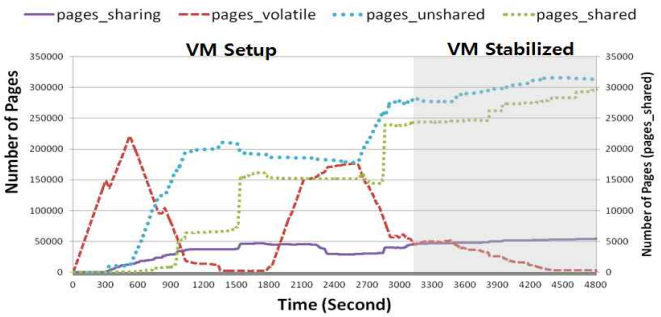

그림 2 Ubuntu 구동 후 메모리 공유 양상

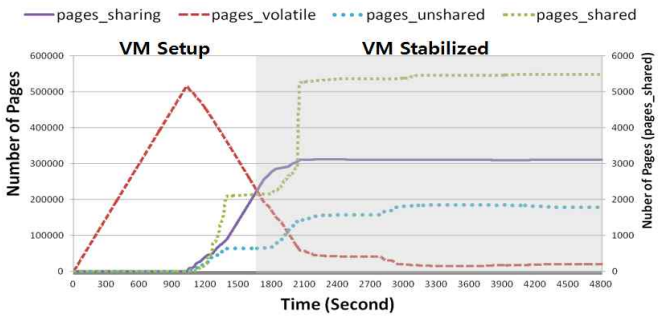

그림 3 Windows 7 구동 후 메모리 공유 양상

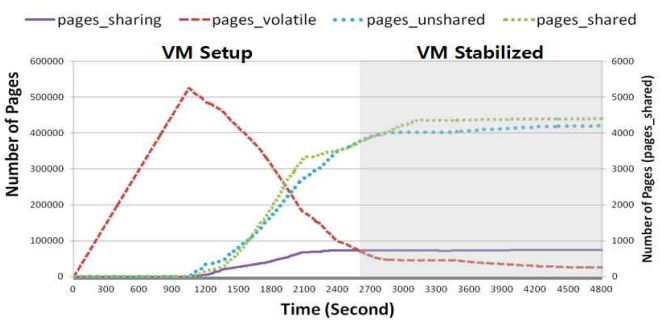

그림 4 Windows 10 구동 후 메모리 공유 양상

먼저 Ubuntu, Windows 7, Windows 10이 각각 설치된 3개의 가상머신을 개별 가동하여 KSM에 의한 메모리 공유의 실제 전개 과정을 실험을 통해 관찰하였다. 그림 2, 3, 4는 3개의 가상머신이 부팅된 후, KSM에 의해 발생하는 메모리 공유를 시간 순으로 보여준다. 부팅 직후 pages_volatile의 수치가 일정하게 증가한 후 다시 감소한다(Ubuntu는 이 현상이 두 번 나타난다). 부팅 중 가상 디스크 이미지로부터 운영체제의 핵심 데이터가 가상머신에 할당된 메모리로 로드되는 과정에서 잦은 내용 변화가 일어나는 다수의 페이지가 검출되기 때문이다. 일정 시간 이후 핵심 데이터의 로딩이 완료되어 페이지 내용 변화가 감소하는 기점부터 페이지 병합이 본격적으로 진행된다. 안정 트리와 불안정 트리의 크기가 증가하며, 해제되는 페이지의 개수가 증가한다. pages_volatile이 pages_sharing 보다 낮아지는 기점을 기준으로 전반부를 구성단계, 후반부를 안정단계로 분류하였다. 최대 4,800초까지 다른 동작 없이 대기하여 가상머신 안정단계에서 나타나는 페이지 절약의 최대치를 측정했다. 각 경우 모두 가상머신을 단독으로 가동했으며, 측정된 가상머신별 내부 공유를 통한 메모리 절약량은 표 2와 같다. 호스트 메모리의 페이지 크기는 4 KB이므로 메모리 절약량은 pages_sharing에 이를 곱한 것과 같다.

표 2 운영체제 별 내부 메모리 공유

	Ubuntu 14.04	Windows 7	Windows 10
메모리 절약량	212.36 MB	1218.16 MB	293.04 MB

3개의 운영체제 중 Windows 7 가상머신의 메모리 공유가 가장 크게 발생함을 확인할 수 있다. 이는 가상머신에 1.2 GB 가량 적은 메모리를 할당하더라도 가상머신 안정단계에 진입하면 이전과 동일한 성능을 나타낼 수 있음을 의미한다.

3.3 가상머신 간 메모리 공유 측정

가상머신 간 메모리 공유 정도를 관찰하기 위해 이번에는 3개 가상머신을 둘씩 그룹지어 가동하는 세 번의 실험을 진행했다. 가상머신 안정단계에서 그룹 당 절약된 메모리의 총량(S_{Total})을 구하고, 여기에서 위에서 제시한 가상머신 별 내부 절약된 메모리($S_{Internal}$)를 제외함으로써 가상머신 간 공유를 통해 절약된 메모리의 총량(S_{Inter})을 구할 수 있었다.

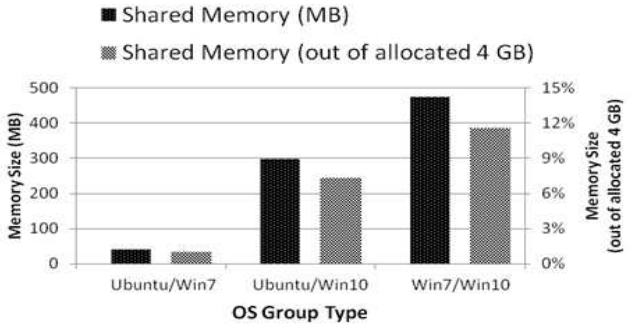

그림 5 가상머신 그룹별 가상머신 간 공유

그림 5에서 Win7 / Win10 그룹의 경우 가상머신 간 공유가 다른 두 그룹의 가상머신 간 공유보다 높은 것을 확인할 수 있다. 반면 Ubuntu / Win7 그룹의 경우 전체 할당된 4 GB 중 1% 이하만이 가상머신 간 공유로 절약됨을 보여준다. 로드 밸런싱 과정에서 같은 계열의 운영체제를 사용하는 가상머신을 그룹으로 묶어 배치하면 호스트의 메모리를 절약할 수 있음을 확인할 수 있다.

표 3 가상머신 그룹별 2가지 공유 방식 비교

	$S_{Internal}$	S_{Inter}
Ubuntu / Win7	1430.52 MB	40.75 MB
Ubuntu / Win10	505.4 MB	298.92 MB
Win7 / Win10	1511.2 MB	474.34 MB

표 3은 두 가지 공유 방식이 메모리 절약에 기여하는 정도를 비교한다. 가상머신 내부 공유가 가상머신 간 공유보다 메모리 절약에 더 크게 기여함을 알 수 있다.

4. 결론 및 향후 연구

KVM 가상머신의 메모리 공유가 어떻게 전개되는지 관찰하였고, 두 방식의 메모리 공유의 비중을 측정하여 가상머신 내부 공유가 가상머신 간 공유보다 메모리 절약에 더 많이 기여함을 증명했다. 또한 서로 같은 운영체제 간의 메모리 공유가 다른 운영체제 간의 메모리 공유보다 더 크게 나타남도 확인할 수 있었다. 본 논문은 가상머신에서 추가 어플리케이션의 구동 없이 실험을 진행했는데, 같은 어플리케이션을 구동하는 경우 더 많은 메모리 공유가 가능하다. 그러나 페이지 공유의 근원이 운영체제와 어플리케이션 중 어느 곳에 위치하는지 파악해야 하므로, 향후 리눅스 커널 코드를 수정해 더 심층적인 가상머신의 메모리 공유를 연구할 것이다.

참고문헌

[1] G. Milos, D. Murray, and S. Hand, “Satori, Enlightened Page Sharing,” *Proc. of the USENIX Annual Technical Conference*, pp. 11-14, 2009.

[2] D. Gupta, S. Lee, M. Vrable, S. Savage, A. Snoeren, G. Varghese, G. Voelker, and A. Vahdat, “Difference Engine: Harnessing Memory Redundancy in Virtual Machines,” *Communications of the ACM*, Vol. 53, No. 10, pp. 85-93, 2010.

[3] K. Miller, F. Franz, M. Rittinghaus, M. Hillenbrand, and F. Bellosa, “XLH: More Effective Memory Deduplication Scanners through Cross-layer Hints”, *Proc. of the USENIX Annual Technical Conference*, pp. 279-290, 2013.

[4] A. Beloglazov and R. Buyya, “Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under Quality of Service Constraints,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, No. 7, pp. 1366-1379, Jul. 2013.

[5] S. Barker, T. Wood, P. Shenoy, and R. Sitaraman, “An Empirical Study of Memory Sharing in Virtual Machines,” *Proc. of the USENIX Annual Technical Conference*, pp. 273-284, 2012.

[6] A. Arcangeli, I. Eidus, and C. Wright, “Increasing Memory Density by Using KSM,” *Proc. the Ottawa Linux Symposium*, pp. 19-28, 2009.