CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

Defer throttle to when task exits to user

字节跳动-路自谦

CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

目录

CONTENTS

01 CPU带宽控制

A designer can use default text to simulate what text would look like.

02 现有机制的问题

A designer can use default text to simulate what text would look like.

03 基于任务的限流机制

A designer can use default text to simulate what text would look like.

04 结论

A designer can use default text to simulate what text would look like.

CPU带宽控制 - 简介

- 目标
  - 限定cpu资源的使用，但不限制在哪个cpu上运行
  - 比cpuset更灵活
- 应用场景
  - 容器资源限制：私有云业务里，限制每个业务的cpu资源
  - 后台任务管理： 限制后台批处理作业（如日志分析、数据备份）的 CPU 用量，保证前端交互式服务的响应速度。

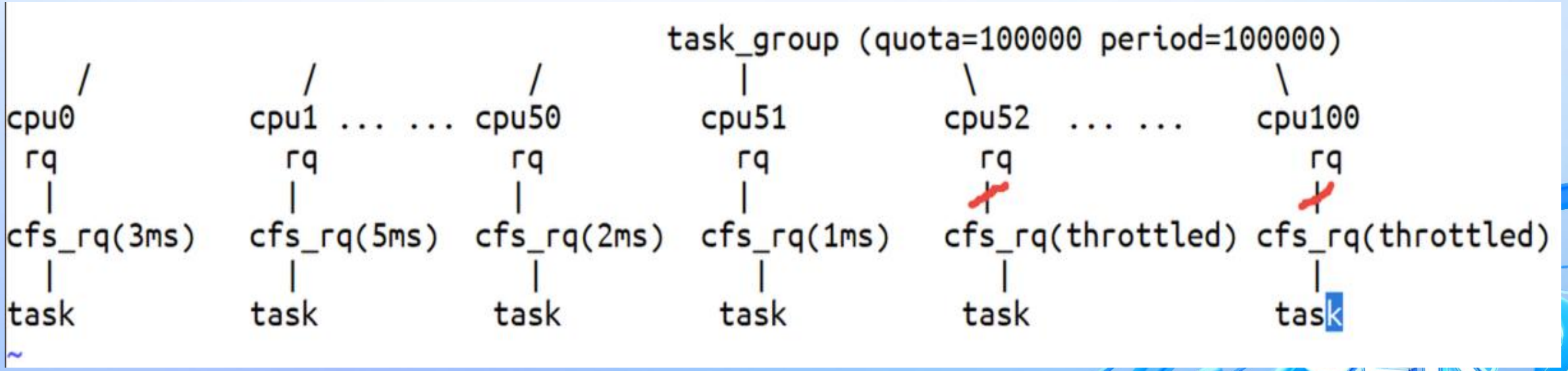
CPU带宽控制 - 接口

- 接口：
  - cgroup v2: cpu.max, cpu.stat
  - cgroup v1: cpu.cfs_quota_us cpu.cfs_period_us cpu.stat
- 例子
  - echo 500000 100000 > cpu.max
  - 在100ms的时间窗口最多使用500ms的cpu时间，等同于5c
  - cat cpu.stat

```
... ...
nr_periods 21107
nr_throttled 21098
throttled_usec 186436319106
```

CPU带宽控制 - 工作机制

- 此前内核限流机制
  - On throttle: dequeue cfs_rq
  - On unthrottle: enqueue cfs_rq



现有机制的问题

- 持锁的任务被唤醒后长时间无法运行
- 可导致整机无响应
- 例子：cgroup_threadgroup_rwsem
  - 需要在fork/exit的时候获取读锁
  - 需要移动任务进出cgroup时获取写锁
- 问题规避：避免配置太小的quota

现有机制的问题 – 例子

- cgroup_threadgroup_rwsem

基于任务的限流机制

- 推迟限流至任务返回用户态时
  - 限流时，标记cfs_rq为throttled状态，但并不将它从rq上摘除
  - 属于throttled cfs_rq的task在ret2user的时候，将自己从rq上dequeue(task work)
    - task在ret2user的时候，不持有任何内核资源
    - 这是唯一的throttle点
  - 新唤醒的任务：即使处于限流的cfs_rq里，也可以执行。等到ret2user的时候才会被dequeue

- 解除限流时将任务重新放回cfs_rq

基于任务的限流机制

- 限流 – 正常运行的任务

```
task execute in user
  -> tick
    -> account_cfs_rq_runtime()
      -> resched_curr()
  -> tick return: exit_to_user_mode_loop() a.k.a. ret2user
    -> schedule()
      -> pick_task_fair(): mark cfs_rq throttled, add throttle work for task
    -> throttle_cfs_rq_work()
      -> dequeue_task_fair()
      -> resched_curr()
    -> schedule(): pick another task
```

基于任务的限流机制

● 限流 – 新唤醒的任务

```
task wakes up in a throttled cfs_rq

cpuX:
wake_up_process(@p)
  -> enqueue_task_fair(@p) -> cpuY

cpuY:
schedule()
  -> pick_task_fair(): add throttle task work to @p
  -> switch_to(@p)

p starts executing:
  -> finish its work in kernel mode
  -> exit_to_user_mode_loop(): holds no kernel resource
    -> throttle_cfs_rq_work()
      -> dequeue_task_fair()
      -> resched_curr()
    -> schedule(): pick another task
```

基于任务的限流机制

- 限流 – 其他特殊情况

```
task group change, affinity change, etc.

      src                          dst

    cfs_rq(throttled)
       |
   task(throttled) -------> cfs_rq

dequeue from prev cfs_rq
enqueue to dst cfs_rq:
  - dst cfs_rq throttled? directly put on throttle list
  - dst cfs_rq not throttled? run normal
```

基于任务的限流机制

- 解除限流

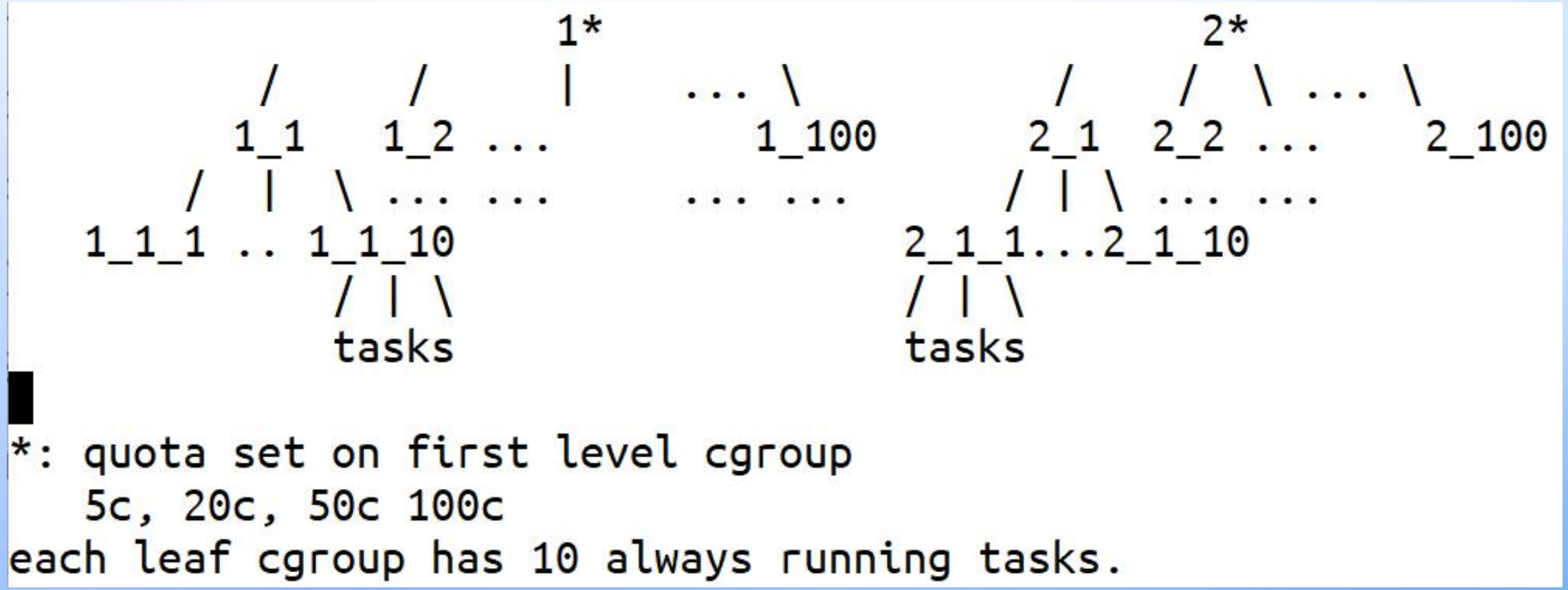
```
A new period begins:
sched_cfs_period_timer() // irqoff context
  -> distribute_cfs_runtime()
    -> send IPI to individual CPU: async unthrottle
      -> unthrottle_cfs_rq()
        -> tg_unthrottle_up()
          -> enqueue_task_fair()
    -> deal with local unthrottle
```

基于任务的限流机制 – 挑战1 – 性能

- 当一个cfs_rq上任务很多时
  - 限流时：
    - 之前的机制只需要将cfs_rq（对应的se）从rq上摘除(dequeue)
    - 现在需要将这些任务逐个摘除
  - 解除限流时：
    - 之前的机制只需要将cfs_rq（对应的se）放回rq即可
    - 现在需要将属于这个cfs_rq的task逐个enqueue
      - 解除限流的操作是在irqoff的场景下进行，可能会延长irqoff的时间
      - How bad it is?

基于任务的限流机制 – 挑战1 – 性能

- Test setup: 2sockets/384cpus AMD Genoa, 2000 cgroups, 每个cgroup有10个不停运行的任务，系统一共有2万个同时运行的任务。



基于任务的限流机制 – 挑战1 – 性能

- 最差的情况下，irqoff的时间大概是2-4ms。
- 大部分情况下，irqoff的时间在2ms以内。
- e.g. unthrottle 383rqs, 1101 cfs_rqs and 69 tasks took 1.6ms

```
distribute_cfs_runtime() during a 1 minute window:
@durations:
[8K, 16K)            274 |@@@@@@@@@@@@@@@@@@@@@                               |
[16K, 32K)           132 |@@@@@@@@@@                                          |
[32K, 64K)             6 |                                                    |
[64K, 128K)            0 |                                                    |
[128K, 256K)           2 |                                                    |
[256K, 512K)           0 |                                                    |
[512K, 1M)           117 |@@@@@@@@@                                           |
[1M, 2M)             665 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[2M, 4M)              10 |                                                    |
```

基于任务的限流机制 – 挑战1 – 性能

- 同样的测试，如果在不支持异步解除限流的内核上跑：unthrottled 384 rqs with a total of 422784 cfs_rqs and enqueued back 10000 tasks，耗时70ms.

```
@durations:
[4K, 8K)                1 |                                                    |
[8K, 16K)               8 |                                                    |
[16K, 32K)              1 |                                                    |
[32K, 64K)              0 |                                                    |
[64K, 128K)             0 |                                                    |
[128K, 256K)            0 |                                                    |
[256K, 512K)            0 |                                                    |
[512K, 1M)              0 |                                                    |
[1M, 2M)                0 |                                                    |
[2M, 4M)                0 |                                                    |
[4M, 8M)                0 |                                                    |
[8M, 16M)               0 |                                                    |
[16M, 32M)              0 |                                                    |
[32M, 64M)            376 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                        |
[64M, 128M)           824 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
```

基于任务的限流机制 – 挑战2 – 部分限流状态

- cfs_rq的partial throttle（部分限流）状态
  - 例如：cfs_rq有3个task，2个已被throttle/dequeue，还有1个跑在内核态
  - 需要统计cfs_rq的throttle时间吗
    - 只要有一个任务被throttle了就开始统计cfs_rq的throttle时间
  - 需要停止cfs_rq的PELT clock吗
    - 停止的话实现简单
    - 社区的人建议只要还有task在跑就不停止PELT clock

基于任务的限流机制 - 结论

- 可以缓解低优先级容器持锁后影响高优先级容器的问题
- 可以避免某些极端场景或配置错误场景下系统发生hung task的问题
- 可以解决实时内核的一些死锁问题
- 已合入Linux v6.18-rc1
  - https://lore.kernel.org/lkml/20250829081120.806-1-ziqianlu@bytedance.com/
  - https://lore.kernel.org/lkml/20250910095044.278-1-ziqianlu@bytedance.com/
- 欢迎报告问题！

THANKS