Parametrizing CFS load balancing

nr_running / utilization / load

Dietmar Eggemann
<dietmar.eggemann@arm.com>

Agenda

- Introduction
- Overview existing CFS Load-Balancer Design
  - Used Statistics
  - Supported Features
  - Applied Heuristics
- Discussion - Changing existing CFS load-balancer to use either nr_running, utilization or load
  - Is it feasible at all ?
  - What has to be changed ?
  - How can this be achieved ?

Current Example from LKML

```
A load balancer calculates imbalance factor for particular shed
domain and tries to steal up the prescribed amount of weighted load.
However, a small imbalance factor would sometimes prevent us from
stealing any tasks at all. When a CPU is newly idle, it should
steal first task which passes a migration criteria.

---
 kernel/sched/fair.c | 13 ++++++++++--
 ...
@@ -6802,6 +6802,14 @@ static int detach_tasks(struct lb_env *env)
                if (env->idle != CPU_NOT_IDLE && env->src_rq->nr_running <= 1)
                        break;

+               /*
+                * Another CPU can place tasks, since we do not hold dst_rq lock
+                * while doing balancing. If newly idle CPU already got something,
+                * give up to reduce latency.
+                */
+               if (env->idle == CPU_NEWLY_IDLE && env->dst_rq->nr_running > 0)
+                       break;
+
                p = list_first_entry(tasks, struct task_struct, se.group_node);

                env->loop++;
@@ -6824,8 +6832,9 @@ static int detach_tasks(struct lb_env *env)
                if (sched_feat(LB_MIN) && load < 16 && !env->sd->nr_balance_failed)
                        goto next;

-               if ((load / 2) > env->imbalance)
-                       goto next;
+               if (env->idle != CPU_NEWLY_IDLE)
+                       if ((load / 2) > env->imbalance)
+                               goto next;

                detach_task(p, env);
```

Load balance path simplified)

load_balance()

find_busiest_group()

update_sd_lb_stats()

update_sg_lb_stats()

update_sd_pick_busiest()

/* Heuristics 1 */

calculate_imbalance()

/* Heuristics 2 */

fix_small_imbalance()

/* Heuristics 3 */

find_busiest_queue()

if (busiest->nr_running > 1)

cur_ld_moved = **detach_tasks()**

if (cur_ld_moved)

attach_tasks()

if (imb)

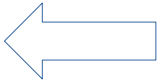
/* affinity handing , LBF_DST_PINNED , LBF_ALL_PINNED */

If (LBF_SOME_PINNED && imb > 0)

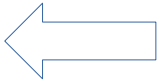
sd_parent→groups→sgc→imbalance = 1

If (!id_moved && need_active_balance())

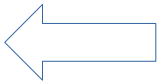
stop_one_cpu_nowait(…, active_load_balance_cpu_stop, ...)



Statistics gathering



How much load should be pulled



Pull load in form of tasks

Load balance input signals

- **Load** for entity (task)
  - PELT: sa.load_avg, runnable/blocked time for entity plus task priority
- **Runnable load** for cfs_rq
  - (PELT: cfs_rq->runnable_load_avg, runnable time of aggregated entities)
- **h_nr_running** (number runnable tasks)
- **Utilization** for cfs_rq (running/blocked time)
- **Cpu capacity**
- **Derived signals**
  - avg_load (runnable load / capacity)

Features

ASYM packing (Power 7/8, Turbo Boost 3)

- find_busiest_group()/update_sd_pick_busiest(): sg->asym_prefer_cpu

Prefer sibling (SMT)

- update_sd_lb_stats(): sgs → group_type = group_overloaded

Fbq (Find_Busiest_Queue) types

- update_sd_lb_stats(): env → fbq_type = fbq_classify_group()
- find_busiest_queue(): fbq_type rt= fbq_classify_rq(rq)

Affinity (sched_group→group_type == group_imbalanced)

- Set in load_balance() for domain / used in find_busiest_group() for domain->parent

Per-system state (root_domain->overload)

- Skip idle balance if not overloaded (more than one runnable task on any cpu)

Idle types

- (nohz idle(CPU_IDLE). Periodic (CPU_NOT_IDLE), idle balance (CPU_NEWLY_IDLE))

CONFIG_PREEMPT

- Break after first task is detached for CPU_NEWLY_IDLE to restrict latency

-> Introduce exceptions in the default avg_load based load balance handing

Statistics gathering

update_sg_lb_stats()

for_each_cpu(sched_group_cpus(group))

sgs→group_load += load **/* min/max(cfs_rq→runnable_load, rq→cpu_load[idx]) */**

sgs→group_util += cpu_util **/* min(rq.cfs.avg.util_avg), cpu_orig_capa */**

sgs→sum_nr_running += rq→cfs.h_nr_running

sgs→sum_weighted_load += weighted_cpuload **/* cfs_rq→runnable_load */**

sgs→group_capacity = group→sgc→capacity

sgs→avg_load = (sgs→group_load * 1024) / sgs->group_capacity

sgs→load_per_task = **sgs→sum_weighted_load** / sgs→sum_nr_running

sgs→group_no_capacity = group_is_overloaded()

sgs→group_type = group_classify()

update_sd_lb_stats()

sds→total_load += sgs→group_load

sds→total_capacity += sgs→group_capacity

if (child->flags & SD_PREFER_SIBLING) /* SMT */

sgs→group_type = group_overloaded

find_busiest_group()

sgs→avg_load = 1024 * sds.total_load / sds.total_capacity

Heuristics 1 - find_busiest_group()

```
/* 1. ASYM packing */
if (check_asym_packing()) -> return sds.busiest
/* 2. nothing to do */
(!sds.busiest || busiest->sum_nr_running == 0) -> out_balanced
/* 3. group classification -> bypass 'avg load' heuristics */
if (busiest == group_imbalanced) -> force_balance
/* 4. nr_running & utilization/capacity */
if (CPU_NEWLY_IDLE && group_has_capacity(l) && b->group_no_capacity) -> force_balance
/* 5. avg load based */
if (local->avg_load >= busiest->avg_load) -> out_balanced
/* 6. do not create more idle cpus */
if (CPU_IDLE && b != group_overloaded && l->idle_cpus <= b->idle_cpus + 1) -> out_balanced
/* 7. compare avg load with imbalance factor */
if (!CPU_IDLE && 100 * busiest->avg_load <= imbalance_pct * local->avg_load) -> out_balanced
```

Heuristics 2 - calculate_imbalance()

```
/* 1. cannot rely on group-wide averages */
if (busiest->group_type == group_imbalanced)
        busiest->load_per_task = min(busiest->load_per_task, sds->avg_load)
/* 2. because of capacity and skipping of sg's w/ smaller group_type*/
if (busiest->avg_load <= sds->avg_load || local->avg_load >= sds->avg_load)
        fix_small_imbalance()
/* 3. avoid creating idle cpus */
if ({b/l}->group_type == overloaded && b->nr_running * 1024 > b→capa)
        load_above_capa = (b->nr_running * 1024 – b->capa)*1024/b->capa
/* 4. trying to get all the cpus to the average_load */
/*    do not push local above and busiest below avg_load */
/*    do not reduce group load below group capacity */
imb = min(min(diff_avg_ld(b,sds), load_above_capa)*b→cap, diff_avg_ld(sds,l)*l→capa))
/* 5. no guarantee that any task will be suitable */
if (imb < busiest->load_per_task)
        fix_small_imbalance()
```

Heuristics 3 - fix_small_imbalance()

```
/* determine if moving one task over is beneficial */
/* meaning of one task has changed: smpnice & fine grained load accounting */
imbn = (l→sum_nr_running && (b->load_per_task > l->load_per_task)) ? 1 : 2
/* 1. b→avg_load is scaled_busy_load_per_task greater than l->avg_load
if (b->avg_load + (b->load_per_task * 1024/b->capa) >= l->avg_load +
        imbn * (b->load_per_task * 1024/b->capa))
return (imb = busiest->load_per_task)
/* 2. not enough imb to move tasks, try to increase total cpu capacity */
/*    move if throughput increases */
capa_now = …
capa_move = …
if (capa_move > capa_now)
imb = busiest->load_per_task
```

Summary of existing code

- Statistics gathering of all three input signals at the same time
- Heuristics use all three input signals
- Intermingled with Feature support
- Influence of former input signals noticeable (fix_small_imbalance())

How to do a parametrized approach?

(1) Order (nr_running -> utilization -> load)?

(2) Which parameter to use when?

–Static mapping to domain (i.e. topology flags) or

–sds->sum_nr_running <= sd→span_weight (needs statistics gathering first !)

–root_domain->over-utilized (EAS, Tipping Point))

(3) Difference between nr_running and utilization

–sds->sum_nr_running <= sd->span_weight

(4) Difference between utilization and load

–Over the Tipping Point, influence of priority (SMPnice)

How to do a parametrized approach?

(5) (nr_running * 1024) as parameter

- Would allow to have ${param}_per_task and avg_${param}

(6) avg_${param} for dependeny to capacity

(7) Heuristics have to be parametrized as well

- Only use one sort of ${param} , ${param}_per_task and avg_${param}

(8) Helper functions like group_is_overloaded() might go away

(9) Instability due to system running near the boundary b/w 2 params?

(10) Has nothing to do with EAS (best cpu for task)

- But EAS is not about balancing in the first place
- Force balancing big task onto big cpu (misfit task)

How to do a parametrized approach?

(11) Make load balance code easier

- remove rq->cpu_load[idx] array
- replace it with imbalance_pct ?
- Remove fix_small_imbalance()