Workqueues and cpu hotplug

冯博群 Boqun Feng
boqun.feng@gmail.com

Background story

- One day Paul wanted to parallelize expedited grace-period initialization
  - commit 25f3d7effab63

```
...
rcu_for_each_leaf_node(rnp) {
    ...
    INIT_WOKR(&rnp->rew.rew_work, ..);
    queue_work_on(rnp->grplo, rcu_par_gp_wq, &rnp->rew.rew_work);
    rnp->exp_need_flush = true;
}

rcu_for_each_leaf_node(rnp)
    if (rnp->exp_need_flush)
        flush_work(&rnp->rew.rew_work);
```

Background story (cont.)

- But we hit this:

```
> BUG: workqueue lockup - pool cpus=0 node=0 flags=0x4 nice=0 stuck for 59s!
> Showing busy workqueues and worker pools:
> workqueue rcu_gp: flags=0x8
>   pwq 22: cpus=11 node=0 flags=0x0 nice=0 active=1/256
>     in-flight: 28:wait_rcu_exp_gp
> workqueue rcu_par_gp: flags=0x8
>   pwq 0: cpus=0 node=0 flags=0x4 nice=0 active=1/256
```

flags=0x4 means `POOL_DISASSOCIATED`, which means pwq 0 is offline

What happen?

```
/**
 * queue_work_on - queue work on specific cpu
 * @cpu: CPU number to execute work on
 * @wq: workqueue to use
 * @work: work to queue
 *
 * We queue the work to a specific CPU,  the caller must ensure it
 * can't go away.
 *
 * Return: %false if @work was already on a queue, %true otherwise.
 */
bool queue_work_on(int cpu, struct workqueue_struct *wq,
            struct work_struct *work)
```

What happens? (cont.)

rnp->grplo is already offlined when we try to queue the work

```
...
rcu_for_each_leaf_node(rnp) {
    ...
    INIT_WOKR(&rnp->rew.rew_work, ..);
    queue_work_on(rnp->grplo, rcu_par_gp_wq, &rnp->rew.rew_work);
    rnp->exp_need_flush = true;
}

rcu_for_each_leaf_node(rnp)
    if (rnp->exp_need_flush)
        flush_work(&rnp->rew.rew_work);
```

Solution

```
...
rcu_for_each_leaf_node(rnp) {
    ...
    INIT_WOKR(&rnp->rew.rew_work, ..);
    preempt_disable();
    cpu = cpumask_next(rnp->grplo - 1, cpu_online_mask);
    /* If all offline, queue the work on an unbound CPU. */
    if (unlikely(cpu > rnp->grphi))
        cpu = WORK_CPU_UNBOUND;
    queue_work_on(cpu, rcu_par_gp_wq, &rnp->rew.rew_work);
    preempt_enable();
    rnp->exp_need_flush = true;
}

rcu_for_each_leaf_node(rnp)
    if (rnp->exp_need_flush)
        flush_work(&rnp->rew.rew_work);
```

Better solution?

- Limitation of current workqueue API
  - per-cpu workqueue allow to run work items in parallel, but need to deal with cpu hotplug when queue_work_on().
  - unbound workqueue only provide the parallel level the same as the numbers of NUMA node.
- Ideally we want the ability to:
  - Run N (N > # of NUMA nodes) work items in parallel or,
  - For each fine-grained group of CPUs (smaller than a NUMA node, e.g. rcu_node), run a work item in parallel
  - and need no worry for racing with cpu hotplug.

Possible solution #1

- Allow to queue a work item on a offline cpu in per-cpu workqueue
  - having some mechanism to steal/grab work item from a worker pool if the cpu is offlined.
- Pros
  - No need to introduce another workqueue API
- Cons
  - Increase the complexity of work item processing
  - Will it work well with load balance?

Possible solution #2

- Generalize numa_pwq to support more fine-grained node.
  - Modify alloc_workqueue() to allocate pwqs more than # of NUMA.
  - each workqueue has its own cpu_to_node()
  - a slightly different wq_calc_node_cpumask()
  - also need to handle cpu hotplug differently

Possible solution #3

- Solve this in another layer higher than workqueue