Linaro connect
San Francisco 2017

Deadline Scheduling Accounting, CPUset and CPUhotplug

Mathieu Poirier

In This Presentation

- Describe a classic feature interaction problem between:
  - The deadline (DL) scheduler
  - CPUhotplug
  - CPUset
- Short overview of the initial RFC published before Linux Plumbers
- Open discussion on how best to approach the remaining work

- We are **not** talking about…
  - How the Linux scheduler works
  - What is deadline scheduling
  - How deadline scheduling works

A Very Simple Problem

- Deadline accounting metrics are lost when...
  - A CPU is hotpluged in or out of a system
  - CPUsets are created or destroyed

- Why is this a problem?
  - Losing track of deadline utilisation leads to a collapse of the mathematical model behind DL
  - Under utilisation of the system's potential
  - Over selling capacity → failure of the system to honor deadline contracts

- Steve Rostedt's initial report [1].

[1]. https://lkml.org/lkml/2016/2/3/966

Before a Hotplug Operation...

```
root@linaro-developer:/home/linaro# grep dl /proc/sched_debug
  dl_rq[0]:
  .dl_nr_running                 : 0
  .dl_nr_migratory               : 0
  .dl_bw->bw                     : 996147
  .dl_bw->total_bw               : 629145
 Dl_rq[1]:
  .dl_nr_running                 : 0
  .dl_nr_migratory               : 0
  .dl_bw->bw                     : 996147
  .dl_bw->total_bw               : 629145
dl_rq[2]:
  .dl_nr_running                 : 1
  .dl_nr_migratory               : 1
  .dl_bw->bw                     : 996147
  .dl_bw->total_bw               : 629145
dl_rq[3]:
  .dl_nr_running                 : 0
  .dl_nr_migratory               : 0
  .dl_bw->bw                     : 996147
  .dl_bw->total_bw               : 629145
```

<------ Result of a 6:10 DL reservation is accounted on all runqueues included in the root domain

After a Hotplug Operation...

```
root@linaro-developer:/home/linaro# grep dl /proc/sched_debug
  dl_rq[0]:
  .dl_nr_running                 : 0
  .dl_nr_migratory               : 0
  .dl_bw->bw                     : 996147
  .dl_bw->total_bw               : 0          <------ DL accounting is lost on all the runqueues
 Dl_rq[1]:                                             in the root domain
  .dl_nr_running                 : 0
  .dl_nr_migratory               : 0
  .dl_bw->bw                     : 996147
  .dl_bw->total_bw               : 0
dl_rq[2]:
  .dl_nr_running                 : 1
  .dl_nr_migratory               : 1
  .dl_bw->bw                     : 996147
  .dl_bw->total_bw               : 0
```

- The same can be achieved with various CPUset operations

Why Is This Happening?

- When a CPUhotplug/CPUset operation happens:
  - Runqueues are removed from the **current** root domain and added to the **default** root domain
  - **Current** root domains are deleted
  - **New** root domains are created
  - Runqueues are removed from the **default** root domain and added to the **new** root domains

- DL accounting metrics are lost in the first step
- For CPUhotplug the above is split in a two-step (and asynchronous) process
- Tricky to fix because the code is shared between CPUhotplug and CPUset

Obvious Solution, Not So Easy To Implement

- Obviously recompute DL bandwidth utilisation when new root domains are created

But…

- Common code between CPUhotplug and CPUset
  - But the processes calling the common code is different
    - CPUhotplug → Asynchronous
    - CPUSet → Synchronous

- Problems to keep in mind:
  - Tasks are removed from runqueues when they get suspended → they can go anywhere
  - Tasks can span more than one root domain → not supported by DL scheduler

The Current Solution

- An RFC that fix the problem using CPUsets has been posted [1]
- It is based on the premise that ***every*** task (suspended or not) in the system belongs to a single CPUset.

- As such to recompute root domains' DL metrics:
  - Circle all the CPUsets in the system
  - If a DL task is found, get a reference to the root domain it is associated to [2]
  - Add DL task utilisation to the root domain

[1]. https://marc.info/?l=linux-kernel&m=150291845422763&w=2

[2]. By way of the task's cpu_allowed field and runqueues

Pros And Cons Of The Solution

- Advantages:
  - Approach is relatively simple
  - Works for any kind of CPUset topology one can imagine
  - Deals with running, runnable and suspended task in the same way

- Disadvantages:
  - Currently prevents tasks from belonging to more than a single root domain
  - Hard to cover all the scenarios that can lead to the above

Task In Multiple Root Domains

How can this happen?

- By default most new tasks can use all the CPUs in the system
- New tasks belong to the default CPUset [1]
- Newly created CPUsets will have their own root domain
- Existing tasks aren't impacted and as such are allowed to use the CPUs in the new root domains, something that breaks the DL scheduler acceptance test

[1].Tasks need to be explicitly assigned to a CPUset in order to belong to that set

Open Question

How do we deal with tasks spanning more than a single root domain?

- Continue with what the current RFC does and prevent DL tasks from spanning more than one root domain?
  - Somewhat simple
  - Hackish and brittle (in my opinion)

- Update the DL scheduler so that it can deal with tasks spanning more than one root domain?
  - Substantial amount work
  - Possible ramifications on the DL scheduler's mathematical model

Thoughts and ideas on how to move forward are welcomed...

Linaro connect
San Francisco 2017

Thank You

#SFO17

SFO17 keynotes and videos on: connect.linaro.org

For further information: www.linaro.org