THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Towards PREEMPT_RT for the Full Task Isolation

Jim Huang, BiiLabs Co., Ltd.
Oscar Shiang, National Cheng Kung University

Jun 23, 2022

Goals vs. Non-Goals

- Goals
  - Why `NOHZ` is not sufficient for task isolation
  - Identify the source of noise, crucial to PREEMPT_RT
  - Task isolation = escape from noise, by introducing isolation mechanism (existes for a long time)
  - Problem of current task isolation → Definition of “full” task isolation
  - Revisit the evolution of full task isolation ⇒ Meanwhile, review the existing problems.
- Non-goals
  - Jailhouse or hypervisor-based solution
  - Yet another RT patchset ⇒ Minimize the necessary changes, it works even for non-RT.

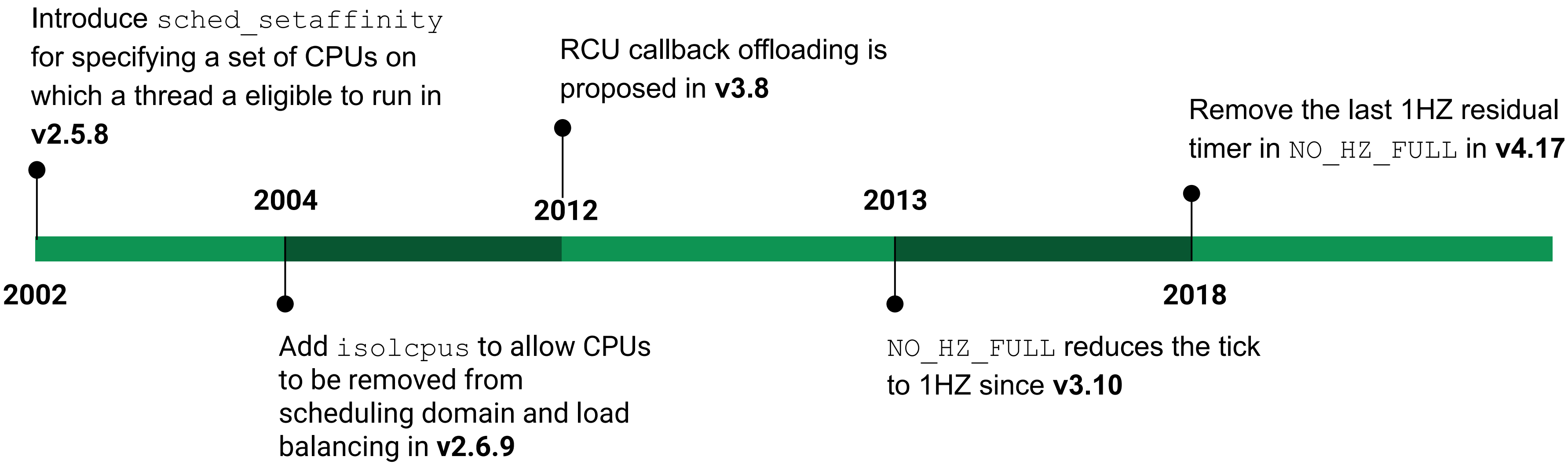
Sources of noises

- Interrupt
  - Interrupt handlers (IRQ, SoftIRQ)
  - Scheduling tick
- I / O ⇒ e.g. blocking to receive data from socket
- Kernel housekeeping works
  - Unbounded works, e.g. rcuo, timer
  - Bounded works, e.g. rcuc, vmstat_update

Full task isolation

- Definition
    - Provides a (nearly) bare-metal-like environment for computationally intensive or real-time applications to run on

Current infrastructure for task isolation



2002

Introduce `sched_setaffinity` for specifying a set of CPUs on which a thread a eligible to run in **v2.5.8**

2004

Add `isolcpus` to allow CPUs to be removed from scheduling domain and load balancing in **v2.6.9**

2012

RCU callback offloading is proposed in **v3.8**

2013

`NO_HZ_FULL` reduces the tick to 1HZ since **v3.10**

2018

Remove the last 1HZ residual timer in `NO_HZ_FULL` in **v4.17**

`sched_setaffinity` mechanism

- The first mechanism for isolating tasks in Linux (v2.5.8)
- Control each CPU affinity mask of the task to indicate which CPUs can it run on
- Need to manipulate each masks to achieve task isolation

CPU isolating mechanism

- Remove the specified CPUs from scheduling domain
- Isolate processes from selected CPUs by default
- Processes will not be migrated to the isolated CPUs during load balancing

`NO_HZ_FULL` mechanism

- Reduce timer tick when the system does not need to do scheduling
- Timer tick may not be disabled easily. ⇒ it has some dependencies:
    - POSIX timer
    - Perf event
    - Clock unstable
    - Scheduler: need to perform preemption
    - RCU callback lifecycle accounting and handling

RCU Callback Offloading Mechanism

- Generally, Linux needs to do grace period accounting and callback invocation to prevent itself from freezing due to RCU
- Callback execution and accounting can add significant jitter
- Offloads RCU callbacks lifecycle handling and execution out of the enqueuer's CPU to specific kthreads instead (rcuo and rcuog)

Problems for Current Infrastructure

- It is suitable for isolating from unbounded works by setting affinity masks or passing `isolcpus=` and `nohz_full=` as kernel parameters
- But it fails to prevent bounded works from interrupting task isolating CPUs, e.g. vmstat_update worker will be queued to per-cpu run queues and executed every second by default

Task Isolation Patches

- Originally proposed by Chris Metcalf (2015)
- Features
  - Provide configuration via **`prctl`**
  - Evaluate the possibility to disable tick at the beginning of task isolation
  - Cancel **`vmstat_update`** worker
  - Drain pagevecs to avoid IPI
- Problem
  - The kernel may busy-wait until there is no more pending timers to run

What Alex Belits did

- Changes based on Chris’ one (2019 - 2020)
  - Prevent IPI from sending to isolated cores
  - Add hooks to enable isolation at syscall, IRQ and IPI entries
- Problems
  - Break some semantic of kernel API, e.g. **`kick_all_cpu_sync`** but will not sync on isolated cpu
  - Race condition when changing isolation mask
  - The modification across several paths including syscalls, IRQ, irqchip
  - ARM64 only

What Marcelo Tosatti did (since 2021)

- Aim to improving KVM’s performance
- Fine-grained configuration, he believe to have the flexibility to decide which interruptions are acceptable to our own system
- Only supports cancelling **`vmstat_update`** worker
  - Less impact to kernel since the frequency of update can be modified via sysctl
  - The cost of updating vmstat is more expensive in KVM
- Problem
  - TIF must be updated if the task isolated task is preempted via preempt_notifier

API Usage (based on Marcelo’s patch)

- **Configure**: set the feature bits you would like to use (only `ISOL_F_QUIESCE_VMSTATS` for now)
- **Activate**: activate specified features

```
unsigned long long fmask;

ret = prctl(PR_ISOL_CFG_GET, I_CFG_FEAT, 0, &fmask, 0);
if (ret != -1 && fmask != 0) {
        ret = prctl(PR_ISOL_ACTIVATE_SET, &fmask, 0, 0, 0);
        if (ret == -1) {
                perror("prctl PR_ISOL_ACTIVATE_SET");
                return ret;
        }
}
```

API Usage (take `oslat` as example)

- Use prctl to mark the beginning and end latency-sensitive section
- Take the mainloop of [`oslat`] as example

```
static void doit(struct thread *t)
{
        unsigned long long isol_mask;
        <...>
        /* Retrieve default configuration */
        ret = prctl(PR_ISOL_CFG_GET, I_CFG_FEAT, 0, &isol_mask, 0);
        if (ret != -1 && isol_mask != 0)
                /* Enable task isolation if supported */
                prctl(PR_ISOL_ACTIVATE_SET, &isol_mask, 0, 0, 0);
        <...>
        /* Disable all task isolation features */
        if (isol_mask != 0) {
                isol_mask = 0;
                prctl(PR_ISOL_ACTIVATE_SET, &isol_mask, 0, 0, 0);
        }
}
```

Benchmarking Tools

- **`oslat` (from rt-tests suite)**: Poll the timer value repeatedly, which can stimulate the some usage, i.e. userspace network driver
- **Function tracer**: kernel tracer which record the behavior of system (including executed functions and events)
- **`OSNOISE` tracer**: new kernel tracer introduced in v5.12. It has similar behavior to oslat but can record more information (actual executing time, type of noise) about the candidate noises

Tools for Tuning and Workload Generation

- **`tuned`**: machine tuning tool developed by Red Hat. It can be used in several scenarios and help us to configure systems in straightforward ways
- **`stress-ng`**: a stress tool that generate various kinds of workload, e.g. VM, timer interrupts,

Benchmarking Scenarios

- The basic idea is to test the behavior and the effectiveness of task isolation patch
- We focus on the scenarios that have intensive accesses to memory, which forces vmstat_update to synchronize the statistic data between cores frequently
- Based on this idea, we design 3 different workloads
  - frequent page faults
  - frequent OOM kills
  - Mixed workload (page faults + OOM kills)

Choice and Configuration on Platforms

- We choose 2 platforms to do experiments
  - Raspberry Pi 4B (ARM64, w/ BCM2711 SoC, Quad core Cortex-A72, 4 GiB RAM)
  - KVM (x86_64, 4 vCore, 4 GiB RAM)
- Both are configured with:
  - /proc/cmdline: skew_tick=1
  - Tuned: use realtime-virtual-host profile to isolated a single core

Benchmarking Steps

1. **Configuration**: choose the tracer, the events we want to record,
2. **Warming-up**: start the workload on non-isolated cores and wait 5 sec for preheating
3. **Benchmarking**: run the tracer and record the possible noises and corresponding events

Note: see detailed steps in osnoise-measure.sh

Experiments

- Based on kernel v5.15.18-rt28, applied with Marcelo’s v12 patches
- Measured by **oslat** from rt-tests, to catch all possible interferences
- Tested on 2 different platforms: **ARM64** and **x86_64 KVM**
- Runed with 3 different workloads generated by stress-ng:
  - Major / minor page faults
  - VM / mmap with OOM
  - Mixed with page faults, VM and mmap

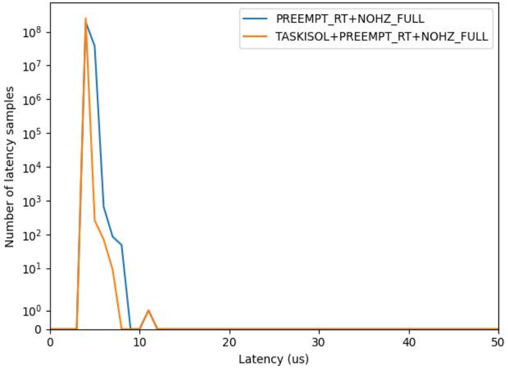
Experiments

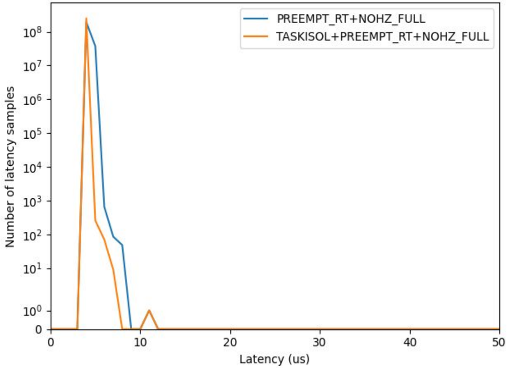
Major / minor page fault

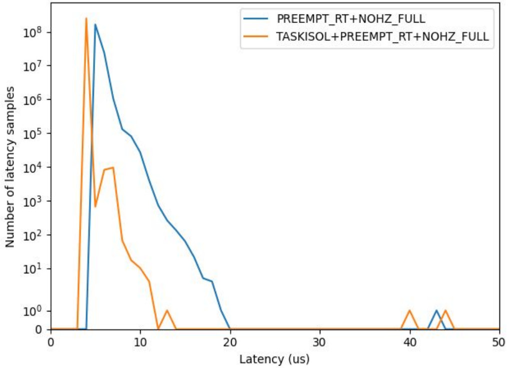
vm / mmap w/OOM

page faults, VM and mmap

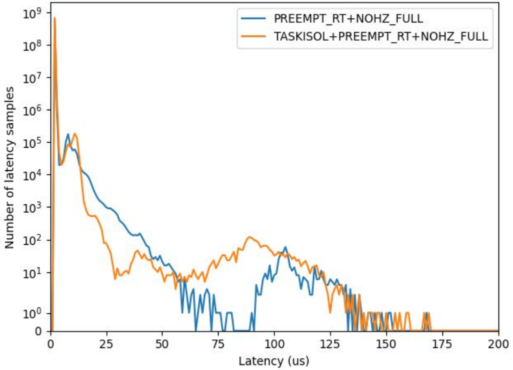
ARM64

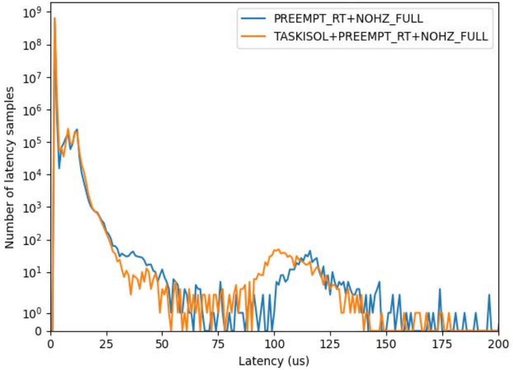


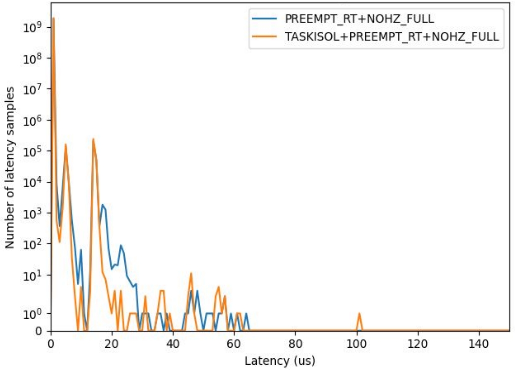




x86_64
KVM







Discussions

- By applying the patches and enabling task isolation, all test cases have lower latencies in average
- In ARM64, since the system is clean and doesn’t run with other applications, task isolation brings an improvement about 2+ us to latency
- For x86_64 KVM, it brings about 10 us latency reduction. It shows that the isolation from vmstat_update is still usable in KVM
- The maximum latency is still high (about 200 us in ARM64 and 900 us in x86_64 KVM) ⇒ there are still other interferences that should be isolated

Conclusion + Insights

- No silver bullet yet – on the way to full task isolation. i.e. , no general solution exists.
- V12 as base, extra efforts are needed for full task isolations