

The MSRs of EC2

15 Sep 2014

Is Intel Turbo Boost running for my AWS EC2 cloud instance (which is a Xen guest)?

```
ec2-guest# ./showboost
CPU MHz      : 2500
Turbo MHz    : 2900 (10 active)
Turbo Ratio  : 116% (10 active)
CPU 0 summary every 5 seconds...

TIME          C0_MCYC          C0_ACYC          UTIL  RATIO    MHz
06:11:35      6428553166        7457384521        51%   116%   2900
06:11:40      6349881107        7365764152        50%   115%   2899
06:11:45      6240610655        7239046277        49%   115%   2899
06:11:50      6225704733        7221962116        49%   116%   2900
[...]
```

Yes! These 2500 MHz CPUs are currently running at 2900 MHz.

I guess the CPUs are cold enough to boost. What's their temperature?

```
ec2-guest# ./cputemp 1
CPU1 CPU2 CPU3 CPU4 CPU5 CPU6 CPU7 CPU8 CPU9 CPU10 CPU11 CPU12 CPU13 CPU14 CPU15 CPU16
70 68 68 65 63 63 61 60 68 64 64 63 62 61 70 68
70 68 69 65 63 63 61 61 68 65 63 63 61 61 70 69
70 69 69 65 63 63 61 60 69 65 64 63 61 61 69 69
69 69 69 66 64 64 61 61 68 65 64 64 61 61 70 69
[...]
```

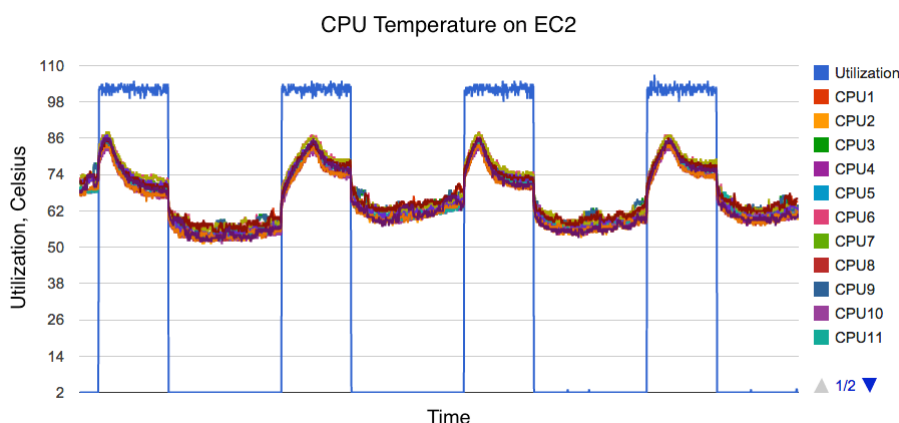
Relatively cool: between 60 and 70 degrees Celsius. This is another tool from my [msr-cloud-tools](#).

In this post I'll describe MSRs, how to read them, and why measuring turbo boost is important.

Model Specific Registers (MSRs)

Aka *machine specific* registers, these are described in [Vol 3c](#) of the Intel 64 and IA-32 Architectures Software Developer's Manual. They access low level CPU information, including turbo boost ratios and temperature readings. They are read and written using the RDMSR and WRMSR instructions.

The image on the right shows how CPU temperatures, measured using MSRs on an EC2 instance, vary based on CPU utilization (in blue). The workload is synthetic: all CPUs driven to 100% utilization for 5 minutes, then to 0% for a while, repeat. What's interesting is that temperature rises initially with CPU load, then drops sharply. Did system fans kick in? (So far I haven't found fan RPM MSRs, to confirm.)



I'm usually focused on the performance monitoring counters (PMCs; aka performance instrumentation counters (PICs), CPU performance counters (CPCs), etc.). These are read by RDPMC, and described in [Vol 3b](#) of the same manual. These can measure data cache misses, stall cycles, and other useful performance events.

On AWS EC2, within a cloud instance (Xen guest), PMCs aren't available, as you'll see with, say, "[perf stat](#)". That doesn't mean they can't ever work, just that they (or their controlling MSRs) aren't currently available.

But a small handful of MSRs are available on EC2. Here are the more interesting ones I've found:

Reg	Name	Description
0xe7	IA32_MPERF	Bits 63:0 is TSC Frequency Clock Counter C0_MCNT TSC relative
0xe8	IA32_APERF	Bits 63:0 is TSC Frequency Clock Counter C0_ACNT actual clocks
0x19c	IA32_THERM_STATUS	Bits 22:16 is the CPU therm status digital readout (DO)
0x1a2	MSR_TEMPERATURE_TARGET	Bits 23:16 is temp target (TT)
0x1ad	MSR_TURBO_RATIO_LIMIT	Bits 7:0 is the turbo boost ratio (x100 for MHz) for 1 core active
0x1ae	MSR_TURBO_RATIO_LIMIT1	Bits 15:8 (for example) is the turbo boost ratio for 10 cores active

Table 1. MSRs for Intel(R) Xeon(R) CPU E5-2670 v2

These are used by various kernel routines, like the idle thread and cpufreq.

Note that these are **model specific**, which means they can vary between different processor models (micro-architectures). For example, Silvermont has a read/write target offset in MSR_TEMPERATURE_TARGET (bits 29:24), which can lower the throttle temperature (PROCHOT). Such differences make MSRs non-portable and tricky to use, which is why standards like [PAPI](#) are important.

Reading MSRs

Here's how you can measure MSRs (assuming Intel):

1. Determine your CPU type and micro-architecture

```
# head /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 62
model name    : Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz
[...]
```

The family and model numbers tell us that this is the Ivy Bridge micro-architecture (see the [Intel decoder](#)). You can also use the `cpuid` tool (from the `cpuid` package), which should report micro-architecture directly.

2. Look up MSRs for your processor type

These are in Vol 3c of the [Intel software developer manual](#). This is a 540 page volume, and if you're new to it, you'll get lost a few times before you get the hang of it.

3. Install and load msr-tools:

```
# apt-get install msr-tools
# modprobe msr
```

(Assuming Ubuntu.) The `msr-tools` package adds the `rdmsr` and `wrmsr` tools, and the `msr` kernel module.

4. Use `rdmsr`

Based on the addresses from (2). Eg, to read the turbo boost ratio when 10 cores are active (Ivy Bridge):

```
# rdmsr 0x1ae -f 15:8 -d
29
```

Multiply by 100 to get MHz. The way these work is explained earlier in the manual.

I did share a couple of tools in a [msr-cloud-tools](#) collection, however, I've only written them for the processor type I'm currently analyzing. You may need to edit them to use the right MSRs.

Why Measure Turbo Boost

We live in an annoying age for computer performance analysts: the error margin for many measurements is over 10%, thanks to turbo boost, the Intel processor technology that can dynamically over-clock CPUs. Ubuntu is 10% faster than CentOS? Could just be turbo boost. New software version regressed by 5%? Could just be turbo boost. Tunable made things 10% faster? Could just be ... You get the picture.

Turbo boost can make a 2500 MHz processor run at 3300 Mhz, depending on factors including temperature, power consumption, and the C-state of the cores. Colder servers run faster. I once had two identical servers at the top and bottom of a rack, and the top server ran 5% faster, as it received more cold air from the air conditioners. That's both great and maddening: I'll take the better performance, but it can also mess up measurements when I'm comparing systems or software.

There are three ways I've historically dealt with this:

1. Turn off turbo boost in the BIOS when doing performance comparisons.
2. Measure actual CPU cycles using CPU performance counters to observe the turbo boost rate.
3. Run a short experiment (benchmark) to measure the current cycle rate, eg, [noploop](#).

If you run your own datacenter, you can do them all. But as a Xen guest on AWS EC2, you can't change the BIOS and do (1). You can do option (3), but that can be time-consuming and difficult (less reliable) on very busy systems. Until recently, I thought you couldn't do (2), either, and then I found the MSRs...

Discovering MSRs

I was working on a suspected turbo boost issue when a colleague at Netflix, Scott, mentioned that he liked using the `i7z` command to debug turbo boost. We didn't think it would work on EC2, but I tried anyway.

Most of the output was clearly wrong, but a lone column of temperature readings showed that something was working. Using opensnoop from my [ftrace perf-tools](#) collection to see how:

```
# ./opensnoop -n i7z
Tracing open()s issued by process name "i7z". Ctrl-C to end.
COMM      PID      FD  FILE
i7z        8427     0x3 /proc/cpuinfo
i7z        8427     0x3 /dev/cpu/0/msr
i7z        8427     0x3 /dev/cpu/0/msr
i7z        8427     0x3 /dev/cpu/0/msr
i7z        8427     0x3 /dev/cpu/0/msr
[...]
```

This showed that `i7z` was reading `/dev/cpu/0/msr`, and led me to take a close look at the available MSRs.

I'd normally use `CPU_CLK_Unhalted.Core`, but that wasn't available. After some digging, I found I could use the ratio of `IA32_APERF` deltas to `IA32_MPERF` deltas, which shows how much faster the time stamp counter (TSC, which is cycle-based) is moving when the processor is in the C0 state.

It was an enormous relief to find a way to directly measure real clock rates, and turbo boost. My error margins have vanished: **I can measure performance again.**

Conclusion

A handful of model-specific registers, MSRs, are available in Xen guests including on AWS EC2. These allow the real clock rate, and the degree of turbo boost, to be measured. This is important to know for any performance comparison, as variations in turbo boost can skew results by over 10%, based on how hot or cold servers were during the test.

I've written a couple of tools so far, in [msr-cloud-tools](#), to measure CPU turbo boost and temperature. As is the nature of MSRs, they are specific to processor types, and these scripts only work (so far) on our Intel(R) Xeon(R) CPU E5-2670 v2s. If you want to use these tools or MSRs yourself, you may need to find out the right MSRs to use for your processor type. The good news is that the vendor documentation from Intel and AMD is very good, although it takes some time to dig through.

4 Comments

Brendan Gregg's Blog

1 Login ▾

♥ Recommend

🐦 Tweet

f Share

Sort by Best ▾

**Zongheng Yang** • 4 years ago

Thanks for the post. Although the cpuid snippet clearly shows that it's an Ivy Bridge, the model number (62 = 0x3E) is not shown in the Intel Decoder page. Additionally, could you share your conclusion to the utilization ~ temperature graph showing CPUs suddenly cooling off?

2 ^ | ▾ • Share ›

**brendangregg** Mod ➔ Zongheng Yang • 4 years ago

G'Day,

Yes, sorry, that Intel decoder link was handy, but a bit out of date, and missed model 0x3e. The up to date lists are in the Intel Architectures Software Developer Manual. Currently Vol. 3C page 35-1 has the updated list of DisplayFamily and DisplayModel.

My conclusion so far is that the system fans are kicking in, cooling off the CPUs. I did check various MSRs that should register CPU thermal throttling, but they were always zero. Including these (from the Intel manual, Vol. 3B, 14-26):

"• Thermal Status (bit 0, RO) — This bit indicates whether the digital thermal sensor high-temperature output signal (PROCHOT#) is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.

• Thermal Status Log (bit 1, R/WC0) — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#). Bit 1 = 1 if PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero."

So, unless EC2 is masking these bits (other bits in the same MSR can be read), then it doesn't look like thermal throttling.

^ | ▾ • Share ›

**Nitsan Wakart** • 4 years ago

Nice post :)

An alternative to turning off the turbo boost in the BIOS is to fix the cpu frequency using a 'governor' which allows it: <https://wiki.debian.org/How...>

^ | ▾ • Share ›

**Syam** • 4 years ago

i7z is a pretty nifty tool to help measure some of these - <https://code.google.com/p/i7z/>

^ | ▾ • Share ›

[✉ Subscribe](#) [D Add Disqus to your site](#) [Add Disqus](#) [Add](#) [Disqus' Privacy Policy](#) [Privacy Policy](#) [Privacy Policy](#) [Privacy Policy](#)

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).

[Homepage](#)

[Blog](#)

[Full Site Map](#)

[Sys Perf book](#)

[Linux Perf](#)

[Perf Methods](#)

[USE Method](#)

[TSA Method](#)

[Off-CPU Analysis](#)

[Active Bench.](#)

[Flame Graphs](#)

[Heat Maps](#)

[Frequency Trails](#)

[Colony Graphs](#)

[perf Examples](#)

[eBPF Tools](#)

[DTrace Tools](#)

[DTraceToolkit](#)

[DtkshDemos](#)

[Guessing Game](#)

[Specials](#)

[Books](#)

[Other Sites](#)