Contribution ID: **190** Type: **not specified**

Much Ado About···Migrations!

Monday, 20 September 2021 07:05 (30 minutes)

The Linux scheduler shuffles tasks around on the various CPUs all the time, as mandated by the implementation of a combination of policies and heuristics. In general, this works well for many different workloads and lets us achieve a more than acceptable compromise among often conflicting goals, such as maximum throughput, minimum latency, reasonable energy consumption, etc. Furthermore, for the cases that really have special requirements, there are knobs to poke –such as different scheduling policies, priorities, affinity, up to CPU isolation– for steering the scheduler toward any desired behavior.

Nevertheless, we believe that there are cases where a less "migration prone" attitude from the scheduler could be beneficial, e.g., CPU-bound tasks (possibly HPC workloads) or the virtual CPUs of a virtual machine (at least in some circumstances). These cases would benefit from having tasks a bit more "sticky" to the CPUs where they are running, but for which pinning or a custom policy would be infeasible for the user to be configured. Or maybe it's fine to pin or change the priority, but then this means that we need to know what is causing the unwanted migrations, in order to roll out the best counter-measures. For instance, if I can figure out that my task is often migrated because it is preempted by others, I can think about rising its priority and/or rebalancing (or reconsidering) the load on my system.

We therefore started our investigation around migrations. Basically, with a task migration event as our starting point, we wanted to see if it was possible to figure out what other events caused it to actually occur, and how far back in the chain of such events we could get. We are using a combination of existing (the various tracing facilities) and new (e.g., Sched struct retriever) tools and techniques. We wanted to start really simple and looked at what happens to a very basic main(){while(1);} task, and discovered that on a CPU with multiple cores and multiple threads, it migrates among different cores a bit more than what we expected. If we switch off hyperthreading, though, cross-cores migrations disappear too…

So, even if we are still at the beginning, the tools we are using are still work-in-progress and the one above is just one example, we want to present the current status of this activity to the community, in case anyone else is also interested or has any feedback.

I agree to abide by the anti-harassment policy

I agree

Primary authors: CIRAOLO, Francesco (University of Turin); FAGGIOLI, Dario (SUSE); BINI, Enrico (University of Turin)

Presenters: CIRAOLO, Francesco (University of Turin); FAGGIOLI, Dario (SUSE); BINI, Enrico (University of Turin)

Session Classification: Scheduler MC

Track Classification: Scheduler MC