

Re: [rfc] [patch] mm: Slab – Eliminate lock_cpu_hotplug from slab

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-10/msg10200.html>

- *From:* Srivatsa Vaddagiri <vatsa@xxxxxxxxxx>
 - *Date:* Tue, 31 Oct 2006 09:51:21 +0530
-

On Mon, Oct 30, 2006 at 07:51:40PM -0800, Ravikiran G Thirumalai wrote:

kmem_cache_shrink:

From what I can gather by looking at the cpu hotplug code, disabling preemption before iterating over `cpu_online_map` ensures that a cpu won't disappear from the bitmask (system). But it does not ensure that a cpu won't come up right? (I see `stop_machine` usage in the `cpu_down` path, but not in the `cpu_up` path). But then on closer look I see that `on_each_cpu` uses `call_lock` to protect the `cpu_online_map` against `cpu_online` events. So, yes we don't need to take the `cache_chain_sem` here.

Yes thats what I thought.

kmem_cache_destroy:

We still need to stay serialized against cpu online here. I guess you already know why :)
<http://lkml.org/lkml/2004/3/23/80>

sure!

Maybe I am missing something, but what prevents someone from reading the wrong `tsk->cpus_allowed` at (A) below?

```
static int _cpu_down(unsigned int cpu)
{
...
...
set_cpus_allowed(current, tmp);
----- (A)
```

Re: [rfc] [patch] mm: Slab – Eliminate lock_cpu_hotplug from slab

lock_cpu_hotplug() in sched_getaffinity was supposed to guard from reading the wrong value you point out. But with recent churn of cpu hotplug locking, this is broken now.

Ideally, lock_cpu_hotplug() should have taken something equivalent to cpu_add_remove_lock (breaking cpufreq in the course :), but moving mutex_lock(&cpu_bitmask_lock) few lines above (before set_cpus_allowed) should also work in this case.

Alternately, taking a per-subsystem lock in DOWN_PREPARE/LOCK_ACQUIRE notifications, which is used by sched_getaffinity also, would work (as you note below).

```
mutex_lock(&cpu_bitmask_lock);
p = __stop_machine_run(take_cpu_down, NULL, cpu);
...
}
```

If we are discarding this whole lock_cpu_hotplug(), then IMO, we should use LOCK_ACQUIRE/RELEASE, where ACQUIRE notification is sent **before** messing with tsk->cpus_allowed and RELEASE notification sent **after** restoring tsk->cpus_allowed (something like below):

```
@@ -186,13 +186,14 @@ int cpu_down(unsigned int cpu)
{
int err = 0;

- mutex_lock(&cpu_add_remove_lock);
+ blocking_notifier_call_chain(&cpu_chain, CPU_LOCK_ACQUIRE,
+ (void *) (long)cpu);
if (cpu_hotplug_disabled)
err = -EBUSY;
else
err = _cpu_down(cpu);
-
- mutex_unlock(&cpu_add_remove_lock);
+ blocking_notifier_call_chain(&cpu_chain, CPU_LOCK_RELEASE,
+ (void *) (long)cpu);
return err;
}
```

But, since we send CPU_DOWN_PREPARE at _cpu_down before set_cpus_allowed(), is it not possible to take the per scheduler subsystem lock at DOWN_PREPARE and serialize sched_getaffinity with the same per scheduler subsys lock?

Re: [rfc] [patch] mm: Slab – Eliminate lock_cpu_hotplug from slab

CPU_DOWN_PREPARE is a good enough time to acquire the (scheduler subsystem) lock. But I am more concerned about when we release that lock. Releasing at CPU_DEAD/CPU_DOWN_FAILED is too early, since the tasks's cpus_allowed mask would not have been restored by then. That's why having a separate notification to release (and acquire) the lock would make more sense I thought.

--

Regards,
vatsa

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>