July 31 ○ Andre

Understanding Linux CPU Load - when should you be worried?

Posted in Examples | 16 comments

You might be familiar with Linux load averages already. Load averages are the three numbers shown with the `uptime` and `top` commands - they look like this:

load average: 0.09, 0.05, 0.01

Most people have an inkling of what the load averages mean: the three numbers represent averages over progressively longer periods of time (one, five, and fifteen minute averages), and that lower numbers are better. Higher numbers represent a problem or an overloaded machine. But, what's the the threshold? What constitutes "good" and "bad" load average values? When should you be concerned over a load average value, and when should you scramble to fix it ASAP?

First, a little background on what the load average values mean. We'll start out with the simplest case: a machine with one single-core processor.

The traffic analogy

A single-core CPU is like a single lane of traffic. Imagine you are a bridge operator ... sometimes your bridge is so busy there are cars lined up to cross. You want to let folks know how traffic is moving on your bridge. A decent metric would be *how many cars are waiting* at a particular time. If no cars are waiting, incoming drivers know they can drive across right away. If cars are backed up, drivers know they're in for delays.

So, Bridge Operator, what numbering system are you going to use? How about:

- **0.00 means there's no traffic on the bridge at all**. In fact, between 0.00 and 1.00 means there's no backup, and an arriving car will just go right on.
- **1.00 means the bridge is *exactly* at capacity.** All is still good, but if traffic gets a little heavier, things are going to slow down.
- **over 1.00 means there's backup.** How much? Well, 2.00 means that there are two lanes worth of cars total -- one lane's worth on the bridge, and one lane's worth waiting. 3.00 means there are three lane's worth total -- one lane's worth on the bridge, and two lanes' worth waiting. Etc.

 = load of 1.00

 = load of 0.50

 = load of 1.70

This is basically what CPU load is. "Cars" are processes using a slice of CPU time ("crossing the bridge") or queued up to use the CPU. Unix refers to this as the *run-queue length*: the sum of the number of processes that are currently running plus the number that are waiting (queued) to run.

Like the bridge operator, you'd like your cars/processes to never be waiting. So, your CPU load should ideally stay below 1.00. Also like the bridge operator, you are still ok if you get some temporary spikes above 1.00 ... but when you're consistently above 1.00, you need to worry.

So you're saying the ideal load is 1.00?

Well, not exactly. The problem with a load of 1.00 is that you have no headroom. In practice, many sysadmins will draw a line at 0.70:

- The **"Need to Look into it"** Rule of Thumb: **0.70** If your load average is staying above > 0.70, it's time to investigate before things get worse.
- The **"Fix this now"** Rule of Thumb: **1.00**. If your load average stays above 1.00, find the problem and fix it now. Otherwise, you're going to get woken up in the middle of the night, and it's not going to be fun.
- The **"Arrgh, it's 3AM WTF?"** Rule of Thumb: **5.0**. If your load average is above 5.00, you could be in serious trouble, your box is either hanging or slowing way down, and this will (inexplicably) happen in the worst possible time like in the middle of the night or when you're presenting at a conference. Don't let it get there.

What about Multi-processors? My load says 3.00, but things are running fine!

Got a quad-processor system? It's still healthy with a load of 3.00.

On multi-processor system, the load is relative to the number of processor cores available. The "100% utilization" mark is 1.00 on a single-core system, 2.00, on a dual-core, 4.00 on a quad-core, etc.

If we go back to the bridge analogy, the "1.00" really means "one lane's worth of traffic". On a one-lane bridge, that means it's filled up. On a two-late bridge, a load of 1.00 means its at 50% capacity -- only one lane is full, so there's another whole lane that can be filled.

 = load of 2.00 on two-lane road

Same with CPUs: a load of 1.00 is 100% CPU utilization on single-core box. On a dual-core box, a load of 2.00 is 100% CPU utilization.

Multicore vs. multiprocessor

While we're on the topic, let's talk about multicore vs. multiprocessor. For performance purposes, is a machine with a single dual-core processor basically equivalent to a machine with two processors with one core each? Yes. Roughly. There are lots of subtleties here concerning amount of cache, frequency of process hand-offs between processors, etc. Despite those finer points, for the purposes of sizing up the CPU load value, the *total number of cores* is what matters, regardless of how many physical processors those cores are spread across.

Which leads us to a two new Rules of Thumb:

- *The "number of cores = max load"* Rule of Thumb: on a multicore system, your load should not exceed the number of cores available.
- The *"cores is cores"* Rule of Thumb: How the cores are spread out over CPUs doesn't matter. Two quad-cores == four dual-cores == eight single-cores. It's all eight cores for these purposes.

Bringing It Home

Let's take a look at the load averages output from `uptime`:

~ $ uptime
23:05 up 14 days, 6:08, 7 users, load averages: 0.65 0.42 0.36

This is on a dual-core CPU, so we've got lots of headroom. I won't even think about it until load gets and stays above 1.7 or so.

Now, what about those three numbers? 0.65 is the average over the last minute, 0.42 is the average over the last five minutes, and 0.36 is the average over the last 15 minutes. Which brings us to the question:

Which average should I be observing? One, five, or 15 minute?

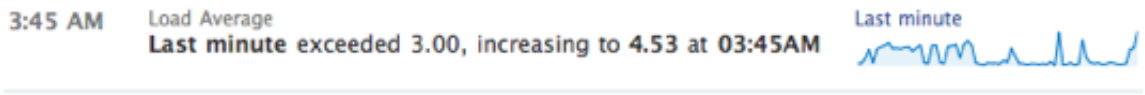
For the numbers we've talked about (1.00 = fix it now, etc), you should be looking at the five or 15-minute averages. Frankly, if your box spikes above 1.0 on the one-minute average, you're still fine. It's when the 15-minute average goes north of 1.0 and stays there that you need to snap to. (obviously, as we've learned, adjust these numbers to the number of processor cores your system has).

So # of cores is important to interpreting load averages ... how do I know how many cores my system has?

`cat /proc/cpuinfo` to get info on each processor in your system. *Note: not available on OSX, Google for alternatives*. To get just a count, run it through `grep` and word count: `grep 'model name' /proc/cpuinfo | wc -l`

Monitoring Linux CPU Load with Scout

Scout provides 2 ways to modify the CPU load. Our original server load plugin and Jesse Newland's Load-Per-Processor plugin both report the CPU load and alert you when the load peaks and/or is trending in the wrong direction:



More Reading

- Wikipedia - A good, brief explanation of Load Average; it goes a bit deeper into the mathematics
- Linux Journal - very well-written article, goes deeper than either this post or the wikipedia entry.

More on Linux Performance

- Understanding Disk I/O - when should you be worried?
- Determining free memory on Linux

Subscribe to our RSS feed or **follow us on Twitter** for more insight on Linux performance.

Comments

1. ***mjablonski*** said 3 days later:

   Very well explained, thank you very much for sharing this great article!!!

2. ***Luke Hartman*** said 3 days later:

   Indeed. I never was quite sure what the #s meant, except that lower is better. Thanks for the clarification.

3. ***John Liptak*** said 3 days later:

   I would add a paragraph about CPU bound applications. For example, if you have a ray tracing app that will use all of CPU that you can give it, your load average will always be > 1.0. It's not a problem if you planned it that way.

4. ***John Walker*** said 3 days later:

   Nice explanation. Now can you explain vmstat and how to understand server io ;)

5. ***feelinglucky*** said 3 days later:

   nice article, this is chinese translated version: http://www.gracecode.com/archives/2973/

6. ***Gimi*** said 4 days later:

   Great writes! Learned a lot from it. Goooood work!

7. ***A real linux admin*** said 4 days later:

   You forgot to mention that any process in IOWAIT is considered on the CPU . So you could have one process spending most of its time waiting on a slow NFS share that could drive your load average up to 1.0, even though your CPU is virtually idle.

8. ***Colin*** said 4 days later:

   typo guy: eight sign-cores is single?

   Fell free to remove after you fix

9. ***skinp*** said 4 days later:

   Great info, thanks. I actually always though the RAM was also taken in count for the load average.

10. ***Derek (Scout)*** said 4 days later:

    Thanks Colin. Fixed.

11. ***Buttson*** said 4 days later:

    grep -c will give you a count without needing to pipe through wc.

12. ***Stephen*** said 4 days later:

    In addition to the CPU bound processed, you've not mentioned nice. If you have processes that are very low priority (high nice value), they may not interfere with interactive use of the machine. For example, the machine i'm typing on is quite responsive. The load average is almost a little over 4. It's a dual CPU . 4 of the processes are niced as far as they can go. These can be ignored. A little over 4 minus 4 is a little over zero. So, there's no idle time, but the machine does what i want it to do, when i want it to do it.

    And, i have 4 processes running because i want to make use of both CPUs. But the code is written to either run single thread, or use 4 threads, cooperating. Well, that's the breaks.

13. ***chkno*** said 4 days later:

    The above is accurate for CPU -bound loads, but there are other resources in play. For example, if you have a one-CPU one-disk system with a load average of 15, do not just run out and buy a 16-core one-disk system. If all those processes are waiting on disk i/o, you would be much better off with a one-CPU 16-disk system (or an ssd).

14. ***jet*** said 4 days later:

    Well done

15. ***Chris*** said 4 days later:

    While I understood the basics of the load rule (over .7 watch out) I never truly got how it scaled to multi-core processors or how it broke it down into multiple minute time frames. Thanks!

One question though, the Pent4 had hyperthreading which “faked” a second core which could help, or seriously hinder performance depending upon the code. Its come back now with the i7/xeon derivatives (not sure about the new i5’s). How does this affect the values to be wary of? Or does it? Since its a “virtual” core in a sense it can help, but too much or the wrong kind of work can starve your main core and cause serious problems.

Any thoughts?

16. ***oldschool*** said 4 days later:

    One slight disagreement…in large scale processing applications, a load of .7 is considered wasteful. While a web server may need available headroom for unexpected traffic spikes, a server dedicated to processing data should be worked as hard as possible. Consider the difference between the plumbing to and from your house (and neighbor hood) versus an oil pipeline to Alaska. You expect the plumbing for sporadically used water or sewage to and from your house to handle unexpected usage patterns. The pipeline from Alaska should be running at full capacity at all times to avoid costly down time. Large grid computing makes use of this model by allowing your servers to have the additional capacity for spikes, while consuming the idle cpu cycles when possible.

Subscribe

Browse