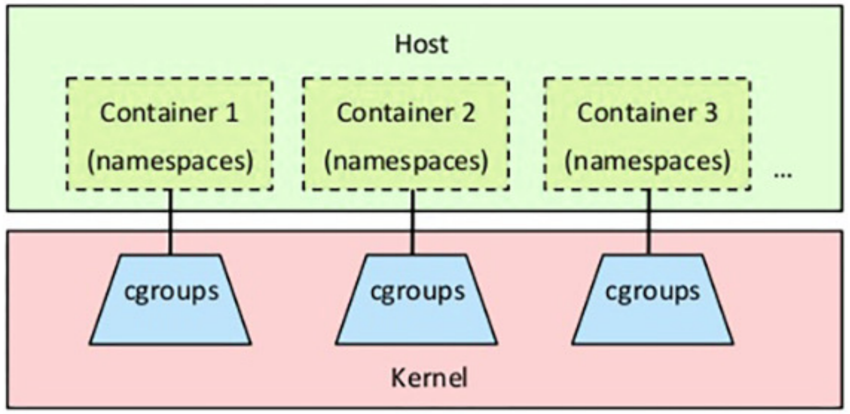
Challenges of using bandwidth controller in SMT systems

Shrikanth Hegde - IBM

CONTAINERS, CONTAINERS
EVERYWHERE

docker


Host
Container 1 (namespaces)
Container 2 (namespaces)
Container 3 (namespaces)
...
cgroups
cgroups
cgroups
Kernel

What is a bandwidth controller

- The bandwidth allowed for a group is specified using a quota and period. Within each given "period" (microseconds), a task group is allocated up to "quota" microseconds of CPU time. That quota is assigned to per-cpu run queues in slices as threads in the cgroup become runnable. Once all quota has been assigned any additional requests for quota will result in those threads being throttled. Throttled threads will not be able to run again until the next period when the quota is replenished.

- Implemented using hrtimer's, values are configurable through sysfs.

- Kubernetes does change the quota values based on the cpu allocated, but it never changes the period values. Its always set to default of 100ms.

Problem 1 – Cgroup's Align at throttled expiry

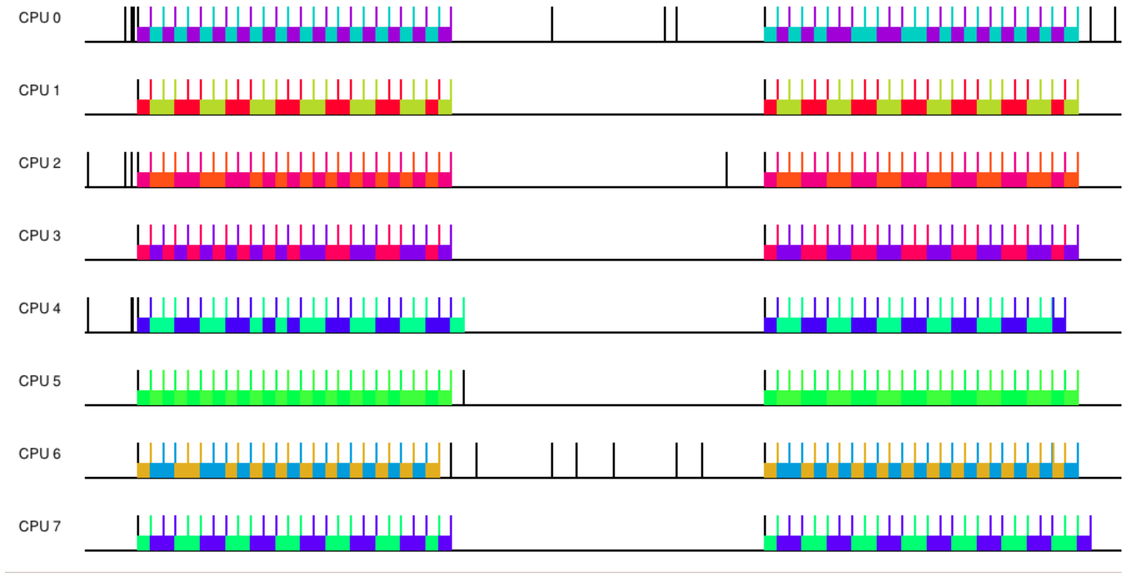
Each container typically has one Pod, roughly translating to a cgroup. (though it is done in heirarchy). Let's assume simpler case.

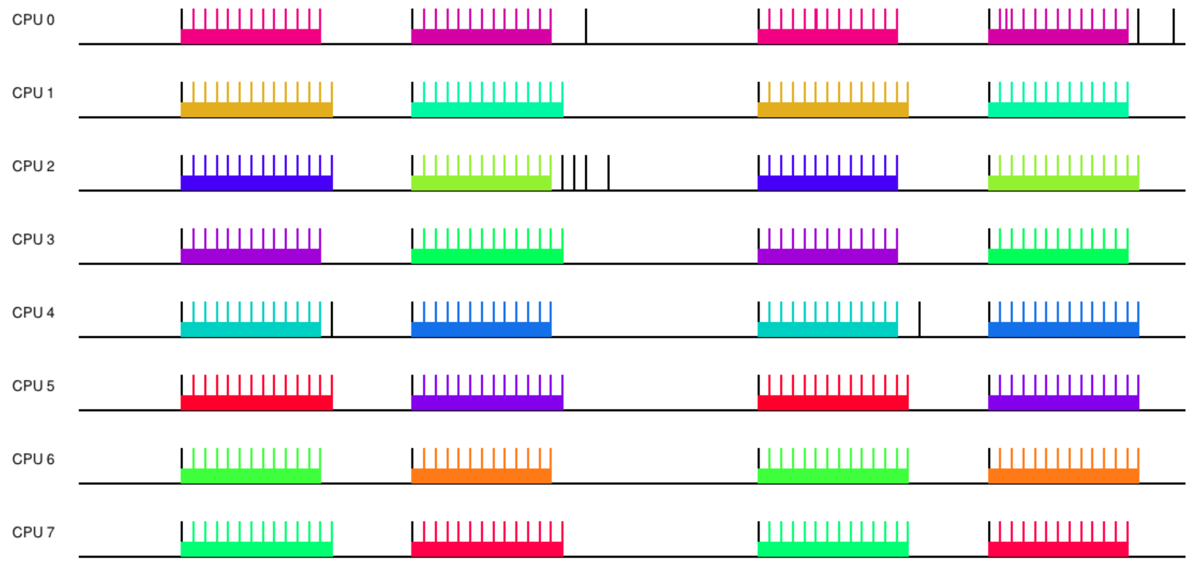
Each container/cgroup has a separate period timer. It is a hrtimer.

All the cgroup's period is same. Initial offset of all the timers is not set. I.e 0.

Hence all the period timers align at 100ms boundary when throttled.

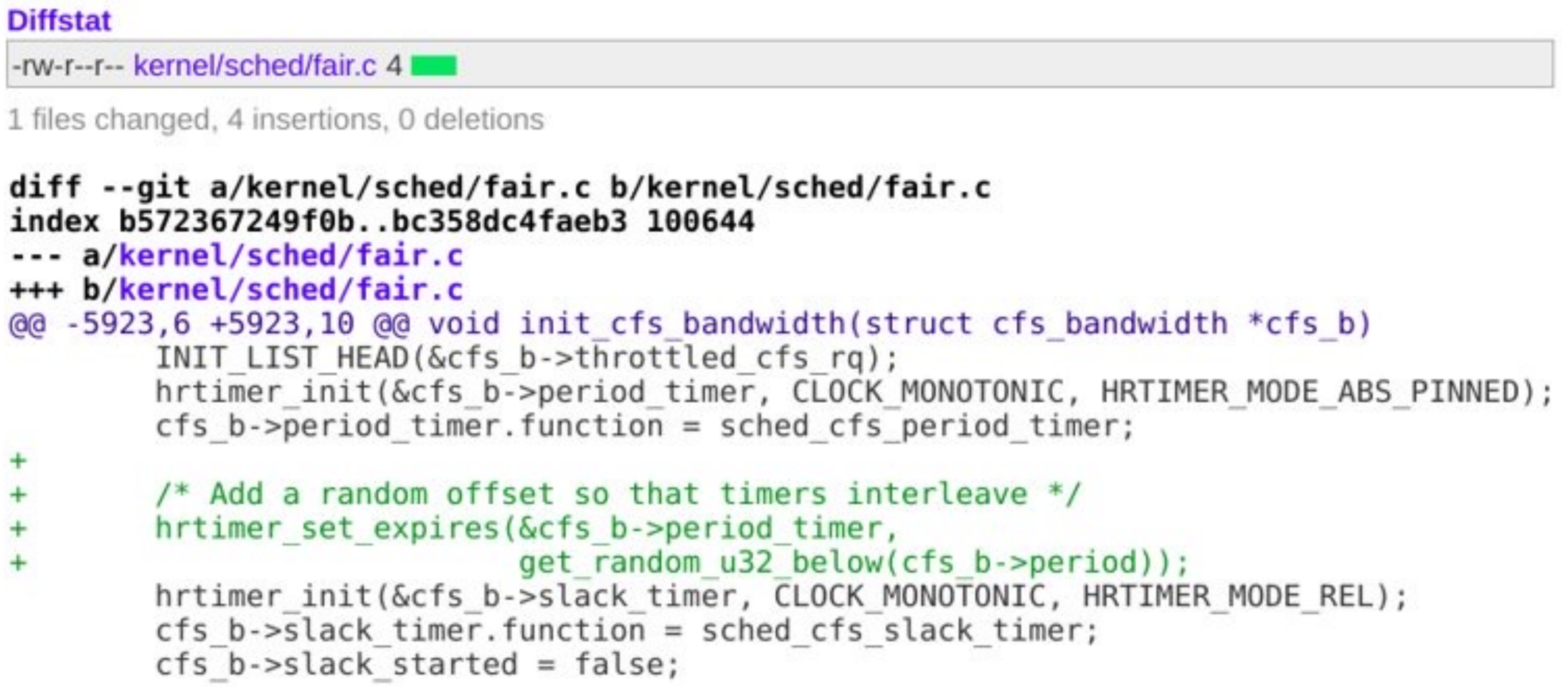
Results in higher SMT mode, lower IPS, more context switches, low throughput.





Solution: Interleave the timers with random initial offset.
A random number below the period value is taken and added to intial offset.

https://git.kernel.org/pub/scm/linux/kernel/git/tip/tip.git/commit/?h=sched/core&id=41abdba9374734b743019fc1cc05e3225c82ba6b



Diffstat

-rw-r--r--	kernel/sched/fair.c	4	🟩

1 files changed, 4 insertions, 0 deletions

```
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index b572367249f0b..bc358dc4faeb3 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -5923,6 +5923,10 @@ void init_cfs_bandwidth(struct cfs_bandwidth *cfs_b)
        INIT_LIST_HEAD(&cfs_b->throttled_cfs_rq);
        hrtimer_init(&cfs_b->period_timer, CLOCK_MONOTONIC, HRTIMER_MODE_ABS_PINNED);
        cfs_b->period_timer.function = sched_cfs_period_timer;
+
+       /* Add a random offset so that timers interleave */
+       hrtimer_set_expires(&cfs_b->period_timer,
+                           get_random_u32_below(cfs_b->period));
        hrtimer_init(&cfs_b->slack_timer, CLOCK_MONOTONIC, HRTIMER_MODE_REL);
        cfs_b->slack_timer.function = sched_cfs_slack_timer;
        cfs_b->slack_started = false;
```

Problem 2 – Performance hit at low utilization and high concurrency in SMT system

Perceived notion from a User is CPU time is in cycles and if same number of cycles given in different SMT modes it should result in same performance.
That is say SMT4 (100%) cycles and SMT8 (50%) cycles should yield similar throughput

User makes use of BW controller to enforce 50% limit to his containers. There is only one container (lets assume)

Performance drops by more than 20%

Reason: CPU when in lower SMT mode, have higher instructions per second. When the concurrency is high within the cgroup, moment its unthrottled, it occupies all the siblings, resulting in SMT8.

Potential Solution: WIP

Number of cpu: 4
Quota cpu: 2

Application
with 4 runnable threads

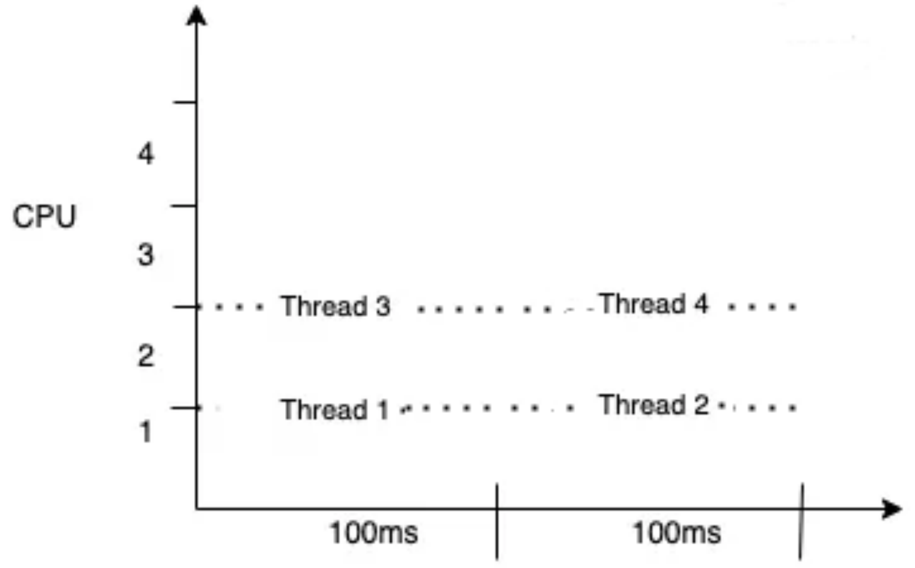


Figure 1.1

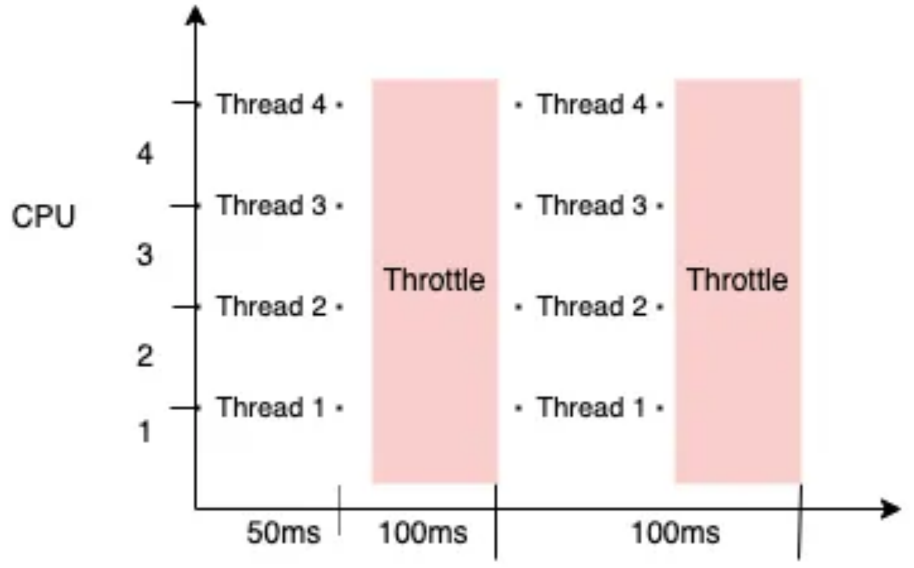


Figure 1.2

Q & A

Thank you

Legal Statement