Understanding of Balanced Performance: Battle of CPU Resource Contention and Gang Sched

Authors: @Wenhui Zhang

Status: Draft

Last modified: 07-17-2022

Summary

Defenses against CPU Resource Contention can be categorized into software based defenses and hardware-based defenses, according to where the defense mechanism is implemented.

1. Software-based Defenses

Contention Detection. The first category of software defenses is to deploy defense systems to detect CPU Resource Contention and take countermeasures. Since the most important pre-requisite of CPU Resource Contention is to achieve co-residency, there are works focusing on actions after detecting co-residency in order to mitigate the CPU Resource Contention. For example, Zhang et al. (2011b) showed that a tenant can use CPU Resource Contention to confirm physical isolation of their VMs.

The performance counters (PMCs) are designed for monitoring specific micro-architectual events for performance, such as cache hit/miss and clock cycles. Since launching the CPU Resource Contention would trigger these events, PMCs can be used to detect CPU Resource Contention. Chiappetta et al. (2016) proposed mechanisms to detect the Flush+Reload attack based on the readings of PMCs. Zhang et al. (2016) introduced CloudRadar, which is a monitoring system that can detect cache-based CPU Resource Contention on the cloud. It used the PMCs to perform anomaly detection to identify cross-VM CPU Resource Contention.

Isolation. Another major category of defenses is to enforce isolation between the attacker VM and the victim VM. This includes isolating the entire VM using schedulers and migrations, or isolating the shared resources such as caches.

The first approach to achieve isolation is to isolate the VMs. This can be achieved by scheduling the VMs to reduce the risk of sharing. Liu et al. (2014b) introduced a **covert-channel aware scheduler**, which strictly limited the interleaving executions of different VMs to reduce the possibility of CPU Resource Contention. Varadarajan et al. (2014) proposed a way to achieve **soft isolation between VMs through hypervisor schedulers**. They showed that by enforcing a minimum run time (MRT) guarantee for the virtual CPUs of VM to **limit the frequency of preemptions**, Prime+Probe attacks can be mitigated. Moon et al. (2015) proposed Nomad, which was a system providing migration-as-a-service on the cloud provider side. It can coordinate VM placement and migration, so that information leakage of co-residency is limited.

Another approach to achieve isolation is to isolate the shared resources such as caches. **Cache partitioning and page coloring** have been proposed by researchers to mitigate cache side-channel attacks. The ideas are similar: reserve specific cache lines or pages, so that only trusted processes (e.g., processes of the same VM) can access them. The first work of applying cache partitioning to mitigate side-channel attacks was due to Page (2005), which dynamically split the cache memory into protected regions to achieve isolation. Kim et al. (2012) introduced StealthMem to lock stealth pages in the cache to store sensitive data. Shi et al. (2011b) used dynamic page coloring to protect security-critical operations to make sure that no other process shares the same color. Similarly, Godfrey and Zulkernine (2014) showed that coloring-based cache partition in Xen was effective in defeating CPU Resource Contention.

There are other techniques to achieve isolation on shared resources. Zhou et al. (2016) introduced CacheBar, a copy-on-access mechanism to manage physical pages shared across mutually distrusting parties. When different parties access the same physical page, each of them will have a local copy, i.e., there is no sharing, thus LLC side-channel attacks (CPU Resource Contention on LLC) are defeated. Zhang and Reiter (2013) proposed a cache cleansing mechanism to remove the signals contained in the cache, in order to achieve isolation. Similarly, Godfrey and Zulkernine (2014) demonstrated that flush caches of all levels during context switches in a hypervisor can effectively defeat cache contention.

2. Hardware-based Defenses

Specialized hardware features. Hardware transactional memory (HTM) allows threads to execute transactions in parallel; each thread works on a private snapshot to execute one transaction. Whenever there are conflicting memory accesses, the transaction will be aborted and the changes will be rolled back. Otherwise, the changes are committed atomically. The most widely used HTMs are cache-based HTMs; one of the recent commercial implementations is the Intel Transactional Synchronization eXtension (TSX). The transactions will abort when the accessed memory regions are not in the cache, which makes it a good tool for detecting cache-based side-channel attacks. Gruss et al. (2017) and Chen et al. (2018) have shown that Intel TSX can be used to effectively detect cache contention.

Intel Cache Allocation Technology (CAT) was introduced in 2016 to provide software control of where data is allocated into the last-level cache (LLC). It allows the OS or hypervisor to group applications into classes of service (CLOS), and specifies the amount of LLC available to each CLOS. Liu et al. (2016) utilized the CAT to partition the LLC into secure and non-secure partitions to build a pseudo-locking mechanism. They built a prototype system using Xen running Linux VMs on a cloud server and showed that it can mitigate LLC cache side-channel attacks. New cache design. Researchers also proposed new cache designs to lock cache lines or perform randomization in order to mitigate LLC contention.

New Cache Design. Wang and Lee (2007) proposed the PartitionLocked cache (PLcache), which added the owner information to cache lines to restrict cross-owner eviction. In the same work, they also proposed the Random Permutation Cache (RPCache) to randomize cache mappings. Wang and Lee (2008) further introduced NewCache, another novel cache architecture that adopted direct-mapping and security-aware cache replacement algorithm to achieve better security and performance. Recently Werner et al. (2019) presented ScatterCache, which enforced cache set randomization and made eviction-based cache attacks impractical.

3. Future Research

We foresee two research future directions in the space that will be of continued interest to the researchers.

First, new threat models in different cloud settings. As the cloud computing paradigm shifts from heavyweight virtualization towards lightweight containerization, from full-fledged cloud servers towards serverless computations, from compute-centric models to data-centric models, new threat models will emerge. Moreover, confidential cloud computing enabled by hardware assisted trusted execution environment (TEE) has also introduced new attack vectors for side channels (Wang et al., 2017; Chen et al., 2017; Chen et al., 2019a). As such, we would expect research studies to continue exploring new CPU resource contention vectors in the ever-changing cloud settings.

Second, the design and implementation of CPU-resource-contention-aware hardware and software systems. For instance, system software, such as operating systems and hypervisors, as well as new computer micro-architectures will be designed and implemented to provide stronger isolation for cloud tenants. We have already witnessed such trends in the confidential cloud computing cases, where changes in the Linux kernels to enable AMD SEV have been merged into the main stream Linux kernels and microcode updates have been released to address ciphertext side-channel attacks which are introduced by CPU resource contention(AMD, 2021).

Mitigating Cross-VM Side Channel Attack on Multiple Tenants Cloud Platform

This paper proposes a **covert channel aware scheduler**. "The scheduler is able to control the execution time overlapping of different VMs, and can also inject noise periodically to mitigate the threat of potential side channels."

This scheduler is based on Xen and its **credit scheduler**.

The new scheduler introduces two parameters: **overlap_cap**, which is the ***threshold of overlap time of execution between every two VMs***; and injected noise_function for different types of side channel.

This paper proposed two policies:

1. Policy-1: Minimize overlapping of execution time of different VMs.
2. Policy-2: Zero-overlapping for specific VMs.

For example, assume that a host has 16 CPU cores, and a protected VM has 16 VCPUs (Virtual CPU). Once the VM is running, all of its VCPUs are scheduled to run and all VCPUs from other VMs are waiting, until all of the VCPUs of the protected VM are scheduled out.

Xen Sched: There are four running states of each VCPU: **BOOST, UNDER, OVER and IDLE**. For each VCPU that is scheduled to run, the time slide is about 30ms.
In **Credit scheduler**, credits are used to track each VCPU’ s execution time. As long as a VCPU has not burned all of its credit, it is in UNDER state. The speed of credit consumption is 100 per 10ms (**10ms is a tick**). Once a VCPU’ s credit has dropped under 0, the priority is set to OVER. **Every 30ms, the scheduler will recharge credit of VCPUs waiting in the queue.** The credit-weight of each VCPU determines the quantity of its charged credit.

Author adds a new parameter to the credit scheduler in Xen: **overlap_cap**, as a threshold in terms of millisecond. The scheduler will ensure that for every two VCPUs, the runtime overlapping will not exceed the threshold.

Also, when a VCPU is in BOOST status, it has the highest priority and will preempt to run even if the VCPUs on other CPUs are from other VMs. When selecting a next VCPU to run, the scheduler prefers
the one from the same VM in order to get longer time slice and to avoid too frequent context switches.

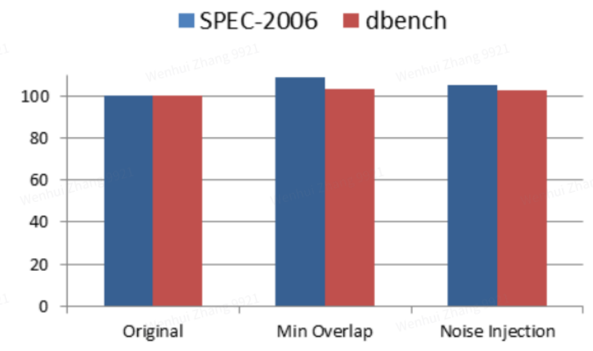


Figure 4. Performance overhead of SPEC-2006, each VM has 4 VCPUs running concurrently.

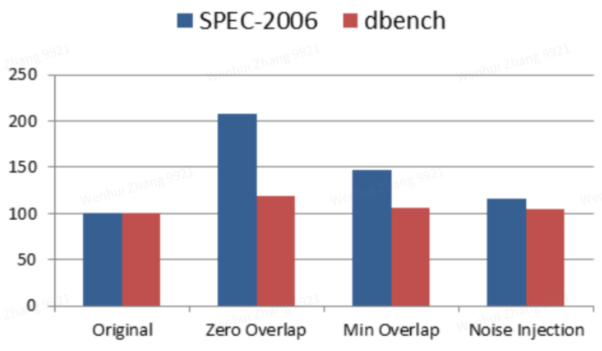


Figure 5. Performance overhead of SPEC-2006, each VM has only one VCPU running.

Scheduler-based Defenses against Cross-VM Side-channels

This paper proposes soft isolation: reduce the risk of sharing through better scheduling. This paper modifies the Xen scheduler to limit the frequency in which an attacker can preempt the victim. With experimental measurements, we show that a **minimum run time (MRT)** guarantee for VM virtual CPUs that **limits the frequency of preemptions** can effectively prevent existing **Prime+Probe** cache-based side-channel attacks.

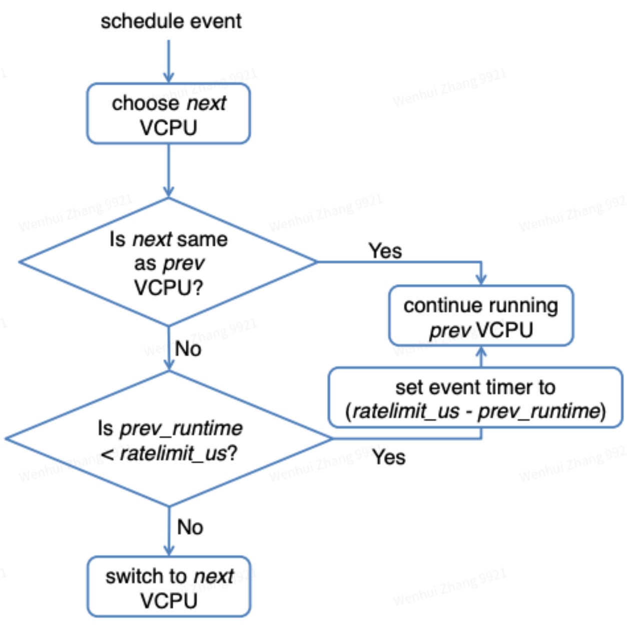
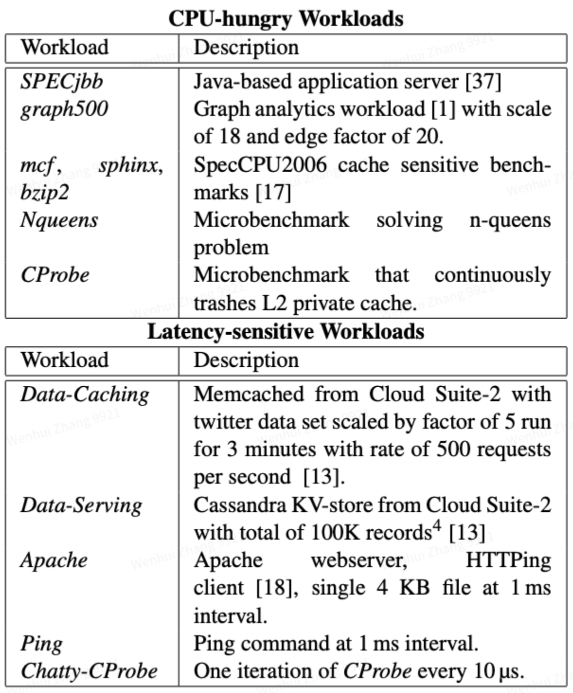

schedule event
choose *next* VCPU
Is *next* same as *prev* VCPU?
Yes
continue running *prev* VCPU
No
set event timer to (*ratelimit_us* - *prev_runtime*)
Is *prev_runtime* < *ratelimit_us*?
Yes
No
switch to *next* VCPU

Figure 1: **Logic underlying the Xen MRT mechanism.**

Xen exposes a hypervisor parameter, **ratelimit_us** (the MRT value) that determines the minimum time any VCPU is guaranteed to run on a PCPU before being available to be context-switched out of the PCPU by another VCPU. The original intent of Xen' s MRT was to improve performance for **CPU-hungry workloads run in the presence of latency-sensitive workloads**: each preemption pollutes the cache and other microarchitectural state, slowing the CPU-intensive workload.



CPU-hungry Workloads

Workload	Description
SPECjbb	Java-based application server [37]
graph500	Graph analytics workload [1] with scale of 18 and edge factor of 20.
mcf, *sphinx*, *bzip2*	SpecCPU2006 cache sensitive benchmarks [17]
Nqueens	Microbenchmark solving n-queens problem
CProbe	Microbenchmark that continuously trashes L2 private cache.

Latency-sensitive Workloads

Workload	Description
Data-Caching	Memcached from Cloud Suite-2 with twitter data set scaled by factor of 5 run for 3 minutes with rate of 500 requests per second [13].
Data-Serving	Cassandra KV-store from Cloud Suite-2 with total of 100K records[4] [13]
Apache	Apache webserver, HTTPing client [18], single 4 KB file at 1 ms interval.
Ping	Ping command at 1 ms interval.
Chatty-CProbe	One iteration of *CProbe* every 10 µs.

Figure 10: **Workloads used in performance experiments.**

🥇 Linux CFS Bandwidth Control is similar to ratelimit_us,

CFS bandwidth allowed for a group is specified using a quota and period (microseconds).

"A group’ s unassigned quota is globally tracked, being refreshed back to cfs_quota units at each period boundary. As threads consume this bandwidth it is transferred to cpu-local “silos” on a demand basis. The amount transferred within each of these updates is tunable and described as the “slice” ."

Nomad: Mitigating Arbitrary Cloud Side Channels via Provider-Assisted Migration

(1) A formal model to capture information leakage via side channels in shared cloud deployments;

(2) identifying provider-assisted VM migration as a robust defense for arbitrary side channels; (3) a scalable online VM migration heuristic that can handle large datacenter workloads; and
(4) a practical implementation in OpenStack.

Migration with stop of preemptive behavior