Subject: Re: [RFC] mm-controller
Posted by Peter Zijlstra on Thu, 21 Jun 2007 13:50:55 GMT
View Forum Message <> Reply to Message

On Thu, 2007-06-21 at 16:33 +0530, Balbir Singh wrote:
> 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?

Yes, those bother me on two counts, the memory footprint, and
conceptually. How can one page have two ages?

The interaction between two lrus it non-trivial.

> > - 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

vm_dirty_ratio - is about dirty pages, mapped or not.
vm_swappiness - is the ugliest part of reclaim, and not something I
consider worthy of an example.

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

Not quite sure on 2, from reading the pagecache controller, I got the
impression that enforcing both limits got you into trouble. Merging the
limits would rid us of that issue, no?

> > - 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

Thanks.

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

Right, but the thing I'm worried about is a large part of text getting
stuck in 1 container because it is very active (glibc comes to mind).
That will never be properly distributed among containers.

Inter queue CPU load balancing should (and does with recent versions of
CFS IIRC) over balance the queues, otherwise you get the 2-cpu 3-tasks
2:1 imbalance where the lumped tasks get an unfair amount of cpu time.

I was thinking of similar artifacts.

With the current dual list approach, something like that could be done
by treating the container lists as pure FIFO (and ignore the reference
bit and all that) and make container reclaim only unmap, not write out
pages.

Then global reclaim will do the work (if needed), and containers get
churn, equating the page ownership.

> 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

Ah, here we differ of opinion; I see it as design, not optimisation.

> > 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?

nod

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

> > };
> >
>
> Same comment as above

indeed.

> >
> > 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?

```
              h  h  h  h  h  h  h   (hot end)
address_space    |  |  |  |  |  |
 lru lists       c  c  c  c  c  c  c   (cold end)


global queue     * * * * * * *

container 1      * *   *   *
container 2        * *   *
```

each reclaim context will iterate over all enqueued lists, taking a few
elements off of each tail, scheduling them like 1/N owners.

> > 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.

That is artificial, and only possible because you claim single ownership.