Linux Kernel Support for Kernel Thread Starvation Avoidance

Real-Time MC, Linux Plumbers Conference 2021

Sharan Turlapati (sturlapati@vmware.com)
Srivatsa Bhat (srivatsa@csail.mit.edu)

VMware Photon OS Team
21 Sep 2021
Agenda

Introduction

Problem Statement

Existing Solutions & Limitations

Design and Implementation of Stall Monitor

Challenges and Feedback
Overview of Telco/RAN: Radio Access Network for 5G
Overview of Telco/RAN: Radio Access Network for 5G

Network Packets

Radio Tower

FlexRAN Processing Pipeline

Server running Linux PREEMPT_RT

Data Center

Data transfer latency

[250 us (fiber-link) + 1.5 ms (radio-relative)]

RT Scheduling + Processing Latency

(~1ms)
Overview of Telco/RAN: Radio Access Network for 5G

- Radio Tower
- Network Packets
- FlexRAN Processing Pipeline
- Server running Linux PREEMPT_RT
- Data Center

Data transfer latency:
- [250 us (fiber-link) + 1.5 ms (radio-relative)]

RT Scheduling + Processing Latency:
- (~1ms)

Fixed total latency budget for packet Tx + processing + ack (< 3ms)

Cyclic test latency < 10us
Problem Statement

Task Priority

Housekeeping Cores

Nohz_full Isolated Cores
Problem Statement

- L1 app FIFO/90
  - Housekeeping Cores
  - Nohz_full Isolated Cores
Problem Statement

- System services (sshd etc)
- k8s ctrl plane OTHER/0
- L1 app FIFO/90

Task Priority

Housekeeping Cores

Nohz_full Isolated Cores
Problem Statement

- L1 app FIFO/90
- kthread FIFO/1
- System services (sshd etc)
- k8s ctrl plane OTHER/0
- Nohz_full Isolated Cores

- Housekeeping Cores

Task Priority
Problem Statement

Problem: Starved kthreads lead to cascading lockups (hang)
Problem Statement

Problem: Starved kthreads lead to cascading lockups (hang)

Goal: OS must remain stable, limiting the fault-domain to the RT app
Problem Statement Example: Container destroy causes hang
Problem Statement Example: Container destroy causes hang

Reproducer:
1. Run high prio CPU hog on an isolated CPU
2. Create & destroy a docker container on a housekeeping CPU
Problem Statement Example: Container destroy causes hang

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>P</th>
<th>S</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>S</td>
<td>[cpu/hp/3]</td>
</tr>
<tr>
<td>38</td>
<td>root</td>
<td>rt</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>S</td>
<td>[migration/3]</td>
</tr>
<tr>
<td>39</td>
<td>root</td>
<td>rt</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>S</td>
<td>[posix/putmr/3]</td>
</tr>
<tr>
<td>40</td>
<td>root</td>
<td>-2</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>S</td>
<td>[rcuc/3]</td>
</tr>
<tr>
<td>41</td>
<td>root</td>
<td>-2</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>S</td>
<td>[ktime/softd/3]</td>
</tr>
<tr>
<td>42</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>S</td>
<td>[ksoft/rd/3]</td>
</tr>
<tr>
<td>44</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>I</td>
<td>[kworker/3:mm/pcpu.wq]</td>
</tr>
<tr>
<td>44</td>
<td>root</td>
<td>0</td>
<td>-20</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>I</td>
<td>[kworker/3:mm/pcpu.wq]</td>
</tr>
<tr>
<td>270</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>I</td>
<td>[kworker/3:mm/pcpu.wq]</td>
</tr>
<tr>
<td>1334</td>
<td>root</td>
<td>0</td>
<td>-20</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>3</td>
<td>R</td>
<td>[kworker/3:mm/pcpu.wq]</td>
</tr>
<tr>
<td>3068</td>
<td>root</td>
<td>-56</td>
<td>0</td>
<td>2.1m</td>
<td>0.7m</td>
<td>99.9</td>
<td>0.0</td>
<td>1:32</td>
<td>3</td>
<td>R</td>
<td>/loop-rt</td>
</tr>
</tbody>
</table>
Problem Statement Example: Container destroy causes hang

CPU 3 is nohz_full isolated
Problem Statement Example: Container destroy causes hang

```
<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>P</th>
<th>S</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>S</td>
<td>[cpuhp/3]</td>
</tr>
<tr>
<td>38</td>
<td>root</td>
<td>rt</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.09</td>
<td>3</td>
<td>S</td>
<td>[migration/3]</td>
</tr>
<tr>
<td>39</td>
<td>root</td>
<td>rt</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>S</td>
<td>[posixcpumr/3]</td>
</tr>
<tr>
<td>40</td>
<td>root</td>
<td>-2</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>S</td>
<td>[rcuc/3]</td>
</tr>
<tr>
<td>41</td>
<td>root</td>
<td>-2</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>S</td>
<td>[ktimevald/3]</td>
</tr>
<tr>
<td>42</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>S</td>
<td>[ksoftirqd/3]</td>
</tr>
<tr>
<td>43</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>S</td>
<td>[kworker/3:0-mm_percpu_wq]</td>
</tr>
<tr>
<td>44</td>
<td>root</td>
<td>-20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>I</td>
<td>[kworker/3:0H-events_highpri]</td>
</tr>
<tr>
<td>270</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>I</td>
<td>[kworker/3:1-mm_percpu_wq]</td>
</tr>
<tr>
<td>1334</td>
<td>root</td>
<td>0</td>
<td>20</td>
<td>0.0m</td>
<td>0.0m</td>
<td>0.0</td>
<td>0.6</td>
<td>0:00.00</td>
<td>3</td>
<td>R</td>
<td>[kworker/3:1H-events_highpri]</td>
</tr>
<tr>
<td>3068</td>
<td>root</td>
<td>-56</td>
<td>0</td>
<td>2.1m</td>
<td>0.7m</td>
<td>99.9</td>
<td>0.6</td>
<td>1:32.52</td>
<td>3</td>
<td>R</td>
<td>./loop-rt</td>
</tr>
</tbody>
</table>
```

loop-rt has high RT prio (SCHED_FIFO/55)

Two runnable tasks on CPU 3:
loop-rt and kworker/3
Problem Statement Example: Container destroy causes hang

loop-rt hogs the CPU
kworker/3 is starved
Problem Statement Example: Container destroy causes hang

```
PID USER   PR NI  VIRT  RES  %CPU %MEM    TIME+  COMMAND
37 root    20   0   6.0m   0.0  0.0    0:00.00 S [cpuhp/3]
38 root    30   0   6.0m   0.0  0.0    0:00.00 S [migration/3]
39 root    30   0   6.0m   0.0  0.0    0:00.00 S [posixcpumr/3]
40 root    30   0   6.0m   0.0  0.0    0:00.00 S [rcuc/3]
41 root    30   0   6.0m   0.0  0.0    0:00.00 S [ktimersoftd/3]
42 root    20   0   6.0m   0.0  0.0    0:00.00 S [ksoftirqd/3]
43 root    20   0   6.0m   0.0  0.0    0:00.00 S [kworker/3:0-mm_percpu_wq]
44 root    20   0   6.0m   0.0  0.0    0:00.00 S [kworker/3:0H-events_highpri]
270 root   20   0   6.0m   0.0  0.0    0:00.00 S [kworker/3:11-mm_percpu_wq]
1334 root  -20  0   6.0m   0.0  0.0    0:00.00 S [kworker/3:1H-events_highpri]
3068 root  -56  0   2.1m   0.7  99.9    1:32.52 R [/loop-rt]
```

Stalld DEBUG: Dumping Stack for dockerd(PID = 1021)

```
[<0>] __flush_work+0x13e/0x1e0
[<0>] flush_work+0x10/0x20
[<0>] rollback_registered_many+0x168/0x540
[<0>] unregister_netdevice_many.part.124+0x12/0x90
[<0>] unregister_netdevice_many+0x16/0x20
[<0>] rtlDelete_link+0x3f/0x50
[<0>] rtlDeleteLink+0x121/0xb0
[<0>] rnetlink_rcv_msg+0x12a/0x310
[<0>] netlink_rcv_skb+0x54/0x130
[<0>] rnetlink_rcv+0x15/0x20
[<0>] netlink_unicast+0x17b/0x220
[<0>] netlink_sendmsg+0x2b5/0xb0
[<0>] sock_sendmsg+0x3e/0x50
[<0>] __sys_sendto+0x13f/0x180
[<0>] __x64_sys_sendto+0x28/0x30
[<0>] do_syscall_64+0x960/0x1b0
[<0>] entry_SYSCALL_64 after_hwframe+0x44/0xa9
```
Problem Statement Example: Container destroy causes hang

```
static int rtnetlink_rcv_msg(...) {
    rtnl_lock();
    ->flush_all_backlogs();
    rtnl_unlock();
}
```
Problem Statement Example: Container destroy causes hang

```c
static int rtnetlink_rcv_msg(...) {
    rtnl_lock();
    ->flush_all_backlogs();
    rtnl_unlock();
}
```
Problem Statement Example: Container destroy causes hang

```
static int rtnetlink_rcv_msg(...)
{
    rtnl_lock();
    ->flush_all_backlogs();
    rtnl_unlock();
}
```

Problem pattern is pervasive in Linux. Ex: ext4, cgroups, ftrace, sysctl etc.
Existing solutions & limitations: stalld
Existing solutions & limitations: stalld

Overview of stalld

- Monitors for starving tasks + boosts them using SCHED_DEADLINE
- Revives system by operating within tolerable OS-jitter (user-configurable)
Existing solutions & limitations: stalld

Overview of stalld

- Monitors for starving tasks + boosts them using SCHED_DEADLINE
- Revives system by operating within tolerable OS-jitter (user-configurable)
- Critical to RAN deployments to maintain stability
Existing solutions & limitations: stalld

Overview of stalld
- Monitors for starving tasks + boosts them using SCHED_DEADLINE
- Revives system by operating within tolerable OS-jitter (user-configurable)
- Critical to RAN deployments to maintain stability

Limitations of stalld
Existing solutions & limitations: stalld

Overview of stalld

- Monitors for starving tasks + boosts them using SCHED_DEADLINE
- Revives system by operating within tolerable OS-jitter (user-configurable)
- Critical to RAN deployments to maintain stability

Limitations of stalld

<table>
<thead>
<tr>
<th>Limitation</th>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>Stallds threads run on housekeeping CPUs</td>
</tr>
<tr>
<td>Stalld can get starved itself</td>
<td>Competes for time on housekeeping CPUs</td>
</tr>
<tr>
<td></td>
<td>RT prio stalld is risky – can <strong>cause</strong> stalls itself!</td>
</tr>
<tr>
<td>Unreliable logging</td>
<td>systemd-journald can get stuck</td>
</tr>
<tr>
<td></td>
<td>Verbose logging gets stalld itself stuck</td>
</tr>
<tr>
<td>Trade-off: Response-time vs CPU consumption</td>
<td>Per-CPU threads vs single-threaded mode</td>
</tr>
</tbody>
</table>
Design Goals for Stall Monitor
And why a kernel-based implementation can meet them
Design Goals for Stall Monitor
And why a kernel-based implementation can meet them

- Prevent kthread starvation
  - System hangs are almost always due to starving kernel threads
  - In-kernel starvation avoidance compartmentalizes the fault domain
Design Goals for Stall Monitor

And why a kernel-based implementation can meet them

- Prevent kthread starvation
  - System hangs are almost always due to starving kernel threads
  - In-kernel starvation avoidance compartmentalizes the fault domain

- Ensure scalability
  - Per-cpu kthreads most susceptible to starvation
  - Per-cpu based scheduler hooks scale well
Design Goals for Stall Monitor
And why a kernel-based implementation can meet them

- Prevent kthread starvation
  - System hangs are almost always due to starving kernel threads
  - In-kernel starvation avoidance compartmentalizes the fault domain

- Ensure scalability
  - Per-cpu kthreads most susceptible to starvation
  - Per-cpu based scheduler hooks scale well

- Monitor and boost efficiently
  - Avoid unnecessary periodic monitoring
  - sched events like wakeup and dequeue equip the scheduler to take decisions efficiently
Design Goals for Stall Monitor
And why a kernel-based implementation can meet them

- Prevent kthread starvation
  - System hangs are almost always due to starving kernel threads
  - In-kernel starvation avoidance compartmentalizes the fault domain

- Ensure scalability
  - Per-cpu kthreads most susceptible to starvation
  - Per-cpu based scheduler hooks scale well

- Monitor and boost efficiently
  - Avoid unnecessary periodic monitoring
  - sched events like wakeup and dequeue equip the scheduler to take decisions efficiently

- Guarantee responsiveness
  - We must be able to prevent starvation under any scenario
  - Scheduler invocations inevitably offer the opportunity to monitor for starvation
Design Features of Stall Monitor
Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU.
Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU
- One hrtimer set up (on demand) per cpu to either -
  - Monitor for starving kernel threads (starvation_threshold_time)
    - OR
  - Track the boosted priority duration (boost_duration_time)
Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU
- One hrtimer set up (on demand) per cpu to either -
  - Monitor for starving kernel threads (starvation_threshold_time)
  - OR
  - Track the boosted priority duration (boost_duration_time)
- Boost only one starving kthread on a CPU at any given time
Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU
- One \texttt{hrtimer} set up (on demand) per \texttt{cpu} to either -
  - Monitor for starving kernel threads (\texttt{starvation_threshold_time})
  - OR
  - Track the boosted priority duration (\texttt{boost_duration_time})

- Boost only one starving \texttt{kthread} on a CPU at any given time
- Boost or deboost happens in hardirq context of the \texttt{hrtimer}
Design Features of Stall Monitor

- Each CPU keeps track of starving kernel threads meant to run only on that CPU
- One hrtimer set up (on demand) per cpu to either -
  - Monitor for starving kernel threads (starvation_threshold_time)
  - OR
  - Track the boosted priority duration (boost_duration_time)

- Boost only one starving kthread on a CPU at any given time
- Boost or deboost happens in hardirq context of the hrtimer
- User defined OS jitter
  - With user configurable starvation_threshold_time, boost_duration_time as well as SCHED_DEADLINE parameters
Implementation of Stall Monitor

Enqueue Task

- **is_enqueue_kthread**
  - Yes: Add kthread to starvation monitor list
  - No: Starvation timer active?
    - Yes: Start starvation timer
      - Yes: Finish enqueue_task
  - No: Finish enqueue_task

Hrtimer callback

1. **irq_enter**
2. If not interrupted,:
   - **is_kthread_boosted**
     - Yes: Hrtimer callback handled
     - No: **is_kthread_starving**
       - Yes: hrtimer_forward(boost_duration)
       - No: **is_starvation_list_empty**
         - Yes: Stdproc_reply(HRTIMER_NORESTART)
         - No: hrtimer_forward (starve_threshold)
1. Hrtimer callback handled.
2. Begin irq_exit -> turn off hardirq
3. (sched_setaff()) - boost or deboost
4. Complete irq_exit.
Challenges & Open Questions

- Priority boosting must happen in hardirq context
  - Cannot create more kthreads. Or can we use CPU stopper threads?
  - Better alternatives?

- Restrict the monitoring and boosting to isolcpus only?

- How much latency does it introduce?
Thank you!
Additional Data Points

- CFS code already has functions to track wait times spent by task on the runqueue –
  - Handled by `update_stats_wait_start()` and `update_stats_wait_end()`
  - This needs to be added to RT (SCHED_FIFO and SCHED_RR)

- `__sched_setscheduler` invoked by `sched_setattr()` has checks on `pi` being invoked from interrupt context. This is suspectedly due to `rt_mutex_adjust_prio_chain()` that enables interrupts using `raw_spin_unlock_irq(&task->pi_lock)` unconditionally