Subject: Re: [RFC] mm-controller
Posted by Balbir Singh on Thu, 21 Jun 2007 11:03:59 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:
> Having read the RSS and Pagecache controllers some things bothered me.
>
> - the duplication of much of the reclaim data (not code)
> and the size increase as a result thereof.
>

Are you referring to the duplication due to the per container LRU list?

> - the clear distinction between mapped (RSS) and unmapped (pagecache)
> limits. Linux doesn't impose individual limits on these, so I don't
> see why containers should.
>

Linux doesn't impose induvidual limits, but we do have vm_swappiness and
vm_dirty_ratio to get some form of control over what pages we evict.
The advantage of splitting RSS and unmapped page cache is that

1. Users get more control over their containers
2. It's easier to implement containers in phases

We don't add any overhead by splitting out the control this way

> - lack of soft limits (but I understand that that is WIP :-)
>

Yes, that is correct.

> - while I appreciate the statistical nature of one container
> per page, I'm bit sceptical because there is no forced migration
> cycle.
>

I have patches for per container page_referenced() feature and we
mark_page_accessed() is already container aware

Please see http://lkml.org/lkml/2007/4/26/482

There is no forced migration, but when a container hits its limit.

http://lwn.net/Articles/237851/ talks about our current TODOs.

>
>

> So, while pondering the problem, I wrote down some ideas....
>
> While I appreciate that people do not like to start over again, I think
> it would be fruit-full to at least discuss the various design decisions.
>

First, thank you for your interest and your proposal. My opinion on
the new design is that, we should not move to it, _unless the current
design_ is completely non-fixable.

Don Knuth says he often re-learnt that
("Premature optimization is the root of all evil -- C.A.R Hoare").
Could you please try our patches

>
> (Will be travelling shorty, so replies might take longer than usual)
>
>

Me too...

> Mapped pages:
> ~~~~~~~~~~~~
>
> Basic premises:
>   - accounting per address_space/anon_vma
>

In the future, we also want to account for slab usage and page tables, etc.

> Because, if the data is shared between containers isolation is broken anyway
> and we might as well charge them equally [1].
>
> Move the full reclaim structures from struct zone to these structures.
>
>
> struct reclaim;
>
> struct reclaim_zone {
>   spinlock_t lru_lock;
>
>   struct list_head active;
>   struct list_head inactive;
>
>   unsigned long nr_active;
>   unsigned long nr_inactive;
>

> struct reclaim *reclaim;
> };
>
> struct reclaim {
> struct reclaim_zone zone_reclaim[MAX_NR_ZONES];
>
> spinlock_t containers_lock;
> struct list_head containers;
> unsigned long nr_containers;
> };
>
>
> struct address_space {
> ...
> struct reclaim reclaim;
> };
>

Each address space has a struct reclaim? which inturn has a per zone reclaim
LRU list?

> struct anon_vma {
> ...
> struct reclaim reclaim;
> };
>

Same comment as above

>
> Then, on instantiation of either address_space or anon_vma we string together
> these reclaim domains with a reclaim scheduler.
>
>
> struct sched_item;
>
> struct reclaim_item {
> struct sched_item sched_item;
> struct reclaim_zone *reclaim_zone;
> };
>
>
> struct container {
> ...
> struct sched_queue reclaim_queue;
> };
>
>

> sched_enqueue(&container->reclaim_queue, &reclaim_item.sched_item);
>
>
> Then, shrink_zone() would use the appropriate containers' reclaim_queue
> to find an reclaim_item to run isolate_pages on.
>
>
> struct sched_item *si;
> struct reclaim_item *ri;
> struct reclaim_zone *rzone;
> LIST_HEAD(pages);
>
> si = sched_current(&container->reclaim_queue);
> ri = container_of(si, struct reclaim_item, sched_item);
> rzone = ri->reclaim_zone;
> nr_scanned = isolate_pages(rzone, &pages);
>
> weight = (rzone->nr_active + rzone->nr_inactive) /
>   (nr_scanned * rzone->reclaim->nr_containers);
>
> sched_account(&container->reclaim_queue,
>   &rzone->sched_item, weight);
>
>
> We use a scheduler to interleave the various lists instead of a sequence of
> lists to create the appearance of a single longer list. That is, we want each
> tail to be of equal age.
>
> [it would probably make sense to drive the shrinking of the active list
>   from the use of the inactive list. This has the advantage of 'hiding'
>   the active list.
>
>   Much like proposed here: http://lkml.org/lkml/2005/12/7/57
>   and here: http://programming.kicks-ass.net/kernel-patches/page-replace
/2.6.21-pr0/useonce-new-shrinker.patch
>]
>

I worry about the scheduler approach. In a system with 100 containers, with 50 of them over their limit
and many sharing VMA's, how do we string together the container LRU list? What do we do on global
memory pressure?

> Unmapped pages:
> ~~~~~~~~~~~~~~~
>
>

> Since unmapped pages lack a 'release' (or dis-associate) event, the fairest
> thing is to account each container a fraction relative to its use of these
> unmapped pages.
>

Currently, we use the "page is reclaimed" event to mark the release event.

> Use would constitute of the following events:
> - pagecache insertion
> - pagecache lookup
>
>
> Each containers proportion will be calculated using the floating proportions
> introduced in the per BDI dirty limit patches.
>
> struct prop_global pagecache_proportion;
> struct reclaim pagecache_reclaim;
>
> enum reclaim_item_flags {
>   ri_pagecache,
> };
>
> struct reclaim_item {
>   ...
>   unsigned long flags;
> };
>
> struct container {
>   ...
>   struct prop_local_single pagecache_usage;
> };
>
>
> and add it to the vm scheduler.
>
>
> if (ri->flags & ri_pagecache) {
>   unsigned long num, denom;
>   unsigned long long w;
>
>   prop_fraction(&pagecache_proportion,
>   &container->pagecache_usage,
>   &num, &denom);
>
>   w = (rzone->nr_active + rzone->nr_inactive) * denom;
>   do_div(w, num);
>
>   weight = (unsigned long)w * nr_scanned;

> } else
>   ....
>
>
> Considerations:
> ~~~~~~~~~~~~~~~
>
>
> Advantages:
> - each page is on a single list
> - no distinction between container vs global reclaim
> - no increase of sizeof(struct page)
> - pages are but on a single list
> - breaks up lru_lock (might get a scheduler lock in return)
>
> Disadvantages:
> - major overhaul of the reclaim code
> - naive container usage calculation will be O(nr vmas)
>   however a smarter scheme would still be O(nr containers)
>
>
> Notes:
> ~~~~~
>
> [1] if needed one could refine this using floating proportions
>   charging each container a fraction relative to its usage
>
>
>

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL