Intel® Workload Interference Detector

Version 1.2.0

User Guide

June 2023

Quick Start

Use the Intel® Workload Interference Detector (AKA procmon) to monitor the performance of processes or containers and detect interference between workloads. Procmon leverages the Intel PMU to provide light weight and real-time performance tracking.

Download

Clone from public Github repo: https://github.com/intel/interferencedetector

Setup

Run the following commands to install required dependencies:

```
cd procmon
python3 -m pip install -r requirements.txt
```

Collect metrics per PID

```
sudo python3 procmon.py
---------------------------------------------
Timestamp,PID,process,cgroupID,core,cycles,insts,cpi,,...
1686005439.574799,1301,kworker/u193:10,0,11,2000000,2000000,1.00,...
1686005439.575033,1460,irqbalance,0,52,8000000,15000000,0.53,...
1686005439.575084,2404,python3,0,1,11000000,7000000,1.57,...
---------------------------------------------
Timestamp,PID,process,cgroupID,core,cycles,insts,cpi,...
1686005440.590906,2404,python3,0,1,6000000,7000000,0...
---------------------------------------------
Timestamp,PID,process,cgroupID,core,cycles,insts,cpi,...
1686005441.605687,2404,python3,0,1,7000000,7000000,1.00,...
```

Collect metrics per container

```
sudo python3 dockermon.py
---------------------------------------------
Timestamp,containerID,PID,process,cgroupID,core,cycles,insts,cpi,…
1686008486.308105,01b5bb37a0d3,4828,mlc,0,4,3217000000,2821000000,1.14,...
----------------------------------------------------------------------
Timestamp,containerID,PID,process,cgroupID,core,cycles,insts,cpi,...
1686008487.395614,01b5bb37a0d3,4828,mlc,0,4,3171000000,2784000000,...
-----------------------------------------------------------------------
Timestamp,containerID,PID,process,cgroupID,core,cycles,insts,cpi,,...
1686008488.501863,01b5bb37a0d3,4828,mlc,0,4,3137000000,2832000000,,...
```

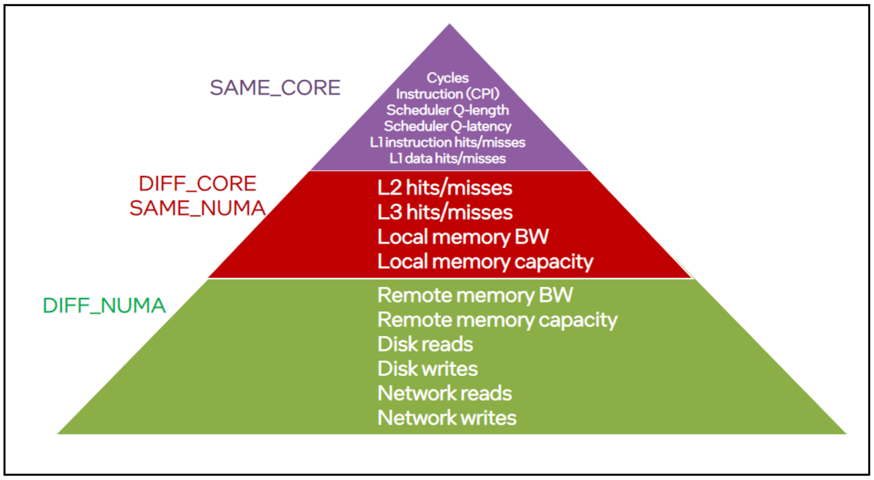
Monitor performance of running containers

```
sudo python3 dockermon.py --collect_signatures -d 10
sudo python3 NN_detect.py -s dockermon_*/signatures_mean.csv
---------------------------------------------
________
 ContainerID:  XXXXXXXXXX
At time: 06/06/2023, 20:20:52 detected signature on core 5
Distance from reference: 6.0%    ==> Performance is OK
```

Overview

The Intel® Workload Interference Detector (AKA procmon) is a tool that leverages the Intel PMU to monitor and detect interference between workloads. Traditional PMU drivers that work in counting mode (i.e. emon, perf-stat) provide system level analysis with very little overhead. However, these drivers lack the ability to breakdown the system level metrics (CPI, cache misses, etc) at a process or application level. With ebpf, it is possible to associate the process context with the HW counter data, providing the ability to breakdown PMU metrics by process at a system level. Additionally, since ebpf runs filters in the kernel and uses perf in counting mode, this incurs very little overhead, allowing for real-time performance tracking as well as fast detection of noisy neighbors.

First, prcomon collects a set of metrics through Intel PMU. The PMU is configured in *Counting* mode, in which the events are measured for the application as a whole. This has the advantage of being light-weight compared to *Sampling* mode. Sampling mode is able to collect performance events for specific code regions within the application, and hence is very useful in detecting hotspots. However, here the focus is on interference between workloads, and therefore "Counting" mode is both suffiecient and also desirable to reduce the noise that can originate from procmon itself.



The collected event are divided into 3 categories, based on which level of deployment a pair of workloads can have. For example:

1) **SAME_CORE**: When two workloads run on the same physical core (or hyper thread), they can interfere in Cycles, Instructions, Run-Queue, and L1 or L2 caches (assuming each core has a separate L1 and L2 caches).

2) **DIFF_CORE, SAME_NUMA**: Here the two workloads run on different physical cores, but both cores are in the same NUMA zone. In this case, the two workloads won't be contending for CPU Cycles, Instructions, L1, or L2 caches (since each as a separate physical core). However, they can compete for shared resources within the same NUMA zone as L3 cache and Local Memory BW.

3) DIFF_NUMA: Here the two workloads run in different cores and in different NUMA zones. Therefore, contention cannot be on the Core level, nor the NUMA level. However, they can still contend for shared resources on the system-level resources such as Remote memory BW, Disk Reads/Writes, and Network BW.

Notice that each level is a super-set of the following levels. For example, two workloads running in the same Core can be contending for system-level resources such as network BW. Accordingly, for workloads that execute in level 1 (SAME_CORE), we consider possible contention in all levels. For workloads that execute in level 2 (DIFF_CORE, SAME_NUMA), we consider possible contention in level 2 and level 3 metrics only.

Afterwards, a second script "dockermon.py" aggregates the events collected by procmon.py on a container level. This is achieved by running two parallel threads: (1) A thread the runs procmon and collects the events per process id. (2) A thread the uses Linux ps command to get the mapping between process ids and container ids. The two threads share a dictionary that represents this mapping. Every second, the dictionary is updated and then used by procmon's thread in the following second. Therefore, for newly executed containers, the aggregated events might be delayed by a single second while the mapping data is captured, and the shared dictionary is updated. Dockermon has the two exporting features (in addition to console printing):

(1) Collecting signatures *(--collect_signatures* flag): which stores signatures (event vectors) in pandas CSV files.

(2) Export to AWS cloudwatch *(--export_to_cloudwatch* flag): which sends the collected data per container to AWS cloudwatch. To use this feature, AWS cli must to be configured with the appropriate Access key ID, Secret Access Key, and Region as shown in this link. The following figure shows an example of cloudwatch dashboard showing 2 raw metrics collected by dockermon (cpi and l3_miss_ratio) on top, and an alarm configured w.r.t l3_miss_ratio at the bottom. One can setup multiple alarms in cloudwatch and associate specific actions to be executed when a alarm is triggered (e.g., migrate a workload to a different host).

Figure 1

Third, NN_detect.py can be executed to monitor the performance of running containers, and flag performance degradation incidents. It can also identify which metric is being impacted the most (e.g., L3 cache hits), and use the most impacted metrics to identify the interfering workloads. The following figure shows an example of NN_detect's output showing two scenarios: (1) Performance is ok. (2) Performance is degraded.

Performance is OK (No noise detected):

```
-----------------------------------------------------------------
________
 ContainerID:  ce945eac58ee
At time: 06/06/2023, 21:41:44 detected signature on core 5
Distance from reference: 4.0%    ==> Performance is OK
```

Performance is degraded (Noise detected from another container):

```
-----------------------------------------------------------------
________
 ContainerID:  ce945eac58ee
At time: 06/06/2023, 21:43:25 detected signature on core 5
Distance from reference: 57.16%  ==> Performance may suffer. Impacted metrics: cycles,insts,l3_miss_ratio,local_bw
[Same-Core/Thread] Noisy Neighbor #1 on core #5:        0a8c9acee68b
```

Running the Intel® Workload Interference Detector

Intel® Workload Interference Detector has been tested with the CPU architectures shown in the table below. It may run with limited functionality on other architectures. The tar utility must be installed in order to procmon<version>.tgz. Also BCC needs to be compiled from source: link.

Architecture	Operating System
Supported Servers	
SPR, ICX, CLX, SKX	Ubuntu 14.04 or newer
Supported Cloud Servers	
AWS Metal instances (e.g., r5.metal, m6i.metal, etc.)	Ubuntu14.04 or newer
AWS Single Socket instances (e.g., c5.12xlarge, c6i.16xlarge)	Ubuntu14.04 or newer

The first step to install procmon is to compile and install BCC. BCC is a toolkit that makes BPF programs easier to write, with kernel instrumentation in C (and includes a C wrapper around LLVM), and front-end in Python. To compile BCC from the source, follow OS and distribution specific instructions given here: Install BCC from source.

Next, follow these commands to unpack procmon<version>.tgz. Navigate to the procmon folder, install dependencies, and verify the correct installation of procmon:

```
$ tar zxvf procmon.v1.0.1.tgz
$ cd procmon
$ sudo pip3 install -r requirements.txt
$ sudo python3 procmon.py
```

If BCC is not installed properly, the following messages will be printed

```
$ sudo python3 procmon.py
 BCC modules (BPF, Perf, & PerfType) are not installed. Did you compile and build
BCC from the source? https://github.com/iovisor/bcc/blob/master/INSTALL.md#source
```

If all dependencies are installed correctly, procmon will start printing the event metrics every second as follows:

```
$ sudo python3 procmon.py
Architecture:  ICELAKE has OCR support!
Timestamp,PID,process,cgroupID,core,cycles,insts,cpi,...
1686091850.390772,36320,python3,0,69,158000000,98000000,1.61,...
-----------------------------------------------------------------------
Timestamp,PID,process,cgroupID,core,cycles,insts,cpi,...
1686091851.406713,36320,python3,0,69,23000000,15000000,1.53,...
```

For each running process, a separate row is printed. If the same process runs on more than one core, a separate row will be printed for each process-core.

The Intel® Workload Interference Detector supports arguments for frequency and duration of gathering data. Add the arguments explained below to the basic procmon command:

procmon.py [-h] [-v]

[-f SAMPLE_FREQ]

[-d DURATION]

[-i INTERVAL]

[--aggregate_cpus]

[--aggregate_cgroup]

[--acc]

The following sections explain each parameter and offer usage examples.

Procmon Arguments

Argument	Description

-h	Intel® Workload Interference Detector scripts come with a built-in help files that you can access at any time from the command line:
	`$ python3 procmon.py -h`
	`$ python3 dockermon.py -h`
	`$ python3 NN_detect.py -h`

-v, --verbose	Increase verbosity of procmon. For example, show raw counters in every interval
	`$ python3 procmon.py -v`
	`SCALE: 3.82`
	`CYCLES: 128 {(42053, b'python3', 0, 0): 0, (42053, b'python3', 1, 0): 0, (42053, b'python3', 2, 0): 0, ...`

-f SAMPLE_FREQ	Frequency of copying counters from the PMU to user space, with default value of 10M. Accordingly, events that occur less than SAMPLE_FREQ/sec will be zeroed out.
-d DURATION	Specify the duration of procmon’s runtime. If not set, procmon runs indefinitely.
-i INTERVAL	Interval of measurement. Default is 1 sec causing each metric to be reported once every second.

Argument	Description
--aggregate_cpus	`Aggregate metrics across all CPUs. If the process runs multiple threads on the CPU, all metrics across these threads will be aggregated together`
--aggregate_cgroup	`Aggregate metrics across all PIDs within the same cgroup.`
--acc	`Run in accumulative mode. This will not reset the counters each INTERVAL. Accordingly, the metrics will be summed over time from the start of procmon`

Dockermon Arguments

Argument	Description

-v	`show raw verbose logging info. For example, it will print how long it took to map process ids to container ids.`
	`Sudo ps and grep Latency:  0.03  seconds`
	`Total API Calls Latency:  0.03  seconds`

--collect_signatures	`collect signatures of running containers and dump to signatures_*.csv.`
	`Collected signatures are in the following format:`
	`containerID_PID,TimeStamp_Count,cycles,insts,cpi,...`
	`3ce488b7aaf3_22918,20.0,1630000000.0,732000000.0,2.23,...`
	`44c0a1d029cb_22911,20.0,1630500000.0,5536500000.0,0.29,...`

-d DURATION	`Duration of collecting signatures. For example, the following command collects signatures for 20 seconds:` `sudo python3 dockermon.py --collect_signatures -d 20`
--aggregate_on_core	`Show signatures aggregated on the core level (i.e., one signature per core)`
--aggregate_on_containerID	`Show signatures aggregated on Container ID (i.e., one signature per container ID)`
--export_to_cloudwatch	`Send collected data to AWS cloudwatch. Expects the following AWS parameters to be configured in 'aws cli': aws_access_key_id, aws_secret_access_key, and aws_region.`

Argument	Description
--cloudwatch_sampling_duration_in_sec	Duration between samples of data points sent to cloudwatch. Default is 10 (one sample every 10 seconds). The minimum duration is 1 second. Note: this argument is only effective when --export_to_cloudwatch is set.

NN_detect Arguments

Argument	Description
-v/--verbose	Increase verbosity. For example, will print the detected signatures of the workload of interest and of noisy neighbors (if any).
-p/--pid PID	Process level Noisy Neighbor detection. NN_detect will use the provided process pid as the "workload of interest" and show its performance status ('OK', or 'May suffer'). If performance status is `May suffer`, a list of noisy neighbor Process IDs will be printed, with most noisy neighbor on top.
-c/--cid CID	Same as -p/pid but on container level.
--outfile OUTFILE	File to save the Noisy Neighbor detection output
-s SYSTEM_WIDE_SIGNATURES_PATH	Path to saved signatures generated by dockermon with --collect_signatures flag. See "Usage Demo" section for examples.
-r REF_SIGNATURE	Reference signature for a single workload (either a process or a container). This flag is mutually exclusive with `-s` flag
-t THRESHOLD	Threshold of acceptable distance from reference (default is 15% from reference). If the distance is higher than this threshold, the monitored workload will be flagged as a noisy neighbor victim.

Usage Demo

Now we give steps to run Intel® Workload Interference Detector and detect noisy neighbors in a AWS EC2 host.

- **(A) Start an AWS EC2 instance of type** [m5.metal] **with Ubuntu 22.04.1 LTS operating system**
- **(B) Compile BCC from source as shown here:** [BCC Installation from Source]
- **(C) Confirm correct installation by running sudo procmon.py**
  If BCC is not installed correctly, the following message will show up:
  "BCC modules (BPF, Perf, & PerfType) are not installed. Did you compile and build BCC from the source? https://github.com/iovisor/bcc/blob/master/INSTALL.md#source"
- **(D) Now run a stress-ng workload in a container on core 1 for 10 minutes:**
  - **a. docker run --cpuset-cpus 1 --rm --detach colinianking/stress-ng --cpu 0 --cpu-load 80 --timeout 600s**
- **(E) Run dockermon to collect the signature for the stress-ng container**
  - **a. python3 dockermon.py --collect_signatures -d 20**
- **(F) Run NN_detect to capture the performance status of the running container:**
  - **a. python3 NN_detect.py -s dockermon_<<>>/signatures_mean.csv**
- **(G) NN_dectect shows the performance status is OK as shown in Figure 3.**

Figure 3

```
------------------------------------------------------------------
 ContainerID:  96e5bd6bf2ea
At time: 06/09/2023, 19:01:46 detected signature on core 1
Distance from reference: 0.75%   ==> Performance is OK
```

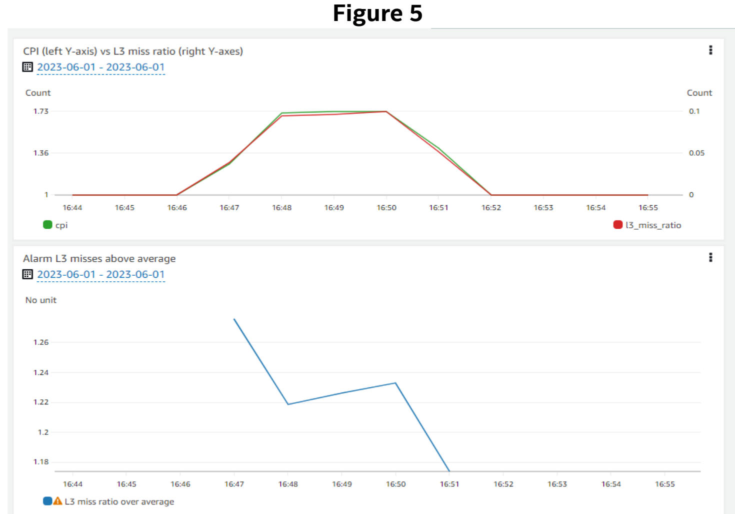
- **(H) Now we run a noisy neighbor on core 1 for 20 seconds:**
  - **a. docker run --cpuset-cpus 1 --rm --detach colinianking/stress-ng --cpu 0 --cpu-load 80 --timeout 20s**
- **(I) The performance is degraded as shown in Figure 4**
- **(J) After 20 seconds, the noisy container terminates, and performance status is back to OK.**

Figure 4

```
------------------------------------------------------------------
 ContainerID:  96e5bd6bf2ea
At time: 06/09/2023, 19:00:34 detected signature on core 1
Distance from reference: 39.41%  ==> Performance may suffer. Impacted metrics: cycles,insts
[Same-Core/Thread] Noisy Neighbor #1 on core #1:        6213e95c87da
```

- **(K) Finally, one can run dockermon with --**export_to_cloudwatch **flag and visualize collected metrics per container ID.**
  - **a. It is also possible to create an alarm when a metric increases or decreases by a threshold.**
  - **b. For example, we set create an alarm on L3 misses when it increases by 10% above average and see the following dashboard in Cloudwatch as shown in Figure 5.**

Figure 5



Feedback

We value your feedback. If you have comments (positive or negative) on this guide or are seeking something that is not part of this guide, please reach out and let us know what you think.

If you have information about a security issue or vulnerability with Intel® Workload Interference Detector, please send an e-mail to secure@intel.com. Encrypt sensitive information using our PGP public key.