Subject: Re: [PATCH v2 02/13] memcg: Kernel memory accounting infrastructure.
Posted by Glauber Costa on Tue, 13 Mar 2012 10:37:30 GMT
View Forum Message <> Reply to Message

> After looking codes, I think we need to think
> whether independent_kmem_limit is good or not....
>
> How about adding MEMCG_KMEM_ACCOUNT flag instead of this and use only
> memcg->res/memcg->memsw rather than adding a new counter, memcg->kmem ?
>
> if MEMCG_KMEM_ACCOUNT is set -> slab is accoutned to mem->res/memsw.
> if MEMCG_KMEM_ACCOUNT is not set -> slab is never accounted.
>
> (I think On/Off switch is required..)
>
> Thanks,
> -Kame
>

This has been discussed before, I can probably find it in the archives
if you want to go back and see it.

But in a nutshell:

1) Supposing independent knob disappear (I will explain in item 2 why I
don't want it to), I don't thing a flag makes sense either. *If* we are
planning to enable/disable this, it might make more sense to put some
work on it, and allow particular slabs to be enabled/disabled by writing
to memory.kmem.slabinfo (-* would disable all, +* enable all, +kmalloc*
enable all kmalloc, etc).

Alternatively, what we could do instead, is something similar to what
ended up being done for tcp, by request of the network people: if you
never touch the limit file, don't bother with it at all, and simply does
not account. With Suleiman's lazy allocation infrastructure, that should
actually be trivial. And then again, a flag is not necessary, because
writing to the limit file does the job, and also convey the meaning well
enough.

2) For the kernel itself, we are mostly concerned that a malicious
container may pin into memory big amounts of kernel memory which is,
ultimately, unreclaimable. In particular, with overcommit allowed
scenarios, you can fill the whole physical memory (or at least a
significant part) with those objects, well beyond your softlimit
allowance, making the creation of further containers impossible.
With user memory, you can reclaim the cgroup back to its place. With
kernel memory, you can't.

In the particular example of 32-bit boxes, you can easily fill up a large part of the available 1gb kernel memory with pinned memory and render the whole system unresponsive.

Never allowing the kernel memory to go beyond the soft limit was one of the proposed alternatives. However, it may force you to establish a soft limit where one was not previously needed. Or, establish a low soft limit when you really need a bigger one.

All that said, while reading your message, thinking a bit, the following crossed my mind:

- We can account the slabs to memcg->res normally, and just store the information that this is kernel memory into a percpu counter, as I proposed recently.
- The knob goes away, and becomes implicit: if you ever write anything to memory.kmem.limit_in_bytes, we transfer that memory to a separate kmem res_counter, and proceed from there. We can keep accounting to memcg->res anyway, just that kernel memory will now have a separate limit.
- With this scheme, it may not be necessary to ever have a file memory.kmem.soft_limit_in_bytes. Reclaim is always part of the normal memcg reclaim.

The outlined above would work for us, and make the whole scheme simpler, I believe.

What do you think ?
