CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

Energy-Efficient I/O: Block层约束设备频率的探索与支持

王建政 vivo 存储系统工程师

目录

CONTENTS

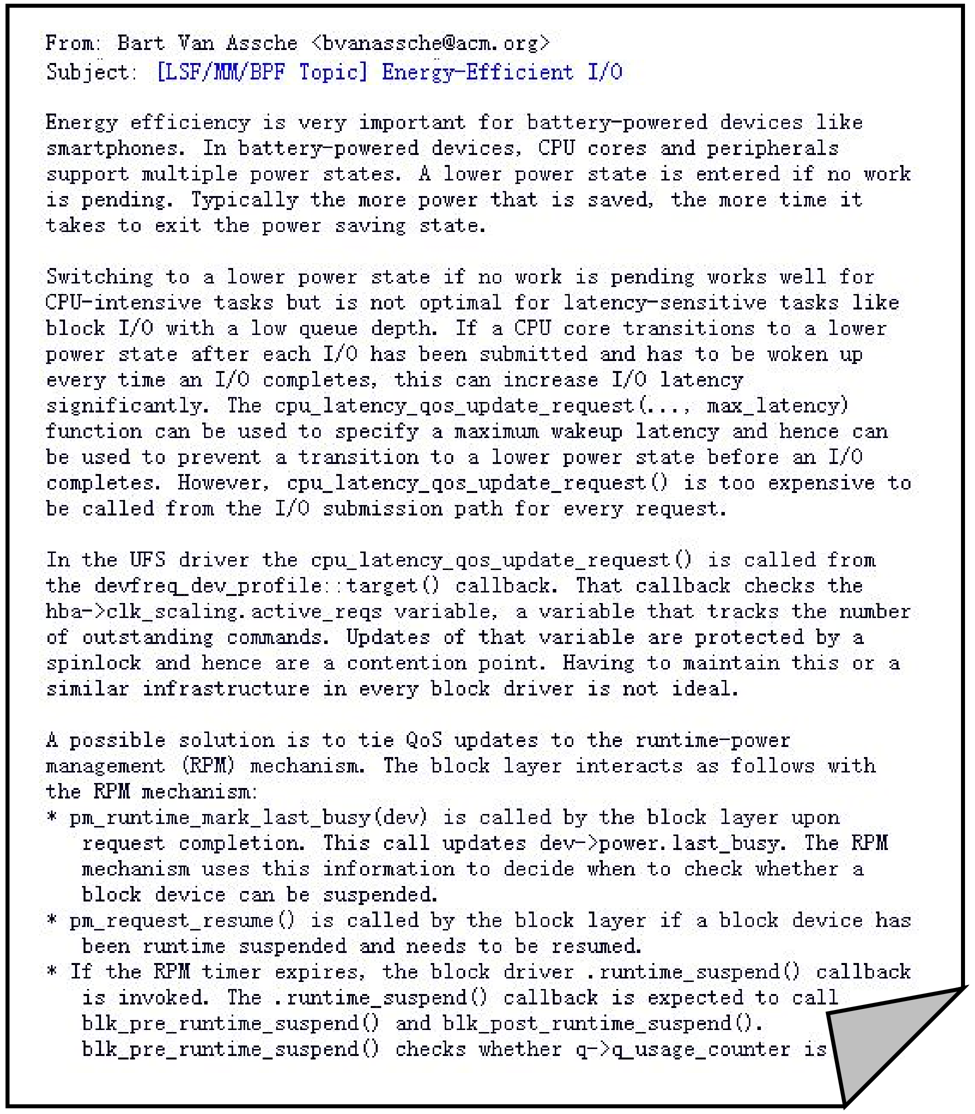
Part 1

背景介绍

Energy-Efficient I/O——当前开发者关注的议题

在 2025 年 LSF/MM 峰会中 Bart Van Assche 提出了“Energy-Efficient I/O”的话题，提出要更多关注电池供电的移动设备的能效表现，并提出**痛点：功耗状态切换时引起时延增加与延迟敏感型任务的性能表现之间存在矛盾。**

- Runtime PM
- Frequency Monitor
- PM QoS

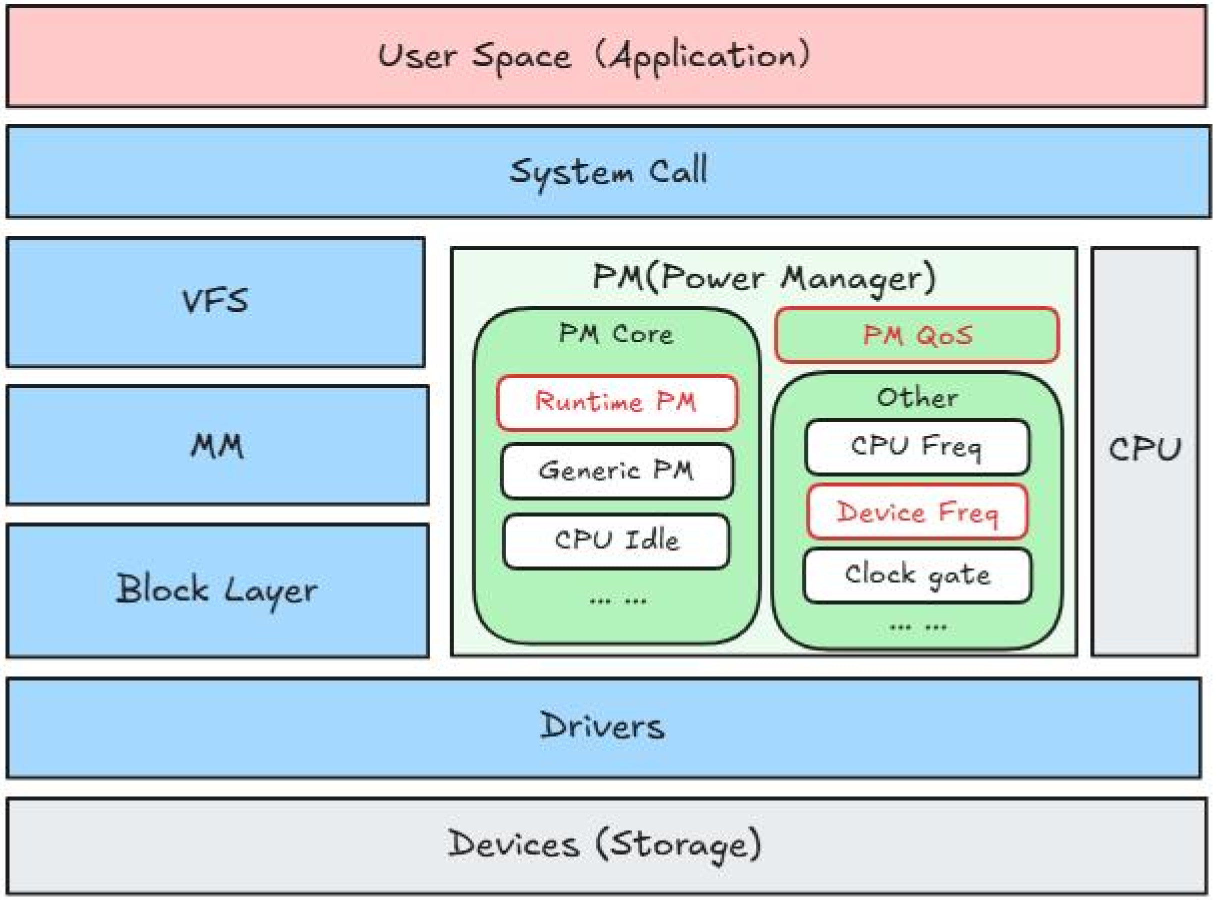


```
From: Bart Van Assche <bvanassche@acm.org>
Subject: [LSF/MM/BPF Topic] Energy-Efficient I/O

Energy efficiency is very important for battery-powered devices like
smartphones. In battery-powered devices, CPU cores and peripherals
support multiple power states. A lower power state is entered if no work
is pending. Typically the more power that is saved, the more time it
takes to exit the power saving state.

Switching to a lower power state if no work is pending works well for
CPU-intensive tasks but is not optimal for latency-sensitive tasks like
block I/O with a low queue depth. If a CPU core transitions to a lower
power state after each I/O has been submitted and has to be woken up
every time an I/O completes, this can increase I/O latency
significantly. The cpu_latency_qos_update_request(..., max_latency)
function can be used to specify a maximum wakeup latency and hence can
be used to prevent a transition to a lower power state before an I/O
completes. However, cpu_latency_qos_update_request() is too expensive to
be called from the I/O submission path for every request.

In the UFS driver the cpu_latency_qos_update_request() is called from
the devfreq_dev_profile::target() callback. That callback checks the
hba->clk_scaling.active_reqs variable, a variable that tracks the number
of outstanding commands. Updates of that variable are protected by a
spinlock and hence are a contention point. Having to maintain this or a
similar infrastructure in every block driver is not ideal.

A possible solution is to tie QoS updates to the runtime-power
management (RPM) mechanism. The block layer interacts as follows with
the RPM mechanism:
* pm_runtime_mark_last_busy(dev) is called by the block layer upon
   request completion. This call updates dev->power.last_busy. The RPM
   mechanism uses this information to decide when to check whether a
   block device can be suspended.
* pm_request_resume() is called by the block layer if a block device has
   been runtime suspended and needs to be resumed.
* If the RPM timer expires, the block driver .runtime_suspend() callback
   is invoked. The .runtime_suspend() callback is expected to call
   blk_pre_runtime_suspend() and blk_post_runtime_suspend().
   blk_pre_runtime_suspend() checks whether q->q_usage_counter is
```

Energy-Efficient I/O——PM模块辅助系统实现更好的能效

Linux中有较多实现了功耗状态切换以达到提升能效的模块

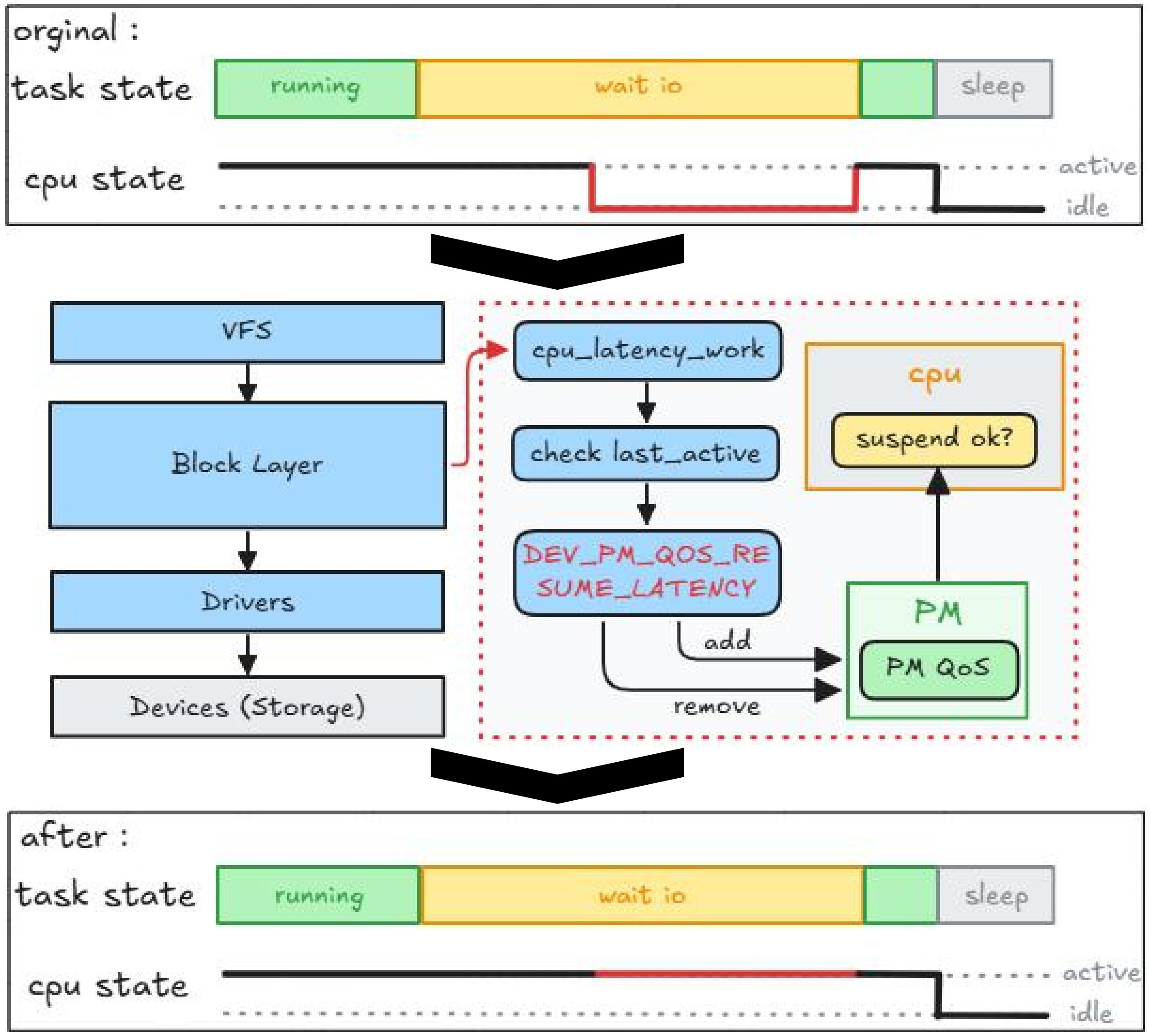
- Runtime PM：实现系统中的部分驱动或子系统独立休眠与唤醒
- Frequency Monitor：通过识别设备工作负载动态调节设备工作频率，以达到更好的能效比
- PM QoS：用于提升PM模块服务质量的模块，包含时延、频率等关键指标



Energy-Efficient I/O——社区针对CPU功耗状态切换的解决方案

主要针对<u>CPU功耗状态切换时影响IO性能</u>的问题，提出了相关的解决方案。

即在Block层中设置PM QoS中的RESUME_LATENCY 质量指标，限制CPU 功耗切换，最终避免了IO请求等中断过程中，CPU陷入idle状态从而引入唤醒耗时，影响IO性能。



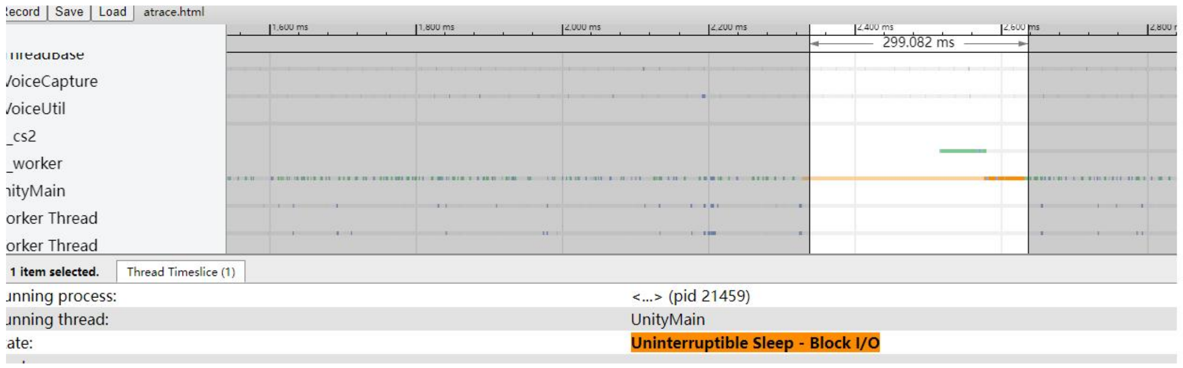
Energy-Efficient I/O——仍需关注Storage功耗状态切换引入的问题

从诸多实际案例中分析发现，Storage的功耗状态切换引入的IO性能问题更为明显：

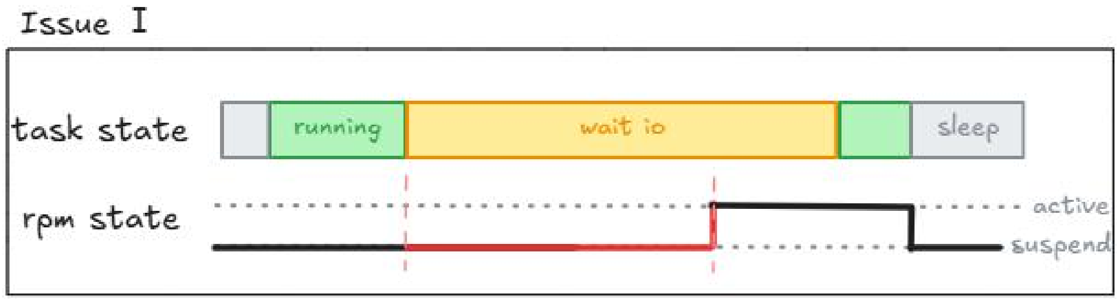
①由于<u>RPM模块</u>中autosuspend机制，驱动会在空闲状态进入休眠。从休眠状态唤醒所产生的耗时，致使首次IO耗时增加，影响性能。

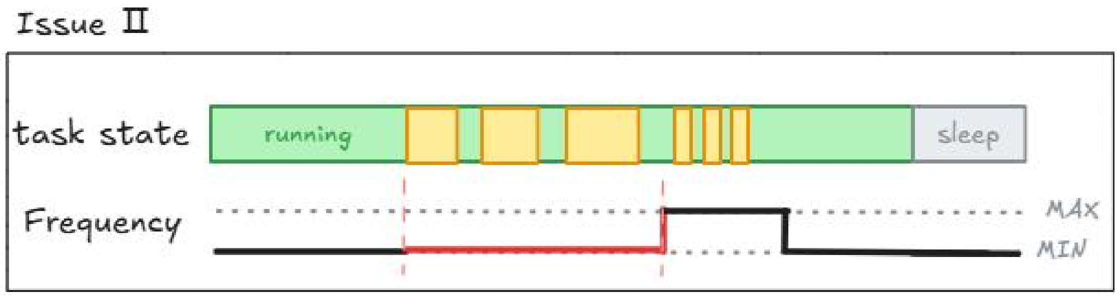
②由于Storage<u>动态调频</u>机制，当IO工作负载较低时，会切换至低频率。而敏感型线程的请求在低频率状态下完成，会引起IO耗时的增加

③<u>功耗状态切换的不确定性</u>，对IO性能表现带来极大的挑战



unning process:	<...> (pid 21459)
unning thread:	UnityMain
ate:	Uninterruptible Sleep - Block I/O





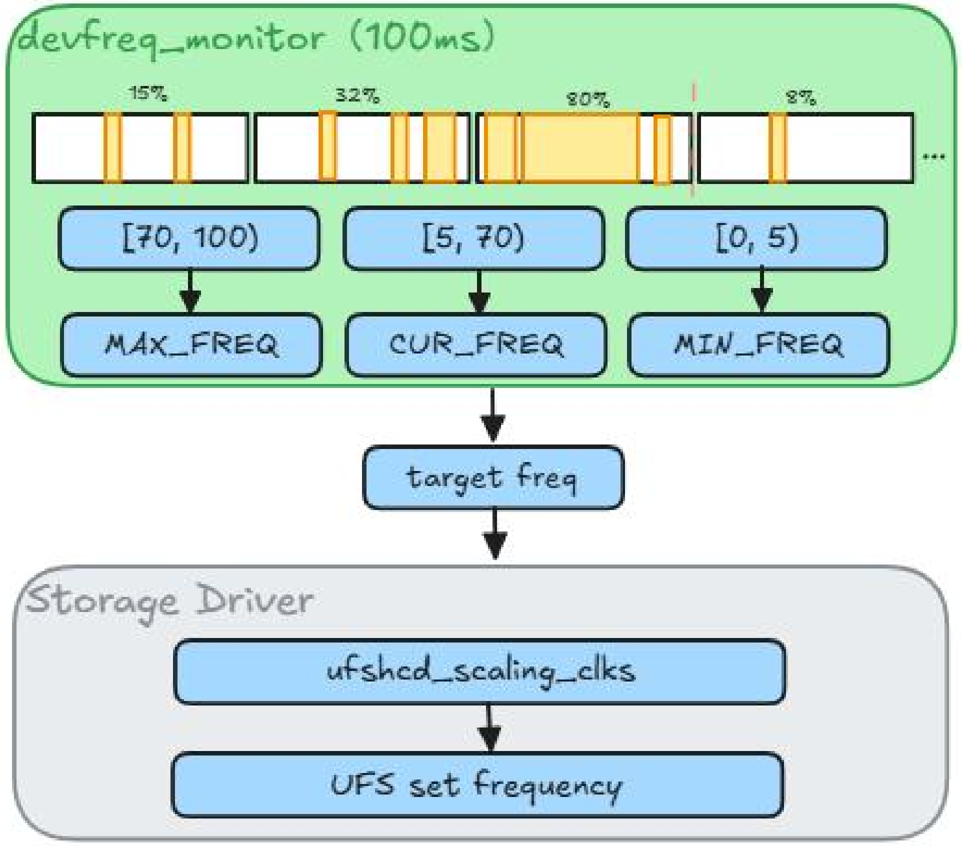
Part 2

方案探索

方案探索——动态调频的滞后性影响IO性能表现

原生动态调频方案是通过统计单位时间内IO负载为参考，根据给定的标准范围，设置存储设备期望的频率。

痛点：由于动态调频的滞后性，导致无法对瞬时高性能需求的场景匹配最优的器件性能，从而出现的影响IO性能的问题。

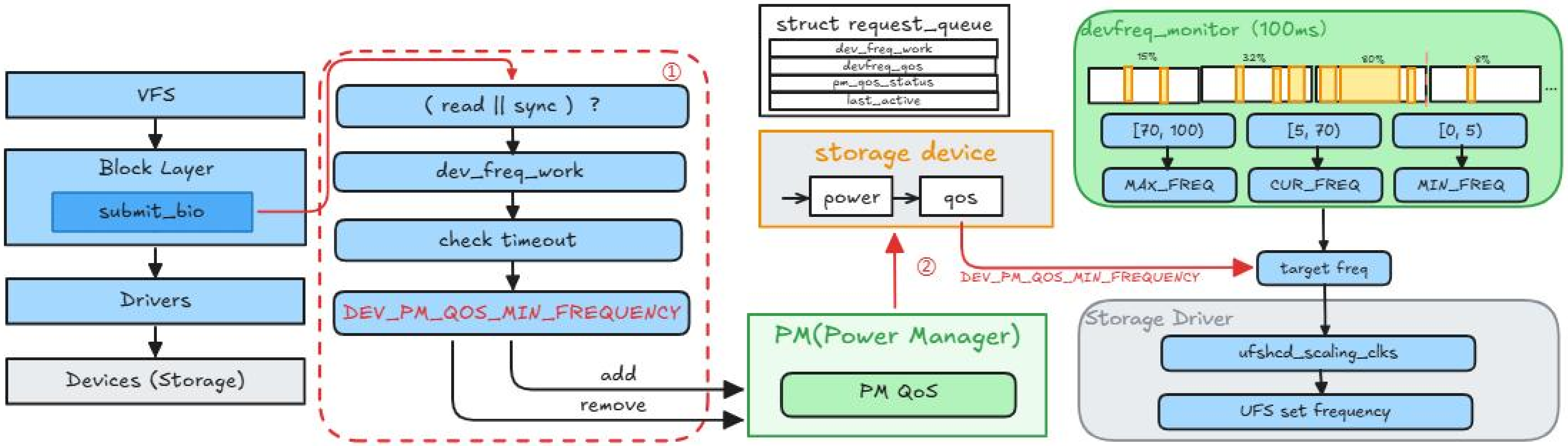


方案探索——Block层约束设备频率方案

针对动态调频机制痛点，提出在Block层中增加基于PM QoS的约束设备频率的方案。

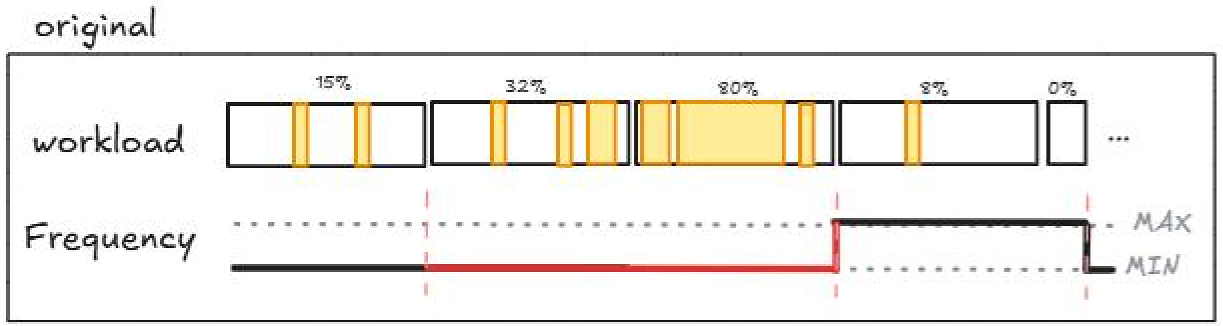
①在Block层触发IO请求时，可以通过 PM QoS 机制，预设对应 Block 设备中PM_QOS_MIN_FREQUENCY的约束值。

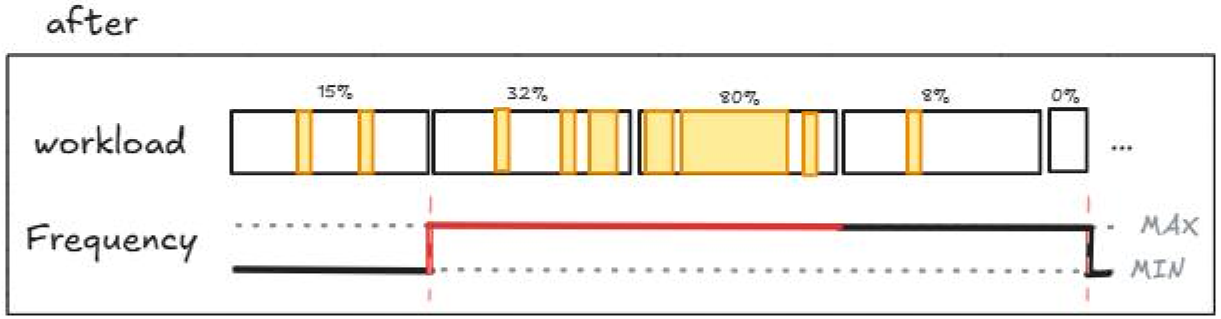
② 在原生的动态调频功能的基础上，增加对PM QoS中对频率质量指标的支撑，从而进一步优化频率调节的策略。



方案探索——目标达到最优的Storage性能配置

方案目标通过PM QoS质量频率的约束，最终实现优化工作负载检测的滞后性，并使得延迟敏感型线程的请求更容易工作在最优的存储性能配置。



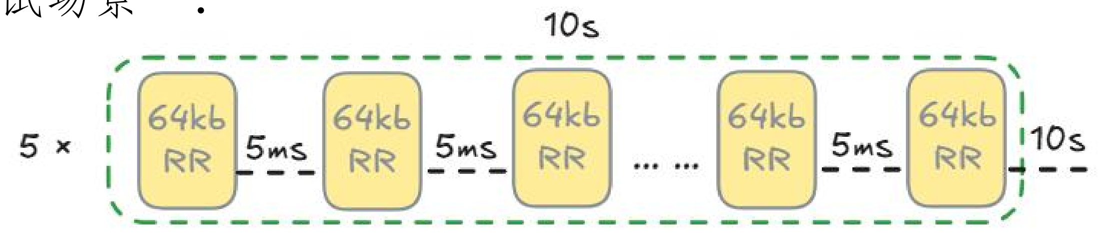


方案收益 —— 间歇IO负载场景中clat完成时延提升约15%

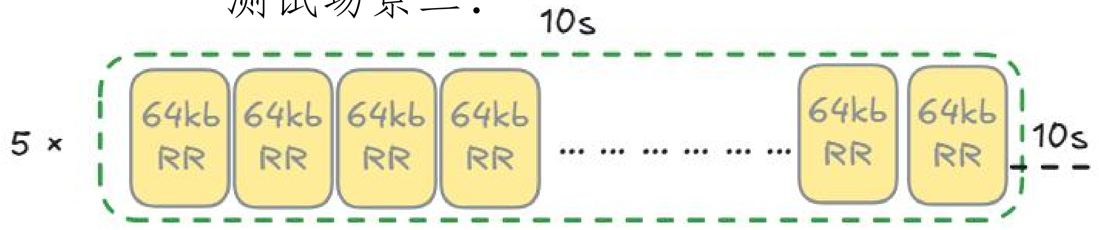
通过本地构建fio测试用例验证收益

测试参数：rw=randread, bs=(R) 64.0KiB, ioengine=libaio, iodepth=1, bound_cpu=3, run_time=10, direct=1

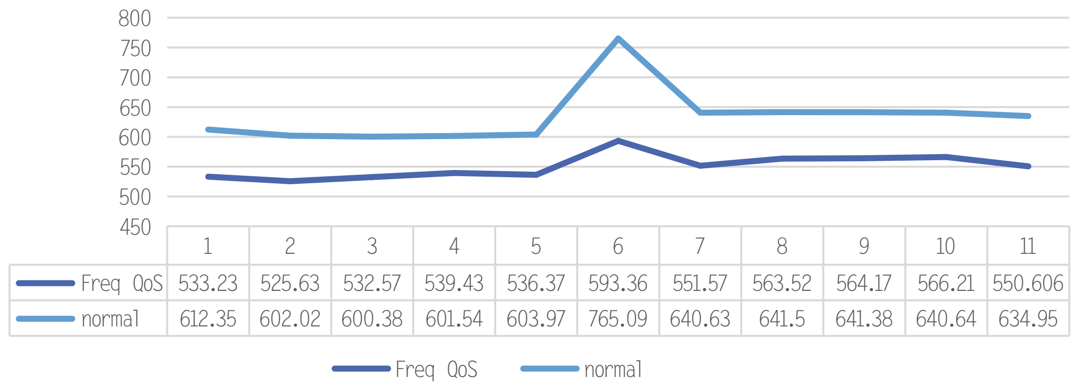
测试场景一：



测试场景二：

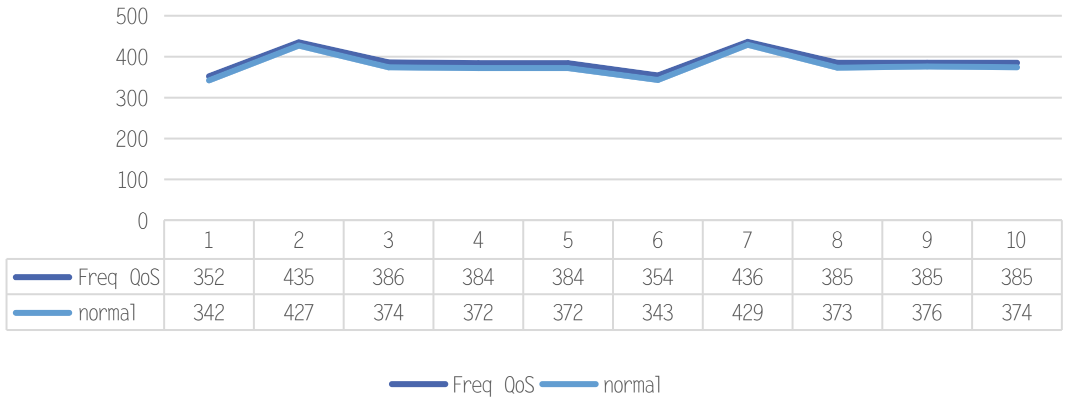


fio测试clat平均值



	1	2	3	4	5	6	7	8	9	10	11
Freq QoS	533.23	525.63	532.57	539.43	536.37	593.36	551.57	563.52	564.17	566.21	550.606
normal	612.35	602.02	600.38	601.54	603.97	765.09	640.63	641.5	641.38	640.64	634.95

fio测试结果bw数据



	1	2	3	4	5	6	7	8	9	10
Freq QoS	352	435	386	384	384	354	436	385	385	385
normal	342	427	374	372	372	343	429	373	376	374

Freq QoS方案经fio测试Clat（完成延迟）的平均值优化约13%，最小值优化约33%

Freq QoS方案经fio测试带宽bw平均值优化约10MB/s，在连续的IO测试中差异不大（约3%）

Part 3

挑战与展望

Block层约束频率方案正在与社区沟通

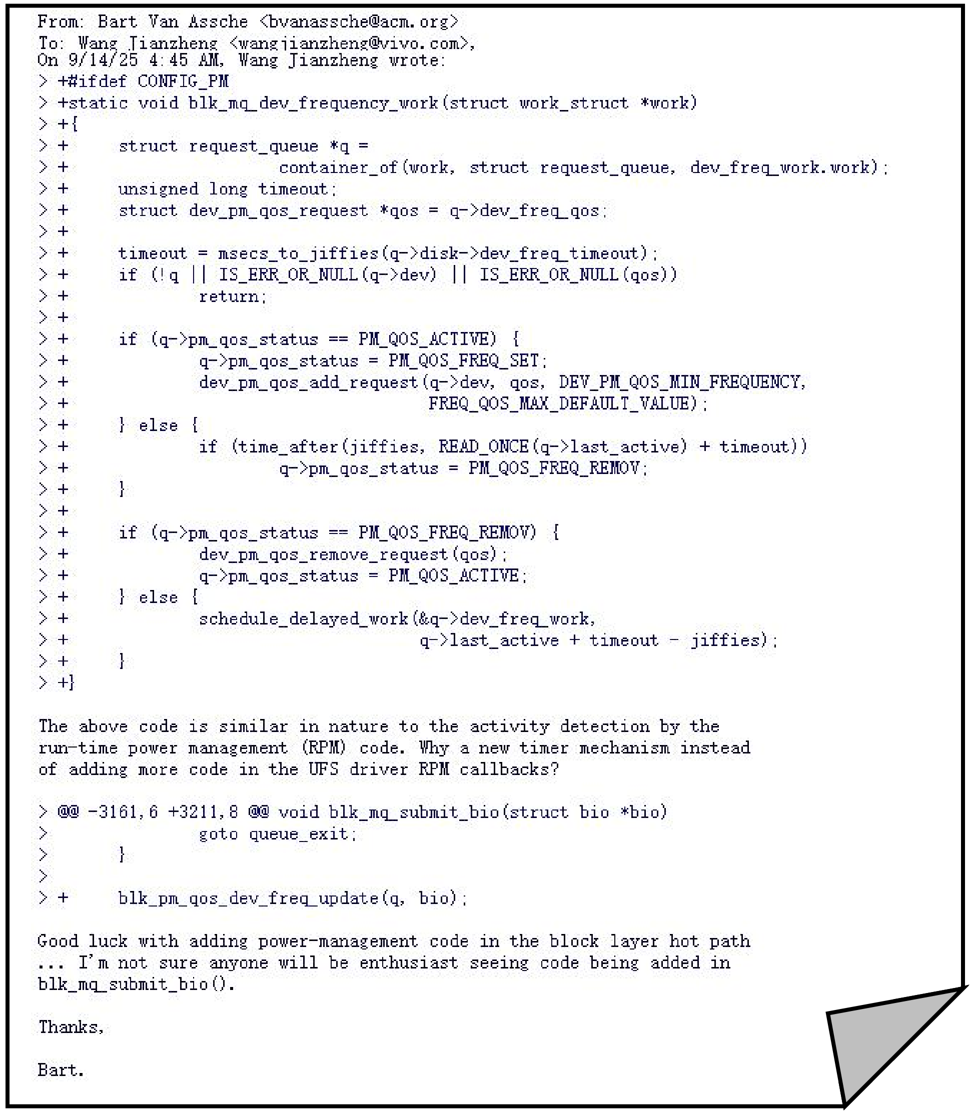
当前约束频率方案的patch已提交至社区，正处于沟通阶段

- 是否可以实现在RPM机制逻辑中
- Sys新增节点的必要性
- 精简代码

在后续测试中仍发现了一些问题

- 通过timeout实现的调度dev_freq_work，计时精度低导致存在频繁work调度

针对以上仍遗留的问题，会在后续的版本中陆续完善。

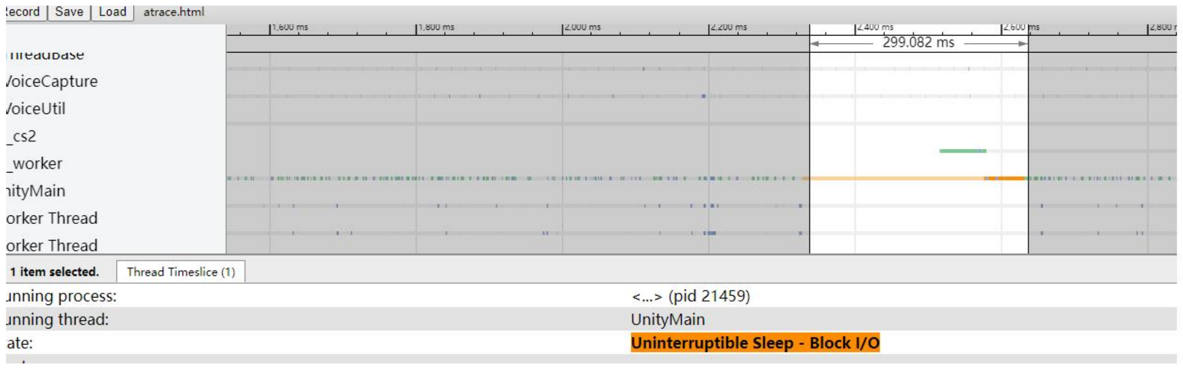


```
From: Bart Van Assche <bvanassche@acm.org>
To: Wang Jianzheng <wangjianzheng@vivo.com>,
On 9/14/25 4:45 AM, Wang Jianzheng wrote:
> +#ifdef CONFIG_PM
> +static void blk_mq_dev_frequency_work(struct work_struct *work)
> +{
> +       struct request_queue *q =
> +                       container_of(work, struct request_queue, dev_freq_work.work);
> +       unsigned long timeout;
> +       struct dev_pm_qos_request *qos = q->dev_freq_qos;
> +
> +       timeout = msecs_to_jiffies(q->disk->dev_freq_timeout);
> +       if (!q || IS_ERR_OR_NULL(q->dev) || IS_ERR_OR_NULL(qos))
> +               return;
> +
> +       if (q->pm_qos_status == PM_QOS_ACTIVE) {
> +               q->pm_qos_status = PM_QOS_FREQ_SET;
> +               dev_pm_qos_add_request(q->dev, qos, DEV_PM_QOS_MIN_FREQUENCY,
> +                                      FREQ_QOS_MAX_DEFAULT_VALUE);
> +       } else {
> +               if (time_after(jiffies, READ_ONCE(q->last_active) + timeout))
> +                       q->pm_qos_status = PM_QOS_FREQ_REMOV;
> +       }
> +
> +       if (q->pm_qos_status == PM_QOS_FREQ_REMOV) {
> +               dev_pm_qos_remove_request(qos);
> +               q->pm_qos_status = PM_QOS_ACTIVE;
> +       } else {
> +               schedule_delayed_work(&q->dev_freq_work,
> +                                     q->last_active + timeout - jiffies);
> +       }
> +}

The above code is similar in nature to the activity detection by the
run-time power management (RPM) code. Why a new timer mechanism instead
of adding more code in the UFS driver RPM callbacks?

> @@ -3161,6 +3211,8 @@ void blk_mq_submit_bio(struct bio *bio)
>               goto queue_exit;
>       }
>
> +     blk_pm_qos_dev_freq_update(q, bio);

Good luck with adding power-management code in the block layer hot path
... I'm not sure anyone will be enthusiast seeing code being added in
blk_mq_submit_bio().

Thanks,

Bart.
```

未来方向 —— 进一步探索RPM机制影响的IO性能问题

后续会进一步探索RPM机制影响的IO性能问题

- 探索RPM 机制resume操作会出现异常情况存在较大的切换耗时的原因
  - __synchronize_srcu
  - resume work runable
- 探索RPM 机制与PM QoS相结合，优化动态休眠唤醒的服务质量


ecord | Save | Load | atrace.html

1.600 ms 1.800 ms 2.000 ms 2.200 ms 2.400 ms 2.600 ms 2.800 ms

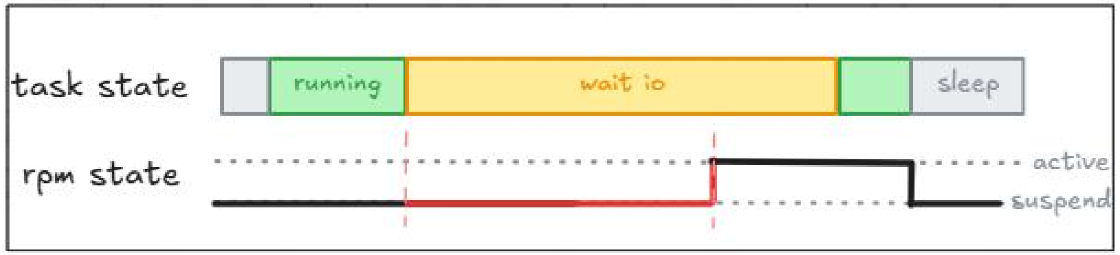
299.082 ms

hreadbase
VoiceCapture
VoiceUtil
_cs2
_worker
nityMain
orker Thread
orker Thread

1 item selected. Thread Timeslice (1)

unning process:	<...> (pid 21459)
unning thread:	UnityMain
ate:	Uninterruptible Sleep - Block I/O

Issue I



THANKS