Subject: Re: [RFC][PATCH][0/4] Memory controller (RSS Control)
Posted by Magnus Damm on Mon, 19 Feb 2007 11:56:06 GMT
View Forum Message <> Reply to Message

On 2/19/07, Balbir Singh <balbir@in.ibm.com> wrote:
> Magnus Damm wrote:
> > On 2/19/07, Andrew Morton <akpm@linux-foundation.org> wrote:
> >> On Mon, 19 Feb 2007 12:20:19 +0530 Balbir Singh <balbir@in.ibm.com>
> >> wrote:
> >>
> >> > This patch applies on top of Paul Menage's container patches (V7)
> >> posted at
> >> >
> >> >   http://lkml.org/lkml/2007/2/12/88
> >> >
> >> > It implements a controller within the containers framework for limiting
> >> > memory usage (RSS usage).
> >
> >> The key part of this patchset is the reclaim algorithm:
> >>
> >> Alas, I fear this might have quite bad worst-case behaviour. One small
> >> container which is under constant memory pressure will churn the
> >> system-wide LRUs like mad, and will consume rather a lot of system time.
> >> So it's a point at which container A can deleteriously affect things
> >> which
> >> are running in other containers, which is exactly what we're supposed to
> >> not do.
> >
> > Nice with a simple memory controller. The downside seems to be that it
> > doesn't scale very well when it comes to reclaim, but maybe that just
> > comes with being simple. Step by step, and maybe this is a good first
> > step?
> >
>
> Thanks, I totally agree.
>
> > Ideally I'd like to see unmapped pages handled on a per-container LRU
> > with a fallback to the system-wide LRUs. Shared/mapped pages could be
> > handled using PTE ageing/unmapping instead of page ageing, but that
> > may consume too much resources to be practical.
> >
> > / magnus
>
> Keeping unmapped pages per container sounds interesting. I am not quite
> sure what PTE ageing, will it look it up.

You will most likely have no luck looking it up, so here is what I
mean by PTE ageing:

The most common unit for memory resource control seems to be physical pages. Keeping track of pages is simple in the case of a single user per page, but for shared pages tracking the owner becomes more complex.

I consider unmapped pages to only have a single user at a time, so the unit for unmapped memory resource control is physical pages. Apart from implementation details such as fun with struct page and scalability, handling this case is not so complicated.

Mapped or shared pages should be handled in a different way IMO. PTEs should be used instead of using physical pages as unit for resource control and reclaim. For the user this looks pretty much the same as physical pages, apart for memory overcommit.

So instead of using a global page reclaim policy and reserving physical pages per container I propose that resource controlled shared pages should be handled using a PTE replacement policy. This policy is used to keep the most active PTEs in the container backed by physical pages. Inactive PTEs gets unmapped in favour over newer PTEs.

One way to implement this could be by populating the address space of resource controlled processes with multiple smaller LRU2Qs. The compact data structure that I have in mind is basically an array of 256 bytes, one byte per PTE. Associated with this data strucuture are start indexes and lengths for two lists. The indexes are used in a FAT-type of chain to form single linked lists. So we create active and inactive list here - and we move PTEs between the lists when we check the young bits from the page reclaim and when we apply memory pressure. Unmapping is done through the normal page reclaimer but using information from the PTE LRUs.

In my mind this should lead to more fair resource control of mapped pages, but if it is possible to implement with low overhead, that's another question. =)

Thanks for listening.

/ magnus