Subject: controlling mmap()'d vs read/write() pages
Posted by Dave Hansen on Tue, 20 Mar 2007 16:15:34 GMT
View Forum Message <> Reply to Message

On Sun, 2007-03-18 at 11:42 -0600, Eric W. Biederman wrote:
> Dave Hansen <hansendc@us.ibm.com> writes:
> > To me, a process sitting there doing constant reads of 10 pages has the
> > same overhead to the VM as a process sitting there with a 10 page file
> > mmaped, and reading that.
>
> I can see temporarily accounting for pages in use for such a
> read/write and possibly during things such as read ahead.
>
> However I doubt it is enough memory to be significant, and as
> such is probably a waste of time accounting for it.
>
> A memory limit is not about accounting for memory pressure, so I think
> the reasoning for wanting to account for unmapped pages as a hard
> requirement is still suspect. A memory limit is to prevent one container
> from hogging all of the memory in the system, and denying it to other
> containers.
>
> The page cache by definition is a global resource that facilitates
> global kernel optimizations. If we kill those optimizations we
> are on the wrong track. By requiring limits there I think we are
> very likely to kill our very important global optimizations, and bring
> the performance of the entire system down.

Let's say you have an mmap'd file. It has zero pages brought in right
now. You do a write to it. It is well within the kernel's rights to
let you write one word to an mmap'd file, then unmap it, write it to
disk, and free the page.

To me, mmap() is an interface, not a directive to tell the kernel to
keep things in memory. The fact that two reads of a bytes from an
mmap()'d file tends to not go to disk or even cause a fault for the
second read is because the page is in the page cache. The fact that two
consecutive read()s of the same disk page tend to not cause two trips to
the disk is because the page is in the page cache.

Anybody who wants to get data in and out of a file can choose to use
either of these interfaces. A page being brought into the system for
either a read or touch of an mmap()'d area causes the same kind of
memory pressure.

So, I think we have a difference of opinion. I think it's _all_ about
memory pressure, and you think it is _not_ about accounting for memory
pressure. :) Perhaps we mean different things, but we appear to

disagree greatly on the surface.

Can we agree that there must be _some_ way to control the amounts of
unmapped page cache? Whether that's related somehow to the same way we
control RSS or done somehow at the I/O level, there must be some way to
control it. Agree?

> >> - Could you mention proper multi process RSS limits.
> >> (I.e. we count the number of pages each group of processes have mapped
> >> and limit that).
> >> It is the same basic idea as partial page ownership, but instead of
> >> page ownership you just count how many pages each group is using and
> >> strictly limit that. There is no page owner ship or partial charges.
> >> The overhead is just walking the rmap list at map and unmap time to
> >> see if this is the first users in the container. No additional kernel
> >> data structures are needed.
> >
> > I've tried to capture this. Let me know what else you think it
> > needs.
>
> Requirements:
> - The current kernel global optimizations are preserved and useful.
>
> This does mean one container can affect another when the
> optimizations go awry but on average it means much better
> performance. For many the global optimizations are what make
> the in-kernel approach attractive over paravirtualization.
>
> Very nice to have:
> - Limits should be on things user space have control of.
...
> - SMP Scalability.

> - Perfect precision.
...

I've tried to capture this:

http://linux-mm.org/SoftwareZones

> We need several more limits in this discussion to get a full picture,
> otherwise we may to try and build the all singing all dancing limit.
> - A limit on the number of anonymous pages.
> (Pages that are or may be in the swap cache).
> - Filesystem per container quotas.
> (Only applicable in some contexts but you get the idea).
> - Inode, file descriptor, and similar limits.
> - I/O limits.

Definitely. I think we've all agreed that memory is the hard one,
though. If we can make progress on this one, we're set! :)

-- Dave

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers