CFS

A decidedly !RT scheduler

Peter Zijlstra – Red Hat

SCHED_OTHER

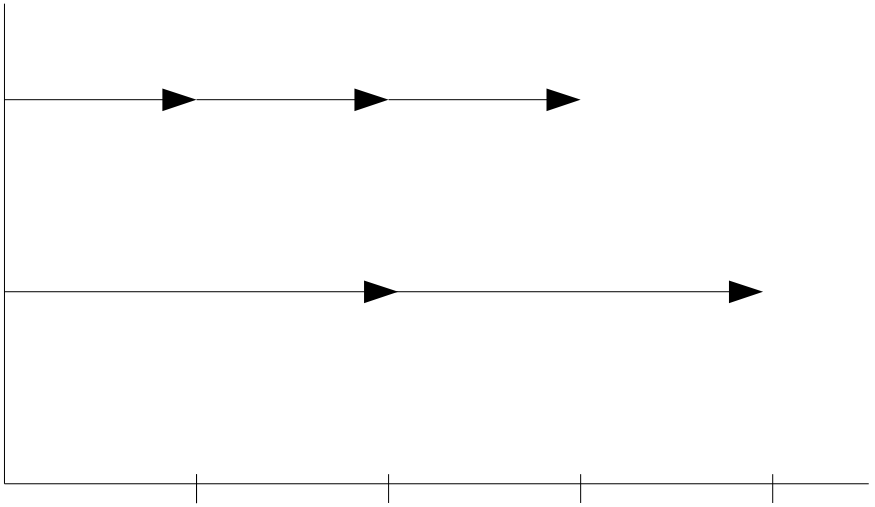
- Runs every 'normal' task on your machine
- Therefore no limit on the number of tasks
- Therefore no guarantees on timeliness
- 'optimized' for throughput

CFS

- The 'Completely' Fair Scheduler
  - As opposed to the old O(1) scheduler
  - Its 'fair' in that it doesn't allow starvation (much) and provides equal cpu time to equal tasks.

The basics

- WFQ based virtual time scheduler: $dv_i = \frac{dt_i}{w_i}$

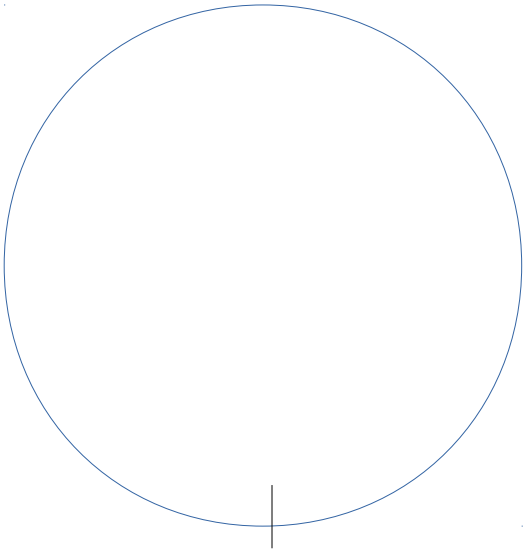


On time and limits

- We track time in nanoseconds
- We store time in u64
- That's ~585 years
- Except remember that division?

Overflows and min_vruntime

- Z mod 2^64 Z
- Undefined behaviour



Slices and preemption

- Run each task < human perception limit
- Overhead of slicing too fine

Task placement

- Lack of zero-lag point
- 'fair' sleepers
- Wakeup-preemption

Buddies

- Prefers running task other than the leftmost
  - Prev – run the task that ran before
  - Next – run the task that just got woken
  - Skip – do not run the task

SMP

- Work-conserving
- Smp-nice on overload

- Load-balancing:
  - Periodic 'slow' load-balance
  - Idle load-balance
  - New-idle load-balance

event balancing

- Exec/fork
- Wakeup (wake_affine/select_idle_siblings)
- Sleep (newidle)

Periodic balancing

- W_i/P_i == W_j/P_j
- Compute capacity P (irq,rt,etc.)
- Topology

```
*      log_2 n     1     n
*    \Sum       { --- * --- * 2^i } = O(n)                    (5)
*      i = 0      2^i   2^i
*                                 `- size of each group
*          |          |     `- number of cpus doing load-balance
*          |          `- freq
*          `- sum over all levels
```

cgroups

```
*   W_i = \Sum_j w_i,j                                                  (2)
*
* Where w_i,j is the weight of the j-th runnable task on cpu i. This weight
* is derived from the nice value as per prio_to_weight[].




* Cgroups make a horror show out of (2), instead of a simple sum we get:
*
*                                s_k,i
*   W_i = \Sum_j \Prod_k w_k * -----                                    (9)
*                               S_k
*
* Where
*
*   s_k,i = \Sum_j w_i,j,k  and  S_k = \Sum_i s_k,i                    (10)
*
* w_i,j,k is the weight of the j-th runnable task in the k-th cgroup on cpu i.
*
* The big problem is S_k, its a global sum needed to compute a local (W_i)
* property.
```

Per task accounting

- Runnable avg per task
- Per-cgroup aggregation
- Migrations
- Still hurts

Per cgroup bandwidth control

- Because no task limit;
- And proportional throughput
- Bandwidth limit is upper bound