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

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.

Subject: Re: [RFC] mm-controller
Posted by Balbir Singh on Thu, 21 Jun 2007 17:31:22 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:
> 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?
>

You need to put on your virtualization cap :-)

> The interaction between two lrus it non-trivial.
>

Actually, the interaction is easy to understand. The per-zone LRU is
a superset of the per-container LRU. A page may belong to both, but
if it is a user page, it will definitely belong to the per-zone LRU.
Isolation of pages occur in both LRU's occur independently. The page
age in the per container LRU list is less than or equal to it's age
in the per-zone 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
>
> vm_dirty_ratio - is about dirty pages, mapped or not.

Yes, but it's a way of controlling/keeping the reclaim time under check.
Too many dirty pages == longer time to recycle a page. Also, it helps
consistency of data. The reason I mentioned this, is because it does
provide some form of control.

> vm_swappiness - is the ugliest part of reclaim, and not something I
> consider worthy of an example.
>

I agree, but my point was that there are scenarios where people want
to be able to control (only) page cache. One example, that I am aware
of is applications that manage their own page cache and use
O_DIRECT for IO. They would want to have almost zero page cache.

>> 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?
>

Yes, it would. This point is worth discussing, but with the example above
I feel we need to differentiate between the two.

>>> - 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.
>

But I still wonder if we need to distribute it, if container 1 is using
it's pages actively. Others get a free ride, but as long as the system
administrators are aware of that happening, it's all good.

> 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.
>

I did implement the unmap only logic for shared pages in version 2
of my RSS controller

http://lkml.org/lkml/2007/2/19/10

It can be added back if required quite easily. Pavel what do you think
about it?

>> 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.
>
>

Yes, I see it as design optimization. But, no seriously, I would request
that we try and see if this design will work and see it's performance and
usability and maintenance overheads before moving on to a newer design.

>>> 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*
>
```

Won't this really bloat up used memory. I also worry about treating address_space
and anon_vma's as objects that manage and reclaim their own memory. Consider for
example a filesystem like sysfs, where each file is associated with a maximum
of one page (on an average).

```
>>>  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.
>
>

This worries me a bit. On a per-zone LRU, pages would look like

 p1 <--> p2 <--> p3 <--> .... p(n)

Now each page has a unique age, but with a per-address-space or per-anon-vma the age uniqueness is lost.

>>> 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.

>

Yes, the design aspects all weave in together :-)

>
> --
> To unsubscribe, send a message with 'unsubscribe linux-mm' in
> the body to majordomo@kvack.org. For more info on Linux MM,
> see: http://www.linux-mm.org/ .
> Don't email: <a href=mailto:"dont@kvack.org"> email@kvack.org </a>

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

Subject: Re: [RFC] mm-controller
Posted by Pavel Emelianov on Fri, 22 Jun 2007 02:21:54 GMT

View Forum Message <> Reply to Message

Balbir Singh wrote:

[snip]

>> 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.
>>
>
> I did implement the unmap only logic for shared pages in version 2
> of my RSS controller
>
> http://lkml.org/lkml/2007/2/19/10
>
> It can be added back if required quite easily. Pavel what do you think
> about it?

I think it's wrong. Look, when the container hits the limit and just
unmaps the pages the following situation may occur: some *other* container
will hit the global shortage and will have to wait till the other's
pages are flushed to disk. This is not a true isolation. If we send the

pages to the disk right when the container hits the limit we spend its
time, its IO bandwidth, etc and allow for others to have the free set of
pages without additional efforts.

[snip]

```
>>>> 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;
>>>>  };
>>>>
```

Peter, could you prepare some POC patches instead? See, when looking at
the patches is simpler to understand what is going on then when reading
the plain text. Moreover, when making the patches some unexpected details
of the kernel internals arise and the ideas begin to change...

[snip]

Subject: Re: [RFC] mm-controller
Posted by Vaidyanathan Srinivas on Fri, 22 Jun 2007 16:35:19 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:
> On Thu, 2007-06-21 at 16:33 +0530, Balbir Singh wrote:
>> Peter Zijlstra wrote:
[snip]

> 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?
>

Hi Peter,

Setting both limits in pagecache controller (v4) will work correct in principle. What I intended in the comment is performance issues and wrong type of page being reclaimed. We are working on the issues and will fix them in future versions. You can set any combination of limits and the reclaim will work to keep the page utilization below limits.

When RSS limit is hit, ANON pages are pushed to swapcache. We would need to limit swapcache (using pagecache_limit) to force a write out to disk.

Merging both limits will eliminate the issue, however we would need individual limits for pagecache and RSS for better control. There are use cases for pagecache_limit alone without RSS_limit like the case of database application using direct IO, backup applications and streaming applications that does not make good use of pagecache.

Thank you for the review and new design proposal.

--Vaidy

[snip]

Subject: Re: [RFC] mm-controller
Posted by Peter Zijlstra on Mon, 25 Jun 2007 16:22:41 GMT
View Forum Message <> Reply to Message

On Fri, 2007-06-22 at 22:05 +0530, Vaidyanathan Srinivasan wrote:

> Merging both limits will eliminate the issue, however we would need
> individual limits for pagecache and RSS for better control. There are

> use cases for pagecache_limit alone without RSS_limit like the case of
> database application using direct IO, backup applications and
> streaming applications that does not make good use of pagecache.

I'm aware that some people want this. However we rejected adding a pagecache limit to the kernel proper on grounds that reclaim should do a better job.

And now we're sneaking it in the backdoor.

If we're going to do this, get it in the kernel proper first.

Subject: Re: [RFC] mm-controller
Posted by Balbir Singh on Mon, 25 Jun 2007 17:33:24 GMT

View Forum Message <> Reply to Message

Peter Zijlstra wrote:
> On Fri, 2007-06-22 at 22:05 +0530, Vaidyanathan Srinivasan wrote:
>
>> Merging both limits will eliminate the issue, however we would need
>> individual limits for pagecache and RSS for better control. There are
>> use cases for pagecache_limit alone without RSS_limit like the case of
>> database application using direct IO, backup applications and
>> streaming applications that does not make good use of pagecache.
>
> I'm aware that some people want this. However we rejected adding a
> pagecache limit to the kernel proper on grounds that reclaim should do a
> better job.
>
> And now we're sneaking it in the backdoor.
>

We'll we are trying to provide a memory controller and page cache is a part of memory. The page reclaimer does treat page cache separately. Isn't this approach better than simply extending the vm_swappiness to per container?

> If we're going to do this, get it in the kernel proper first.
>

I'm open to this. There were several patches to do this. We can do this by splitting the LRU list to mapped and unmapped pages or by trying to balance the page cache by tracking it's usage.

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Subject: Re: [RFC] mm-controller
Posted by Paul Menage on Mon, 25 Jun 2007 17:35:26 GMT
View Forum Message <> Reply to Message

On 6/22/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
>
> Merging both limits will eliminate the issue, however we would need
> individual limits for pagecache and RSS for better control. There are
> use cases for pagecache_limit alone without RSS_limit like the case of
> database application using direct IO, backup applications and
> streaming applications that does not make good use of pagecache.
>

If streaming applications would otherwise litter the pagecache with
unwanted data, then limiting their total memory footprint (with a
single limit) and forcing them to drop old data sooner sounds like a
great idea.

Paul

Subject: Re: [RFC] mm-controller
Posted by Paul Menage on Mon, 25 Jun 2007 17:36:37 GMT
View Forum Message <> Reply to Message

On 6/25/07, Paul Menage <menage@google.com> wrote:
> On 6/22/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
> >
> > Merging both limits will eliminate the issue, however we would need
> > individual limits for pagecache and RSS for better control. There are
> > use cases for pagecache_limit alone without RSS_limit like the case of
> > database application using direct IO, backup applications and
> > streaming applications that does not make good use of pagecache.
> >
>
> If streaming applications would otherwise litter the pagecache with
> unwanted data, then limiting their total memory footprint (with a
> single limit) and forcing them to drop old data sooner sounds like a
> great idea.

Actually, reading what you wrote more carefully, that's sort of what

you were already saying. But it's not clear why you wouldn't also want
to limit the anon pages for a job, if you're already concerned that
it's not playing nicely with the rest of the system.

Paul

Subject: Re: [RFC] mm-controller
Posted by Vaidyanathan Srinivas on Mon, 25 Jun 2007 18:02:37 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:
> On Fri, 2007-06-22 at 22:05 +0530, Vaidyanathan Srinivasan wrote:
>
>> Merging both limits will eliminate the issue, however we would need
>> individual limits for pagecache and RSS for better control. There are
>> use cases for pagecache_limit alone without RSS_limit like the case of
>> database application using direct IO, backup applications and
>> streaming applications that does not make good use of pagecache.
>
> I'm aware that some people want this. However we rejected adding a
> pagecache limit to the kernel proper on grounds that reclaim should do a
> better job.
>
> And now we're sneaking it in the backdoor.
>
> If we're going to do this, get it in the kernel proper first.
>

Good point. We should probably revisit this in the context of
containers, virtualization and server consolidation. Kernel takes the
best decision in the context of overall system performance, but when
we want the kernel to favor certain group of application relative to
others then we hit corner cases. Streaming multimedia applications
are one of the corner case where the kernel's effort to manage
pagecache does not help overall system performance.

There have been several patches suggested to provide system wide
pagecache limit. There are some user mode fadvice() based techniques
as well. However solving the problem in the context of containers
provide certain advantages

* Containers provide task grouping
* Relative priority or importance can be assigned to each group using
resource limits.
* Memory controller under container framework provide infrastructure
for detailed accounting of memory usage
* Containers and controllers form generalised infrastructure to create

localised VM behavior for a group of tasks

I would see introduction of pagecache limit in containers as a safe place to add the new feature rather than a backdoor. Since this feature has a relatively small user base, it be best left as a container plugin rather than a system wide tunable.

I am not suggesting against system wide pagecache control. We should definitely try to find solutions for pagecache control outside of containers as well.

--Vaidy

Subject: Re: [RFC] mm-controller
Posted by Vaidyanathan Srinivas on Mon, 25 Jun 2007 18:22:03 GMT

View Forum Message <> Reply to Message

Paul Menage wrote:
> On 6/25/07, Paul Menage <menage@google.com> wrote:
>> On 6/22/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
>>> Merging both limits will eliminate the issue, however we would need
>>> individual limits for pagecache and RSS for better control. There are
>>> use cases for pagecache_limit alone without RSS_limit like the case of
>>> database application using direct IO, backup applications and
>>> streaming applications that does not make good use of pagecache.
>>>
>> If streaming applications would otherwise litter the pagecache with
>> unwanted data, then limiting their total memory footprint (with a
>> single limit) and forcing them to drop old data sooner sounds like a
>> great idea.
>
> Actually, reading what you wrote more carefully, that's sort of what
> you were already saying. But it's not clear why you wouldn't also want
> to limit the anon pages for a job, if you're already concerned that
> it's not playing nicely with the rest of the system.

Hi Paul,

Limiting memory footprint (RSS and pagecache) for multi media applications would work. However, generally streaming applications have a fairly constant RSS size (mapped pagecache pages + ANON) while the unmapped pagecache pages is what we want to control better.

If we have a combined limit for unmapped pagecache pages and RSS, then we will have to bring in vm_swappiness kind of knobs for each container to influence the per container reclaim process so as to not hurt the application performance badly.

RSS controller should be able to take care of the mapped memory
footprint if needed. In case of database server, moving out any of it
RSS pages will hurt it performance, while we are free to shrink the
unmapped pagecache pages to any smaller limit since the database is
using direct IO and does not benefit from pagecache.

With pagecache controller, we are able to split application's memory
pages into mapped and unmapped pages. Ability to account and control
unmapped pages in memory provides more possibilities for fine grain
resource management.

--Vaidy