Subject: Re: [PATCH 0/8] RSS controller based on process containers (v2)
Posted by xemul on Tue, 10 Apr 2007 08:27:19 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:
> *ugh* /me no like.
>
> The basic premises seems to be that we can track page owners perfectly
> (although this patch set does not yet do so), through get/release

It looks like you have examined the patches not very carefully
before concluding this. These patches DO track page owners.

I know that a page may be shared among several containers and
thus have many owners so we should track all of them. This is
exactly what we decided not to do half-a-year ago.

Page sharing accounting is performed in OpenVZ beancounters, and
this functionality will be pushed to mainline after this simple
container.

> operations (on _mapcount).
>
> This is simply not true for unmapped pagecache pages. Those receive no
> 'release' event; (the usage by find_get_page() could be seen as 'get').

These patches concern the mapped pagecache only. Unmapped pagecache
control is out of the scope of it since we do not want one container
to track all the resources.

> Also, you don't seem to balance the active/inactive scanning on a per
> container basis. This skews the per container working set logic.

This is not true. Balbir sent a patch to the first version of this
container that added active/inactive balancing to the container.
I have included this (a bit reworked) patch into this version and
pointed this fact in the zeroth letter.

> Lastly, you don't call the slab shrinker for container reclaim; which
> would leave slab reclaim only for those few non process specific
> allocations, which would greatly skew the pagecache/slab balance.

Of course I do not call the slab shrinker! We do not have the
kernel memory control yet. Thus we can not shrink arbitrary kernel
objects just because some container has run out of its *user* memory.

Kernel memory control will come later. We decided to start from
a simple RSS control. Please, refer to containers archives for

more details.

>
>
> Let us call
>
> struct reclaim_struct {
> struct list_head active_list;
> struct list_head inactive_list;
> unsigned long nr_active;
> unsigned long nr_inactive;
> }
>
> Lets recognise three distinct page categories:
> - anonymous memory,
> - mapped pagecache, and
> - unmapped pagecache.

We cannot split the user memory in parts. There must be some
overall parameter that will allow administrator to say "Well,
let us run this container in a 64Mb sandbox". With the anonymous
and mapped memory separated administrator will be a bit confused.

>
>
> We then keep anonymous pages on a per container reclaim_struct, these
> pages are fully accounted to each container.

Hmm... We do have such a construction. struct rss_container has
two lists and we shrink from them sequentially using an existing
scanner. Don't forget that this scanner has been evolving for
many years and writing a new scanner is just a waste of time.

> We keep mapped pagecache pages on per inode reclaim_structs, these files
> could be shared between containers and we could either just account all
> pages belonging to each file proportional to the number of containers
> involved, or do a more precise accounting.

What happens if one container fills the RAM with mapped pages from
a single file? Who will be the "owner" of this page set? Who will
expend its IO bandwidth to push these pages on disk? What if this
container will mlock() this set? Who will be killed?

> We keep unmapped pagecache pages on a global reclaim_struct, these pages
> can, in general, not be pinned to a specific container; all we can do is
> keep a floating proportion relative to container 'get' events
> (find_get_page() and perhaps add_to_page_cache()).
>

> Reclaim will then have to fairly reclaim pages from all of these lists.
> If we schedule such that it appears that these lists are parallel
> instead of serial - that is a each tail is really a tail, not the head
> of another list - the current reclaim semantics are preserved.

Yet again. The current scanner came out from the work of many people.
This is a very tricky place that is still evolving. Do you propose
to throw this out and write a new scanner?

> The slab shrinker should be called proportional to the containers size
> relative to the machine.

The slab shrinker must be called only if we do know what kernel
objects are used by this particular container. Otherwise we break
the idea of isolation. Generally speaking if some container runs
out of its resources we should reclaim pages, shrink objects, kill
tasks, etc from this container only.

> Global reclaim will have to call each container reclaim in proportional
> fashion.
>
> The biggest problem with this approach is that there is no per zone
> reclaim left, which is relied upon by the allocator to provide free
> pages in a given physical address range. However there has been talk to
> create a proper range allocator independent of zones.
>
> Just my 0.02 euro..
>
> Peter
>
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/
>