Subject: Fork bomb limitation in memcg WAS: Re: [PATCH 00/11] kmem controller for memcg: stripped down versio
Posted by Glauber Costa on Wed, 27 Jun 2012 09:29:04 GMT
View Forum Message <> Reply to Message

On 06/27/2012 01:55 AM, Andrew Morton wrote:
> On Tue, 26 Jun 2012 11:17:49 +0400
> Glauber Costa <glommer@parallels.com> wrote:
>
>> On 06/26/2012 03:27 AM, Andrew Morton wrote:
>>> On Mon, 25 Jun 2012 18:15:17 +0400
>>> Glauber Costa <glommer@parallels.com> wrote:
>>>
>>>> What I am proposing with this series is a stripped down version of the
>>>> kmem controller for memcg that would allow us to merge significant parts
>>>> of the infrastructure, while leaving out, for now, the polemic bits about
>>>> the slab while it is being reworked by Cristoph.
>>>>
>>>> Me reasoning for that is that after the last change to introduce a gfp
>>>> flag to mark kernel allocations, it became clear to me that tracking other
>>>> resources like the stack would then follow extremely naturaly. I figured
>>>> that at some point we'd have to solve the issue pointed by David, and avoid
>>>> testing the Slab flag in the page allocator, since it would soon be made
>>>> more generic. I do that by having the callers to explicit mark it.
>>>>
>>>> So to demonstrate how it would work, I am introducing a stack tracker here,
>>>> that is already a functionality per-se: it successfully stops fork bombs to
>>>> happen. (Sorry for doing all your work, Frederic =p). Note that after all
>>>> memcg infrastructure is deployed, it becomes very easy to track anything.
>>>> The last patch of this series is extremely simple.
>>>>
>>>> The infrastructure is exactly the same we had in memcg, but stripped down
>>>> of the slab parts. And because what we have after those patches is a feature
>>>> per-se, I think it could be considered for merging.
>>>
>>> hm. None of this new code makes the kernel smaller, faster, easier to
>>> understand or more fun to read!
>> Not sure if this is a general comment - in case I agree - or if targeted
>> to my statement that this is "stripped down". If so, it is of course
>> smaller relative to my previous slab accounting patches.
>
> It's a general comment. The patch adds overhead: runtime costs and
> maintenance costs. Do its benefits justify that cost?

Despite potential disagreement on the following statement from my peer
and friends,
I am not crazy. This means I of course believe so =)

I will lay down my views below, but I believe the general justification
was given already by people commenting in the task counter subsystem,
and I invite them (newly CCd) to chime in here if they want.

For whoever is arriving in the thread now, this is about limiting the
amount of kernel memory used by a set of processes, aka cgroup.

In this particular state - more is to follow - we're able to limit
fork bombs in a clean way, since overlimit processes will be unable
to allocate their stack.

>> The infrastructure is largely common, but I realized that a future user,
>> tracking the stack, would be a lot simpler and could be done first.
>>
>>> Presumably we're getting some benefit for all the downside. When the
>>> time is appropriate, please do put some time into explaining that
>>> benefit, so that others can agree that it is a worthwhile tradeoff.
>>>
>>
>> Well, for one thing, we stop fork bombs for processes inside cgroups.
>
> "inside cgroups" is a significant limitation! Is this capability
> important enough to justify adding the new code? That's unobvious
> (to me).

Again, for one thing. The general mechanism here is to limit the amount
of kernel memory used by a particular set of processes. Processes use
stack, and if they can't grab more pages for the stack, no new
processes can be created.

But there are other things the general mechanism protects against.

Using too much of pinned dentry and inode cache, by touching files
an leaving them in memory forever.

In fact, a simple:

while true; do mkdir x; cd x; done

can halt your system easily, because the file system limits are hard
to reach (big disks), but the kernel memory is not.

Those are examples, but the list certainly don't stop here.

Our particular use case is concerned with people offering hosting
services. In a physical box, we can put a limit to some resources,
like total number of processes or threads. But in an environment
where each independent user gets its own piece of the machine, we

don't want a potentially malicious user to destroy good users' services

This might be true for systemd as well, that now groups services inside cgroups. They generally want to put forward a set of guarantees that limits the running service in a variety of ways, so that if they become badly behaved, they won't interfere with the rest of the system.

fork bombs are a way bad behaved processes interfere with the rest of the system. In here, I propose fork bomb stopping as a natural consequence of the fact that the amount of kernel memory can be limited, and each process uses 1 or 2 pages for the stack, that are freed when the process goes away.

The limitation "inside cgroups" is not as important as you seem to state. Everything can be put "inside cgroups". Being inside cgroups is just a way to state "I want this feature". And in this case: I want to pay the price.

Yes, because as you said yourself, of course there is a cost for that. But I am extensively using static branches to make sure that even if the feature is compiled in, even if we have a bunch of memory cgroups deployed (that already pay a price for that), this code will only be enabled after the first user of this service configures any limit.

The impact before that, is as low as it can be.

After that is enabled, processes living outside of limited cgroups will perform an inline test for a flag, and continue normal page allocation with only that price paid: a flag test.

> Are there any other user-facing things which we can do with this
> feature? Present, or planned?
>

Yes. We can establish all sorts of boundaries to a group of processes. The dentry / icache bombing limitation is an example of that as well.

But that will need the slab part of this patchset to follow. I not only have the code for that, but I posted it for review many times. That is not progressing fully so far because we're still looking for ways to keep the changes in the allocators to a minimum. Until I realized that the infrastructure could be used as-is, and stripped down from its original slab version to account pages, its proposed second user. And this is what you see here now.

So everything that uses resources in the kernel can proceed until it uses too much. Be it processes, files, or anything that consumes kernel

memory - and may potentially leave it there forever.

>> I can't speak for everybody here, but AFAIK, tracking the stack through
>> the memory it used, therefore using my proposed kmem controller, was an
>> idea that good quite a bit of traction with the memcg/memory people.
>> So here you have something that people already asked a lot for, in a
>> shape and interface that seem to be acceptable.
>
> mm, maybe. Kernel developers tend to look at code from the point of
> view "does it work as designed", "is it clean", "is it efficient", "do
> I understand it", etc. We often forget to step back and really
> consider whether or not it should be merged at all.
>
> I mean, unless the code is an explicit simplification, we should have
> a very strong bias towards "don't merge".

Well, simplifications are welcome - this series itself was simplified
beyond what I thought initially possible through the valuable comments
of other people.

But of course, this adds more complexity to the kernel as a whole. And
this is true to every single new feature we may add, now or in the
future.

What I can tell you about this particular one, is that the justification
for it doesn't come out of nowhere, but from a rather real use case that
we support and maintain in OpenVZ and our line of products for years.

It can potentially achieve more than that by being well used by schemes
such as systemd that aim at limiting the extent through which a service
can damage a system.

I hope all the above justification was enough to clarify any points you
may have. I'll be happy to go further if necessary.