이중 스케줄링으로 인한 캐시 성능 저하를 이용한 가상화 환경 탐지 기법

차현수 [1], 장대희 [2]
[1] 경희대학교 대학원 석사과정
[2] 경희대학교 컴퓨터공학과 교수

soo@khu.ac.kr, daehee87@ khu.ac.kr

Double-Scheduling Breaks Cache: A Virtual Machine Detection Method Using Cache Performance Degradation

Hyun-soo Cha[1], Dae-hee Jang[2]
[1]Dept. of Computer Engineering, Kyung Hee University
[2]Dept. of Computer Engineering, Kyung Hee University

요 약

가상화 환경에서는 호스트 스케줄러와 게스트 스케줄러가 동시에 작동하는 이중 스케줄링(double-scheduling) 현상이 발생한다. 이는 호스트 스케줄러(하이퍼바이저 스케줄러)가 가상 CPU 를 물리 CPU 에 스케줄링하고, 게스트 스케줄러가 게스트 OS 의 태스크를 가상 CPU 에 스케줄링하는 구조로, 두 개의 스케줄러가 존재하기 때문이다. 본 연구는 이러한 이중 스케줄링이 캐시 성능에 미치는 영향을 실험적으로 분석하여, 캐시 미스율이 베어메탈 대비 **약 67 배에서 187 배**까지 증가함을 입증하였다. AMD Ryzen 8845HS 와 Intel Xeon Gold 6140 프로세서에서 수행한 실험에서 베어메탈 환경은 0.46%~0.61%의 안정적인 캐시 미스율을 보인 반면, VMware 환경에서는 45.22%, KVM 환경에서는 86.51%의 높은 미스율을 기록했다. 또한 가상화 환경에서는 15.83~26.80 의 높은 표준 편차를 보였다. 이러한 차이를 기반으로 본 논문은 사용자 레벨 (Ring 3)에서 VM-exit 없이 가상화 환경을 탐지하는 기법을 제안한다.

1. 서론

현대 악성코드는 분석 환경을 회피하기 위해 정교한 가상화 탐지 기법을 사용한다. 보안 연구자들이 악성코드 분석을 위해 샌드박스와 가상 머신을 광범위하게 활용함에 따라, 악성코드 제작자들은 이를 탐지하여 실제 동작을 숨기는 전략을 발전시켜왔다. 이러한 회피 기법을 이해하고 대응하기 위해서는 가상화 환경의 근본적 특성과 탐지 가능한 차이점을 명확히 파악해야 한다.

가상화 기술은 자원 활용도 향상과 관리 효율성 제고를 가능하게 한다. 그러나 가상화 환경에서는 필연적으로 이중 스케줄링(double-scheduling) 문제가 발생한다. 호스트 스케줄러(하이퍼바이저)가 가상 CPU 를 물리 CPU 에 스케줄링하고, 게스트 스케줄러가 게스트 OS 의 태스크를 가상 CPU 에 스케줄링하는 구조로, 두 개의 독립적인 스케줄러가 존재한다. [1]

호스트와 게스트 스케줄러 간의 정보 공유 부재는 성능 문제를 야기한다. 두 스케줄러가 동시에 작동하면서 물리 코어 간 이동이 증가하고, 스케줄링 정보가 두 스케줄러 사이에서 손실된다.

가상화 탐지 기법 역시 다양하게 연구되었으나, 대부분 VM-exit 를 유발하거나 특권 명령어를 사용하는 방식이었다. Franklin 등은 제어 레지스터 명령을 수십만 번 반복하여 VMM 오버헤드를 측정했으나[6], 비실용적이고 탐지 가능하다는 한계가 있다. D'Elia 등은 L3 캐시 기반 Prime+Probe 기법을 제안했으나[5], eviction set 구성의 어려움과 긴 실행 시간이 문제였다.

본 연구는 이중 스케줄링이 캐시 성능에 미치는 영향을 분석하고, 이를 기반으로 한 가상화 탐지 기법을 제안한다. 제안 기법은 특권 명령어나 VM-exit 없이 사용자 레벨에서 동작하며, 캐시 미스율의 차이를 통해 가상화 환경을 탐지한다.

2. 배경 및 관련 연구

가상화 플랫폼마다 서로 다른 스케줄러 구조를 채택한다. KVM 에서는 리눅스가 호스트에서 실행되며 하이퍼바이저 역할을 수행하므로, 리눅스 스케줄러가 KVM 가상 머신의 가상 CPU 를 호스트의 물리 CPU 에 스케줄링한다. VM 내부에서도 리눅스가 실행되면 리눅스 스케줄러가 태스크를 VM 의 가상 CPU 에 스케줄링한다. 이 접근법은 리눅스 스케줄러의 개선사항이 호스트와 게스트 모두에 자동으로 적용되는 장점이 있으나, 가상화 특화 기능 추가가 어렵다.

Xen 은 호스트에서 직접 실행되며, Xen 스케줄러가 가상 CPU 를 물리 CPU 에 스케줄링한다. VM 내부에서는 리눅스 스케줄러가 태스크를 가상 CPU 에 할당한다. Xen 은 최근 CREDIT2 를 기본 스케줄러로 전환했다.

VMware 는 Gang Scheduler 접근법을 사용하여 멀티 vCPU 머신의 모든 vCPU 를 함께 묶어서 물리 코어에 스케줄링한다.

캐시를 이용한 사이드 채널 공격은 정보 보안 분야에서 지속적으로 연구되어 왔다. Yarom 과 Falkner 는 Flush+Reload 기법을 통해 LLC 에서 cross-core 공격이 가능함을 보였다.[4], Liu 등은 가상화 환경에서 공유되는 LLC 를 타겟으로 하는 Prime+Probe 기법을 제시했으며[3], inclusive LLC 특성과 large-page mapping 을 이용해 메모리 공유나 동일 코어 배치 없이도 cross-VM 공격이 가능함을 입증했다.

Vateva-Gurova 등[2]은 하이퍼바이저의 로드 밸런싱이 활성화될 때 L1 캐시 타이밍 사이드 채널의 신뢰성이 크게 감소함을 실험적으로 입증했다. 이는 로드 밸런싱으로 인한 빈번한 vCPU 마이그레이션이 캐시 공유를 방해하며, 역설적으로 이 현상이 가상화 환경의 특징적 지표가 될 수 있음을 시사한다.

기존 가상화 탐지 기법들은 크게 세 가지 범주로 분류된다. 첫째, VM-exit 를 유발하는 특권 명령어를 이용한 방법이다. CPUID 명령어 실행 시간을 측정하거나, VMX 명령어의 반환값 차이를 이용한다[8].

둘째, 하드웨어 특성의 차이를 이용한 방법이다. TLB 미스 지연이나 LLC 미스 패널티의 차이를 측정한다[8]. 이러한 방법은 CPU 아키텍처에 의존적이며, 새로운 하드웨어에서는 동작하지 않을 수 있다.

셋째, 스케줄링 특성을 이용한 방법이다. Zhi 등은 CPU-bound 코드의 실행 시간을 확률 변수로 모델링하고, 분산, 첨도, 비대칭도를 조합한 다차원 지표로 VM 을 판별했다[10]. 그러나 이러한 통계적 방법은 시스템 부하에 민감하고 오탐률이 높다.

3. 이중 스케줄링이 캐시 성능에 미치는 영향

이중 스케줄링이 캐시 성능에 미치는 영향을 분석하기 위해, 캐시 워킹셋과 스케줄링 간섭의 관계를 모델링한다. 프로세스 P 가 시간 t 동안 접근하는 메모리 페이지의 집합을 워킹셋 W(P,t)라 정의할 때, 베어메탈 환경에서 프로세스가 코어 C 에 고정되어 있다면 캐시 미스율은 다음과 같이 표현된다:

M_bare = f(|W(P,t)|, C_size, A_pattern, R_policy)

여기서 C_size 는 캐시 크기, A_pattern 은 메모리 접근 패턴, R_policy 는 캐시 교체 정책이다. 프로세스가 동일 코어에서 지속적으로 실행되면, 시간 지역성(temporal locality)과 공간 지역성(spatial locality)이 유지되어 낮은 미스율을 보인다.

반면 가상화 환경에서는 두 가지 추가적인 불확실성이 도입된다. 첫째, vCPU 마이그레이션 확률 P_mig(t)가 존재한다. 이는 호스트 스케줄러의 로드 밸런싱 정책과 시스템 부하에 따라 동적으로 변한다. 둘째, 호스트 스케줄러의 타임 슬라이스 T_host 와 게스트 스케줄러의 타임 슬라이스 T_guest 가 중첩되어 예측 불가능한 간섭이 발생한다.

이러한 요인들을 고려한 가상화 환경의 캐시 미스율은:

M_vm = M_bare + ∫[P_mig(t) × C_flush(t)] dt + ∑[I_sched(i) × C_evict(i)]

여기서 C_flush(t)는 시간 t 에서의 마이그레이션으로 인한 캐시 플러시 비용, Sched(i)는 i 번째 스케줄링 간섭의 강도, C_evict(i)는 해당 간섭으로 인한 캐시 축출 비용이다.

vCPU 마이그레이션은 캐시·마이크로아키텍처의 코어 로컬 상태 전반에 영향을 미친다. 프로세스가 물리 코어 A 에서 코어 B 로 이동할 때:

- L1 캐시 무효화: L1 명령어 캐시(32-64KB)와 L1 데이터 캐시(32-64KB)의 내용이 무효화된다. L1 캐시 접근 지연은 4-5 사이클인 반면, L2 캐시는 12-15 사이클, LLC 는 40-50 사이클이므로, L1 캐시 미스는 10 배 이상의 지연을 야기한다.
- L2 캐시 재구축: 코어별 프라이빗 L2 캐시(256KB-1MB)도 새로 채워져야 한다. L2 캐시는 L1 보다 크기가 크므로 재구축에 더 많은 시간이 소요된다.
- LLC 슬라이스 재매핑: 주소→슬라이스 매핑은 코어 이동으로 바뀌지 않는다. 다만 (대부분 non-inclusive 인) LLC 에서 코어 이동 시 링/메시 hop 변화로 지연이 달라지고, 소켓/CCD 경계를 넘으면 L3 지역성이 약화되어 접근 비용이 커진다.
- 마이크로아키텍처 상태 손실: 분기 예측기, 프리페처, TLB 등 코어별 마이크로아키텍처 구조의 학습된 패턴이 모두 손실된다.

GhatrehSamani 등의 연구에 따르면[7], CPU 집약적 워크로드에서 VM 오버헤드는 베어메탈 대비 2 배 이상의 실행 시간을 보였다. FFmpeg 같은 CPU 집약적 애플리케이션에서는 CPU pinning 이 주는 이점이 관찰되지 않았는데, 이는 호스트 레벨의 스케줄링 간섭이 게스트 레벨의 최적화를 무효화함을 보여준다.

호스트와 게스트 스케줄러의 독립적 동작은 예측 불가능한 간섭 패턴을 생성한다. Fair-share 스케줄러의 경우, vCPU 들에게 CPU 시간을 공평하게 배분하기 위해 주기적으로 로드 밸런싱을 수행한다. KVM 의 Completely Fair Scheduler(CFS)는 기본적으로 100ms 의 스케줄링 주기를 가지며, 각 vCPU 는 이 주기 내에서 weight 에 비례한 타임 슬라이스를 받는다.

게스트 OS 내부에서도 동일한 CFS 가 작동하므로, 두 스케줄러의 주기가 중첩될 때 복잡한 간섭 패턴이 발생한다. 호스트 스케줄러가 vCPU 를 선점할 때, 게스트 스케줄러는 이를 인지하지 못하고 계속 스케줄링 결정을 내린다. 이로 인해 게스트가 캐시 친화적으로 배치한 태스크들이 실제로는 서로 다른 물리 코어에서 실행될 수 있다.

우리의 실험 데이터는 이러한 이론적 예측을 뒷받침한다. 베어메탈에서는 표준편차가 0.04~0.15 범위로 안정적인 반면, VM 환경에서는 16.92%~26.24%의 높은 표준편차를 보였다. 이는 동일한 워크로드임에도 실행 시점의 스케줄링 상태에 따라 캐시 성능이 크게 달라짐을 의미한다.

4. 실험 설계 및 구현

실험은 두 가지 대표적인 하드웨어 플랫폼에서 수행되었다. 첫 번째 플랫폼은 AMD Ryzen 8845HS 프로세서로, Zen 4 아키텍처 기반의 8 코어 16 스레드 구성을 가지며, 32MB 의 L3 캐시와 코어당 1MB L2 캐시, 32KB L1 명령어 캐시 및 32KB L1 데이터 캐시를 갖는다. 두 번째 플랫폼은 Intel Xeon Gold 6140 프로세서로, Skylake-SP 아키텍처 기반의 18 코어 36 스레드 구성에 24.75MB L3 캐시를 가진다. 이 두 플랫폼은 서로 다른 제조사, 아키텍처, 캐시 구조를 가지므로 제안 기법의 일반성을 검증하기에 적합하다.

가상화 환경으로는 VMware Workstation Pro 17 과 KVM/QEMU 9.0 을 사용했다. 각 VM 은 2 개의 vCPU 와 4GB 메모리를 할당받았으며, 게스트 OS 는 Ubuntu 22.04 LTS 를 사용했다. VM 설정에서 CPU 친화도나 NUMA 설정은 기본값을 유지하여 일반적인 사용 환경을 반영했다. 추가로 Akamai 클라우드 환경에서도 실험을 수행하여 실제 프로덕션 클라우드 환경에서의 동작을 확인했다.

캐시 성능을 측정하기 위해 C 언어로 구현된 마이크로벤치마크를 개발했다. 워크로드는 각 스레드가 독립적인 메모리 배열에 랜덤 접근을 수행하는 구조이다. 배열 크기는 스레드당 32KB 로 설정하여 L1 데이터 캐시(32KB)와 동일한 크기로 구성했다. 이는 스레드 마이그레이션 시 캐시 콘텐츠 교체를 최대화하기 위해서이다. 메모리 접근은 랜덤 인덱싱을 사용하여 하드웨어 프리페처의 학습을 방해했다. 각 스레드는 sched_setaffinity()를 통해 특정 코어에 고정된다.

캐시 미스율은 하드웨어 성능 카운터를 통해 직접 측정했다. Linux perf 도구를 사용하여 다음 이벤트들을 수집했다:

cycles(CPU 사이클 수), instructions(실행된 명령어 수), cache-references(캐시 참조 횟수), cache-misses(캐시 미스 횟수).

미스율은 (cache-misses / cache-references) × 100 으로 계산되며, 이는 LLC 레벨의 미스율을 나타낸다.

5. 실험 결과 및 분석

실험 결과는 이중 스케줄링이 캐시 성능에 미치는 영향을 명확히 보여준다. 베어메탈 환경에서 AMD Ryzen 8845HS 는 평균 0.61%(표준편차 0.04)의 낮은 캐시 미스율을 보였고, Intel Xeon Gold 6140 은 평균 0.46%(표준편차 0.14)를 기록했다. 이는 현대 프로세서의 대용량 캐시와 정교한 프리페처가 베어메탈 환경에서는 효과적으로 작동함을 보여준다.

반면 가상화 환경에서는 다른 양상을 보였다. VMware on Ryzen 환경에서는 평균 45.22%(표준편차 15.83)의 미스율을 기록하여 베어메탈 대비 약 75 배 증가했다. KVM on Xeon 환경은 더욱 극단적으로 평균 86.51%(표준편차 26.80)의 미스율을 보여 약 189 배 증가했다. Akamai 클라우드 환경은 평균 26.1%(표준편차 13.37)의 중간 수준을 보였다.

환경 (Environment)	평균 (Mean)	중앙값 (Median)	표준편차 (Std Dev)
Baremetal - Ryzen 8845HS	0.606%	0.61%	0.038
VMware - Ryzen 8845HS	45.217%	54.12%	15.836
KVM - Xeon Gold 6140	86.507%	95.53%	26.803
Baremetal - Xeon Gold 6140	0.458%	0.43%	0.135
Cloud - Akamai	26.175%	20.415%	13.372

[표 1] 환경별 Cache miss rate 실험 결과

100 회 측정 데이터를 시계열로 분석하면 각 환경의 고유한 패턴이 드러난다. 베어메탈 Ryzen 환경은 전 구간에서 0.44-0.68% 범위 내의 매우 일정한 미스율을 유지했다. 표준편차가 0.04 로 극히 낮아 스파이크조차 관찰되지 않았으며, 이는 물리 하드웨어의 안정적인 캐시 성능을 보여준다.

VMware 환경에서는 10.94-60.52% 범위의 큰 변동을 보였다. 전체 측정 기간 동안 평균 45.2%의 미스율을 유지하면서 지속적으로 큰 폭의 변동을 보였는데, 이는 VMware 의 주기적인 vCPU 재배치를 반영하는 것으로 해석된다.

KVM 환경은 극단적인 이원화 패턴을 보였다. 대부분의 측정(약 90%)에서 90% 이상의 높은 미스율을 기록했으나, 8 회의 측정(43, 68-70, 90-93 번째)에서는

갑자기 0.12-8.8%의 매우 낮은 미스율을 보였다. 이러한 급격한 변화는 vCPU 가 일시적으로 물리 코어에 고정되는 순간을 포착한 것으로, KVM 의 CFS 가 특정 조건에서 vCPU 마이그레이션을 중단함을 시사한다.

베어메탈 Xeon 환경은 평균 0.46%의 낮은 미스율을 유지했으나, 91-93 번째 측정에서 1.1-1.17%의 스파이크가 발생했다. 이는 OS 의 정상적인 인터럽트 처리나 컨텍스트 스위칭에 의한 것으로 판단된다.

Akamai 클라우드 환경의 데이터는 약 20.2% 근방(83 회)과 약 55.4% 근방(17 회)의 두 개 그룹으로 나뉘는 이중 분포(bimodal distribution)를 보였다.

6. 논의

제안 기법에 대한 잠재적 회피 방법은 다음과 같다.

첫째, 하이퍼바이저가 vCPU 를 물리 코어에 고정(CPU pinning)하는 방법이 있다. 실제로 일부 고성능 컴퓨팅 환경에서는 이러한 설정을 사용한다. 그러나 이는 전체 시스템의 유연성을 크게 제약한다. 특히 오버커밋 상황(vCPU 수 > 물리 CPU 수)에서는 불가능하며, 클라우드 환경의 핵심 가치인 자원 공유와 탄력성을 포기해야 한다.

둘째, 캐시 파티셔닝 기술을 사용할 수 있다. Intel CAT(Cache Allocation Technology)나 AMD 의 유사 기술을 통해 VM 별로 캐시 way 를 할당하면 간섭을 줄일 수 있다. 그러나 이 역시 캐시 활용도를 떨어뜨린다. 예를 들어, 16-way LLC 에서 각 VM 에 4-way 씩 할당하면 4 개 VM 만 지원 가능하고, 각 VM 은 전체 캐시의 25%만 사용할 수 있다. 또한 way 파티셔닝은 set 간섭을 막지 못하므로 완벽한 격리가 불가능하다.

셋째, 하이퍼바이저와 게스트 간 스케줄링 정보 공유(para-virtualized scheduling)가 근본적 해결책이 될 수 있다. Pillai 와 Fernandes 는 게스트와 호스트가 스케줄링 정보를 공유하는 BPF 기반 솔루션을 제안했다. [11] 그러나 이는 게스트 OS 와 하이퍼바이저 모두의 수정을 요구하며, 서로 다른 버전 간 호환성 문제가 있다.

기존 연구와 비교하면, D'Elia 등의 L3 Prime+Probe[5]는 eviction set 구성이 복잡하고 실행 시간이 수백초인 반면, 제안 기법은 십수초 내 탐지가 가능하다. Zhi 등의 통계 분석[10]은 분산, 첨도, 비대칭도의 복잡한 모델을 사용하나, 우리는 캐시 미스율이라는 단순하고 물리적 의미가 명확한 지표를 사용한다. Franklin 등[6]은 제어 레지스터 명령을 수십만 번 반복해야 하지만, 우리는 정상적인 메모리 접근만으로 탐지한다.

한계점으로는 x86_64 아키텍처에 한정된 실험, 단순 마이크로벤치마크 사용, 네스티드 가상화 미고려 등이 있다.

7. 결론

본 논문은 가상화 환경의 이중 스케줄링이 캐시 성능에 미치는 영향을 분석하여 가상화 탐지 기법을 제안했다. 실험 결과, 베어메탈 대비 VM 환경에서 75-189 배의 캐시 미스율 증가를 관찰했고, VM 환경의 표준편차는 15.84-26.80 으로 베어메탈(0.04-0.14)보다 현저히 높았다.

본 연구는 이중 스케줄링의 성능 영향을 정량화하여 가상화 기술의 한계를 드러냈다. 호스트-게스트 인터페이스의 한계로 완벽한 스케줄링 협력이 어려운 한, 제안된 캐시 성능 기반 탐지 기법은 당분간 유효할 것이다. 향후 다양한 하드웨어 아키텍처에서의 검증과 효과적인 방어 메커니즘 개발이 필요하다.

ACKNOWLEDGMENT

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 융합보안핵심인재양성사업의 연구 결과로 수행되었음 (IITP-2025-RS-2023-00266615)

8. 참고문헌

[1] Dario Faggioli, "Virtual-machine scheduling and scheduling in virtual machines," LWN.net, 2019. https://lwn.net/Articles/793375/

[2] T. Vateva-Gurova, N. Suri, A. Mendelson, "The Impact of Hypervisor Scheduling on Compromising Virtualized Environments," IEEE International Conference on Computer and Information Technology, Liverpool, UK, 2015, pp. 1910-1917.

[3] F. Liu, Y. Yarom, Q. Ge, G. Heiser, R. B. Lee, "Last-Level Cache Side-Channel Attacks are Practical," IEEE Symposium on Security and Privacy, San Jose, CA, USA, 2015, pp. 605-622.

[4] Yarom, Yuval, Katrina Falkner, "FLUSH+RELOAD: A high resolution, low noise, l3 cache Side-Channel attack," 23rd USENIX security symposium (USENIX security 14), San Diego, CA, USA, 2014, pp. 719-732.

[5] Daniele Cono D'Elia, "My Ticks Don't Lie: New Timing Attacks for Hypervisor Detection," BlackHat Europe, London, UK, 2020.

[6] Jason Franklin, Mark Luk, Jonathan M. McCune, Arvind Seshadri, Adrian Perrig, Leendert van Doorn, "Remote detection of virtual machine monitors with fuzzy benchmarking," SIGOPS Operating Systems Review, vol. 42, no. 3, pp. 83-92, 2008.

[7] D. GhatrehSamani, C. Denninnart, J. Bacik, and M. Amini Salehi, "The Art of CPU-Pinning: Evaluating and Improving the Performance of Virtualization and Containerization Platforms," in 2020 49th International Conference on Parallel Processing (ICPP), Edmonton, AB, Canada, 2020, pp. 1-11.

[8] X. Tao, L. Wang, Z. Xu, R. Xie, "Detection of Hardware-Assisted Virtualization Based on Low-Level Feature," IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Hangzhou, China, 2022, pp. 914-919.

[10] Lin, Z., Song, Y., Wang, J., "Detection of Virtual Machines Based on Thread Scheduling," International Conference on Artificial Intelligence and Security (ICAIS 2021), Dublin, Ireland, 2021, pp. 237-248.

[11] J. Fernandes, V. R. Pillai, "Paravirt scheduling with eBPF," Linux Plumbers Conference 2024, Vienna, Austria, 2024.