Subject: Re: [ckrm-tech] [patch00/05]: Containers(V2)- Introduction
Posted by Rohit Seth on Thu, 28 Sep 2006 18:31:15 GMT
View Forum Message <> Reply to Message

On Thu, 2006-09-28 at 13:31 +0530, Balbir Singh wrote:
> Chandra Seetharaman wrote:
> > On Wed, 2006-09-27 at 14:28 -0700, Rohit Seth wrote:
> >
> > Rohit,
> >
> > For 1-4, I understand the rationale. But, your implementation deviates
> > from the current behavior of the VM subsystem which could affect the
> > ability of these patches getting into mainline.
> >
> > IMO, the current behavior in terms of reclamation, LRU, vm_swappiness,
> > and writeback logic should be maintained.
> >
>
> <snip>
>
> Hi, Rohit,
>
> I have been playing around with the containers patch. I finally got
> around to reading the code.
>
>
> 1. Comments on reclaiming
>
> You could try the following options to overcome some of the disadvantages of the
> current scheme.
>
> (a) You could consider a reclaim path based on Dave Hansen's Challenged memory
> controller (see http://marc.theaimsgroup.com/?l=linux-mm&m=1155669825323 45&w=2).
>

I will go through that. Did you get a chance to stress the system and
found any short comings that should be resolved.

> (b) The other option is to do what the resource group memory controller does -
> build a per group LRU list of pages (active, inactive) and reclaim
> them using the existing code (by passing the correct container pointer,
> instead of the zone pointer). One disadvantage of this approach is that
> the global reclaim is impacted as the global LRU list is broken. At the
> expense of another list, we could maintain two lists, global LRU and
> container LRU lists. Depending on the context of the reclaim - (container
> over limit, memory pressure) we could update/manipulate both lists.
> This approach is definitely very expensive.
>

Two LRUs is a nice idea. Though I don't think it will go too far. It
will involve adding another list pointers in the page structure. I
agree that the mem handler is not optimal at all but I don't want to
make it mimic kernel reclaimer at the same time.

> 2. Comments on task migration support
>
> (a) One of the issues I found while using the container code is that, one could
> add a task to a container say "a". "a" gets charged for the tasks usage,
> when the same task moves to a different container say "b", when the task
> exits, the credit goes to "b" and "a" remains indefinitely charged.
>

hmm, when the task is removed from "a" then "a" gets the credits for the
amount of anon memory that is used by the task. Or do you mean
something different.

> (b) For tasks addition and removal, I think it's probably better to move
> the entire process (thread group) rather than allow each individual thread
> to move across containers. Having threads belonging to the same process
> reside in different containers can be complex to handle, since they
> share the same VM. Do you have a scenario where the above condition
> would be useful?
>
>

I don't have a scenario where a task actually gets to move out of
container (except exit). That asynchronous removal of tasks has already
got the code very complicated for locking etc. But if you think moving
a thread group is useful then I will add that functionality.

Thanks,
-rohit