Per-entity load tracking

By **Jonathan Corbet**
January 9, 2013

The Linux kernel's CPU scheduler has a challenging task: it must allocate access to the system's processors in a way that is fair and responsive while maximizing system throughput and minimizing power consumption. Users expect these results regardless of the characteristics of their own workloads, and regardless of the fact that those objectives are often in conflict with each other. So it is not surprising that the kernel has been through a few CPU schedulers over the years. That said, things have seemed relatively stable in recent times; the current "completely fair scheduler" (CFS) was merged for 2.6.23 in 2007. But, behind that apparent stability, a lot has changed, and one of the most significant changes in some time was merged for the 3.8 release.

Perfect scheduling requires a crystal ball; when the kernel knows exactly what demands every process will make on the system and when, it can schedule those processes optimally. Unfortunately, hardware manufacturers continue to push affordable prediction-offload devices back in their roadmaps, so the scheduler has to be able to muddle through in their absence. Said muddling tends to be based on the information that is actually available, with each process's past performance being at the top of the list. But, interestingly, while the kernel closely tracks how much time each process actually spends running, it does not have a clear idea of how much each process is contributing to the load on the system.

One might well ask whether there is a difference between "CPU time consumed" and "load." The answer, at least as embodied in Paul Turner's per-entity load tracking patch set, which was merged for 3.8, would appear to be "yes." A process can contribute to load even if it is not actually running at the moment; a process waiting for its turn in the CPU is an example. "Load" is also meant to be an instantaneous quantity — how much is a process loading the system right now? — as opposed to a cumulative property like CPU usage. A long-running process that consumed vast amounts of processor time last week may have very modest needs at the moment; such a process is contributing very little to load now, despite its rather more demanding behavior in the past.

The CFS scheduler (in 3.7 and prior kernels) tracks load on a per-run-queue basis. It's worth noting that "the" run queue in CFS is actually a long list of queues; at a minimum, there is one for each CPU. When group scheduling is in use, though, each control group has its own per-CPU run queue array. Occasionally the scheduler will account for how much each run queue is contributing to the load on the system as a whole. Accounting at that level is sufficient to help the group scheduler allocate CPU time between control groups, but it leaves the system as a whole unaware of exactly where the current load is coming from. Tracking load at the run queue level also tends to yield widely varying estimates even when the workload is relatively stable.

Toward better load tracking

Per-entity load tracking addresses these problems by pushing this tracking down to the level of individual "scheduling entities" — a process or a control group full of processes. To that end, (wall clock) time is viewed as a sequence of 1ms (actually, 1024µs) periods. An entity's contribution to the system load in a period p_i is just the portion of that period that the entity was runnable — either actually running, or waiting for an available CPU. The trick, though, is to get an idea of contributed load that covers more than 1ms of real time; this is managed by adding in a decayed version of the entity's previous contribution to system load. If we let L_i designate the entity's load contribution in period p_i, then an entity's total contribution can be expressed as:

$$L = L_0 + L_1*y + L_2*y^2 + L_3*y^3 + ...$$

Where y is the decay factor chosen. This formula gives the most weight to the most recent load, but allows past load to influence the calculation in a decreasing manner. The nice thing about this series is that it is not actually necessary to keep an array of past load contributions; simply multiplying the previous period's total load contribution by y and adding the new L_0 is sufficient.

In the current code, y has been chosen so that y^{32} is equal to 0.5 (though, of course, the calculation is done with integer arithmetic in the kernel). Thus, an entity's load contribution 32ms in the past is weighted half as strongly as its current contribution.

Once we have an idea of the load contributed by runnable processes, that load can be propagated upward to any containing control groups with a simple sum. But, naturally, there are some complications. Calculating the load contribution of runnable entities is easy, since the scheduler has to deal with those entities on a regular basis anyway. But non-runnable entities can also contribute to load; the fact a password cracker is currently waiting on a page fault does not change the fact that it may be loading the system heavily. So there needs to be a way of tracking the load contribution of processes that, by virtue of being blocked, are not currently the scheduler's concern.

One could, of course, just iterate through all those processes, decay their load contribution as usual, and add it to the total. But that would be a prohibitively expensive thing to do. So, instead, the 3.8 scheduler will simply maintain a separate sum of the "blocked load" contained in each `cfs_rq` (control-group run queue) structure. When a process blocks, its load is subtracted from the total runnable load value and added to the blocked load instead. That load can be decayed in the same manner (by multiplying it by y each period). When a blocked process becomes runnable again, its (suitably decayed) load is transferred back to the runnable load. Thus, with a bit of accounting during process state transitions, the scheduler can track load without having to worry about walking through a long list of blocked processes.

Another complication is throttled processes — those that are running under the CFS bandwidth controller and have used all of the CPU time available to them in the current period. Even if those processes wish to run, and even if the CPU is otherwise idle, the scheduler will pass them over. Throttled processes thus cannot contribute to load, so removing their contribution while they languish makes sense. But allowing their load contribution to decay while they are waiting to be allowed to run again would tend to skew their contribution downward. So, in the throttled case, time simply stops for the affected processes until they emerge from the throttled state.

What it is good for

The end result of all this work is that the scheduler now has a much clearer idea of how much each process and scheduler control group is contributing to the load on the system — and it has all been achieved without an increase in scheduler overhead. Better statistics are usually good, but one might wonder whether this information is truly useful for the scheduler.

It does seem that some useful things can be done with a better idea of an entity's load contribution. The most obvious target is likely to be load balancing: distributing the processes on the system so that each CPU is carrying roughly the same load. If the kernel knows how much each process is contributing to system load, it can easily calculate the effect of migrating that process to another CPU. The result should be more accurate, less error-prone load balancing. There are some patches in circulation that make use of load tracking to improve the scheduler's load balancer; something will almost certainly make its way toward the mainline in the near future.

Another feature needing per-entity load tracking is the small-task packing patch. The goal here is to gather "small" processes onto a small number of CPUs, allowing other processors in the system to be powered down. Clearly, this kind of gathering requires a reliable indicator of which processes are "small"; otherwise, the system is likely to end up in a highly unbalanced state.

Other subsystems may also be able to use this information; CPU frequency and power governors should be able to make better guesses about how much computing power will be needed in the near future, for example. Now that the infrastructure is in place, we are likely to see a number of developers using per-

entity load information to optimize the behavior of the system. It is still not a crystal ball with a view into the future, but, at least, we now have a better understanding of the present.

Index entries for this article
Kernel Scheduler/Load tracking

(Log in to post comments)

Per-entity load tracking

Posted Jan 11, 2013 3:31 UTC (Fri) by **thoffman** (guest, #3063) [Link]

"Perfect scheduling requires a crystal ball; when the kernel knows exactly what demands every process will make on the system and when, it can schedule those processes optimally".

Actually, I'm pretty sure that's NP-hard, at least for some definitions of optimal. So it might very well be counterproductive to spend the time obtaining an optimal scheduling.

Reply to this comment

Per-entity load tracking

Posted Jan 11, 2013 9:06 UTC (Fri) by **ekj** (guest, #1524) [Link]

NP-hard doesn't automatically mean "too hard", just like "solvable in polynomial time" doesn't automatically mean "practical".

I know you didn't claim that, it's just that I've seen one-to-many arguments that consist of "NP-hard, therefore not doable"

If a problem scales as 1.1**n and n is expected to be 100 at most, then it's easily doable (barring huge constant factors), a n**7 algorithm is actually more computationally intensive up to a n of about 350.

I don't know how large the problem-space tends to be for scheduling-problems though.

Reply to this comment

Per-entity load tracking

Posted Jan 24, 2013 1:33 UTC (Thu) by **igorrs** (guest, #88981) [Link]

Either you misunderstood the quoted text or you quoted the wrong part.

If you were after PERFECT scheduling, finding the solution would require actual prediction of the future (with a crystal ball, for example ;-)). Time complexity was not at stake in that specific sentence.

Reply to this comment

Per-entity load tracking

Posted Jan 11, 2013 9:01 UTC (Fri) by **dvdeug** (subscriber, #10998) [Link]

I find this obsession with CPU scheduling a little weird. YMMV, but from seat it's not a problem; Linux performs smoothly under CPU load. On the other hand, I/O load or memory pressure can make using my system practically impossible.

Reply to this comment

Per-entity load tracking

Posted Jan 11, 2013 18:19 UTC (Fri) by **mathstuf** (subscriber, #69389) [Link]

Not that I know too much about this, but now that I/O would get factored into load, I would think that things like "make --load-average 8" should work better in the face of heavy I/O. The scheduler should also be able to schedule tasks such that the heavy I/O processes aren't run together. I can see some logic behind using the CPU scheduler to help alleviate I/O slowness since the kernel can't do much about I/O speeds, but it can decide when the processes which have a history of making costly I/O get to make more. Memory pressure issues don't make as much sense in the scheduler, but the code covered in the vmpressure_fd article looks promising.

Reply to this comment

Per-entity load tracking

Posted Jan 11, 2013 22:35 UTC (Fri) by **man_ls** (guest, #15091) [Link]

My experience used to be the same. At least those other problems are eco-friendly: just add lots of memory and an insanely fast SSD and be done with it. While adding CPUs generates huge amounts of heat. If not for the planet, do it for your electricity bill -- or your sanity after a year of listening to the airplane jet on top of your desk.

Nowadays with 4 GB of RAM and an SLC SSD most of my bottlenecks are gone. Oh, and with CONFIG_PREEMPT, of course.

Reply to this comment

Per-entity load tracking

Posted Jan 12, 2013 8:49 UTC (Sat) by **jospoortvliet** (guest, #33164) [Link]

even then, copying data to a slow USB drive can lead to minute-long stalls of the entire system. Very embarrassing that this is still such an issue on linux. I am not sure if it is a hardware problem or a software one...

Reply to this comment

Per-entity load tracking

Posted Jan 12, 2013 14:53 UTC (Sat) by **hummassa** (guest, #307) [Link]

I tend to think it's a problem on "cp", since the same operation, when done via GUI, does not cause (for me) the same stalls. In my head, "cp file /media/usbdrive" tries to copy the files as fast as possible and trips on some strange load condition (ubuntu kernel, most recent, here) while "kde-cp", besides showing a fine progress bar and forecast-time-to-complete, takes approximately the same time to make the copy, without thrashing my whole system.

Reply to this comment

stalls caused by USB drive copy

Posted Jan 12, 2013 19:29 UTC (Sat) by **giraffedata** (guest, #1954) [Link]

You can't blame 'cp'. It's the kernel's job to divide up the resources and 'cp' should selfishly concentrate on getting its own work done. If there is code in the alternative copying software which is specifically designed to leave CPU time slots or whatever for other processes it thinks are more important, I'm against that.

Reply to this comment

stalls caused by USB drive copy

Posted Jan 12, 2013 22:01 UTC (Sat) by **man_ls** (guest, #15091) [Link]

I have never seen this. Have you tried with CONFIG_PREEMPT? The kernel does a much better job of not letting random tasks hoard the CPU (or even CPUs these days), interactivity does not suffer even under arbitrary IO load.

Reply to this comment

stalls caused by USB drive copy

Posted Jan 12, 2013 23:24 UTC (Sat) by **nix** (subscriber, #2304) [Link]

It's got nothing to do with the CPU. This is a longstanding problem whereby with a sufficiently large copy, cp fills memory up with dirty pages awaiting writeout and then everything freezes solid because until the writeout to the slow block device has finished, there's hardly any memory left to do anything.

(It used to be much worse: it used to stall basically all I/O to *any* block device due to, IIRC from vague faded memory, everything sharing a single queue. That's not true anymore. So this has got better, though I can't tell whether I don't see it these days because the problem has improved or because machines just have huge amounts of memory these days...)

Reply to this comment

stalls caused by USB drive copy

Posted Jan 13, 2013 0:14 UTC (Sun) by **man_ls** (guest, #15091) [Link]

Ah, the old "a process paged all my memory" problem. Looks like bad design in cp then. But to trigger it nowadays you would have to copy a file bigger than free memory (which can be a few GB) to a USB key; I am not sure that this is what jospoortvliet and hummassa were reporting above.

The solution for the problem you describe should be, as usual, in several places: stop cp from dirtying more pages than it can move to their destination, and make the kernel avoid this behavior in any process. I really don't know how in the general case. To avoid this kind of problem I don't use swap any more, so that a misbehaving process cannot page everything else to disk. I still get severe slowdowns when memory runs out though; I am still trying to figure out how to avoid them.

Reply to this comment

stalls caused by USB drive copy

Posted Jan 13, 2013 0:22 UTC (Sun) by **nybble41** (subscriber, #55106) [Link]

As I'm hardly an expert, take this with a grain of salt, but I put this line in /etc/sysctl.conf (or /etc/sysctl.d/*.conf):

vm.dirty_bytes = 201326592

This limits total dirty memory to 192MB before the kernel forces processes to work on writing data to disk rather than dirtying new pages. The default is something like 10% of total RAM. It seems to help.

Reply to this comment

stalls caused by USB drive copy

Posted Jan 13, 2013 1:12 UTC (Sun) by **cmccabe** (guest, #60281) [Link]

I don't think the issue here is cp's fault. It's the kernel's job to manage the page cache and writeback.

Reply to this comment

stalls caused by USB drive copy

Posted Jan 13, 2013 1:59 UTC (Sun) by **hummassa** (guest, #307) [Link]

if, as nybble41 stated above, vm.dirty_bytes really works, the blame is in the kernel that has a bad default (10% of all RAM) because RAM is cheaper nowadays, but not proportionally faster to access. It causes contention of the cleaning the dirty pages somehow. "cp" just happens to trigger this bug. For information, I can trigger this bug by copying anything (it does not seem to be necessary to be a single file, but a single file happens to trigger it more aggressively apparently) above 1/2 GiB via USB.
Trying to backup my VMs (20G each file) to a USB drive is a nightmare. Usually I do "nice -20 cp" or, more commonly, "kde-cp".

Reply to this comment

stalls caused by USB drive copy

Posted Apr 11, 2013 20:28 UTC (Thu) by **serzan** (subscriber, #8155) [Link]

> Usually I do "nice -20 cp" or, more commonly, "kde-cp".

have you tried ionice -c 3? though not sure it'd have much of an impact, if thrashing memory is the cause

Reply to this comment

stalls caused by USB drive copy

Posted Mar 17, 2017 15:37 UTC (Fri) by **plhk** (guest, #111928) [Link]

https://www.spinics.net/lists/linux-mm/msg64864.html

Reply to this comment

Per-entity load tracking

Posted Jan 13, 2013 9:32 UTC (Sun) by **BlueLightning** (subscriber, #38978) [Link]

even then, copying data to a slow USB drive can lead to minute-long stalls of the entire system. Very embarrassing that this is still such an issue on linux. I am not sure if it is a hardware problem or a software one...

Have you seen this issue recently? I'm sure I read an LWN article a year or two ago where it was described how this issue was tracked down to a change in the I/O buffer window sizing or something similar (and fixed).

Reply to this comment

Per-entity load tracking

Posted Jan 13, 2013 12:56 UTC (Sun) by **hummassa** (guest, #307) [Link]

I had this problem fairly recently (two months ago), i suppose my kernel was already Quantal's (today it's 3.5.0-21).

Reply to this comment

Per-entity load tracking

Posted Jan 12, 2013 1:57 UTC (Sat) by **butlerm** (subscriber, #13312) [Link]

> Linux performs smoothly under CPU load.

Historically speaking, Linux has always prioritized throughput over "smoothness". If you want something resembling soft real time response, you must abstain from using EXT3, set the swappiness to zero, and use a kernel compiled with CONFIG_PREEMPT or something like it, a version not afflicted with the more recent "stable pages" feature.

Reply to this comment

Per-entity load tracking

Posted Jan 12, 2013 19:43 UTC (Sat) by **giraffedata** (guest, #1954) [Link]

Adding to the confusion over the historical focus on CPU time like it's the only resource, this project (apparently for the first time) adds paging device time. So now it's an arbitrary *pair* of resources, while still leaving out other major resources, including filesystem device time and network time.

Reply to this comment