Subject: Supporting overcommit with the memory controller
Posted by Paul Menage on Thu, 06 Mar 2008 00:17:13 GMT
View Forum Message <> Reply to Message

We want to be able to use the memory controller in the following way, and I'd like to know how practical this is currently, and will be in the future.

Users are poor at determining how much memory their jobs will actually use (partly due to poor estimation, partly due to high variance of memory usage on some jobs). So, we want to overcommit machines, i.e. we want the total limits granted to all cgroups add up to more than the total size of the machine.

Our central scheduler will try to ensure that the jobs that are packed on to the same machine are unlikely to all hit their peak usage at once, so the machine as a whole is unlikely to actually run out of memory. But sometimes it will be over-optimistic, and the machine will run out of memory. We will try to ensure that there's a mixture of high and low priority jobs on a machine, so that when the machine runs out of memory the OOM killer can nuke the low-priority jobs and we can reschedule them elsewhere.

The tricky bit is that we don't want this OOM process to impact the high-priority jobs on the machine. I.e. even while the low-priority job is OOM-killing itself, the high priority job shouldn't have any difficulty in doing regular memory allocations. And if the high-priority job gets a spike in its memory usage, we want the low-priority jobs to get killed quickly and cleanly to free up memory for the high-priority job, without stalling the high-priority job.

So for each job we need a (per-job configurable) amount of memory that's essentially reserved for that job. That way the high-priority job can carry on allocating from its reserved pool even while the low-priority job is OOMing; the low-priority job can't touch the reserved pool of the high-priority job.

But to make this more interesting, there are plenty of jobs that will happily fill as much pagecache as they have available. Even a job that's just writing out logs will continually expand its pagecache usage without anything to stop it, and so just keeping the reserved pool at a fixed amount of free memory will result in the job expanding even if it doesn't need to. Therefore we want to be able to include in the "reserved" pool, memory that's allocated by the job, but which can be freed without causing performance penalties for the job. (e.g. log files, or pages from a large on-disk data file with little access locality of reference) So suppose we'd decided to keep a reserve of 200M for a particular job - if it had 200M of stale log file pages in

the pagecache then we could treat those as the 200M reserve, and not
have to keep on expanding the reserve pool.

We've been approximating this reasonably well with a combination of
cpusets, fake numa, and some hacks to determine how many pages in each
node haven't been touched recently (this is a bit different from the
active/inactive distinction). By assigning physical chunks of memory
(fake numa nodes) to different jobs, we get the pre-reservation that
we need. But using fake numa is a little inflexible, so it would be
nice to be able to use a page-based memory controller.

Is this something that would be possible to set up with the current
memory controller? My impression is that this isn't quite possible
yet, but maybe I've not just thought hard enough. I suspect that we'd
need at least the addition of page refault data, and the ability to
pre-reserve pages for a group.

Paul

Subject: Re: Supporting overcommit with the memory controller
Posted by KAMEZAWA Hiroyuki on Thu, 06 Mar 2008 00:59:31 GMT
View Forum Message <> Reply to Message

On Wed, 5 Mar 2008 16:17:13 -0800
"Paul Menage" <menage@google.com> wrote:
> Users are poor at determining how much memory their jobs will actually
> use (partly due to poor estimation, partly due to high variance of
> memory usage on some jobs). So, we want to overcommit machines, i.e.
> we want the total limits granted to all cgroups add up to more than
> the total size of the machine.
>
just depends on middle-ware. I think most of them will not allow that.

> So for each job we need a (per-job configurable) amount of memory
> that's essentially reserved for that job. That way the high-priority
> job can carry on allocating from its reserved pool even while the
> low-priority job is OOMing; the low-priority job can't touch the
> reserved pool of the high-priority job.
>
Hmm, but current resource charging is independent from page allocator.
(I think this is a good aspect of current design.)

> But to make this more interesting, there are plenty of jobs that will
> happily fill as much pagecache as they have available. Even a job
> that's just writing out logs will continually expand its pagecache
> usage without anything to stop it, and so just keeping the reserved
> pool at a fixed amount of free memory will result in the job expanding
> even if it doesn't need to.
It's current memory management style. "reclaim only when necessary".

> Therefore we want to be able to include in
> the "reserved" pool, memory that's allocated by the job, but which can
> be freed without causing performance penalties for the job. (e.g. log
> files, or pages from a large on-disk data file with little access
> locality of reference) So suppose we'd decided to keep a reserve of
> 200M for a particular job - if it had 200M of stale log file pages in
> the pagecache then we could treat those as the 200M reserve, and not
> have to keep on expanding the reserve pool.
>
> We've been approximating this reasonably well with a combination of
> cpusets, fake numa, and some hacks to determine how many pages in each
> node haven't been touched recently (this is a bit different from the
> active/inactive distinction). By assigning physical chunks of memory
> (fake numa nodes) to different jobs, we get the pre-reservation that
> we need. But using fake numa is a little inflexible, so it would be
> nice to be able to use a page-based memory controller.
>
> Is this something that would be possible to set up with the current
> memory controller? My impression is that this isn't quite possible
> yet, but maybe I've not just thought hard enough. I suspect that we'd
> need at least the addition of page refault data, and the ability to
> pre-reserve pages for a group.
>
Can Balbir's soft-limit patches help ?

It reclamims each cgroup's pages to soft-limit if the system needs.

Make limitation like this

Assume 4G server.

	Limit	soft-limit
Not important Apss:	2G	100M
Important Apps :	3G	2.7G

When the system memory reachs to the limit, each cgroup's memory usages will goes down to soft-limit. (And there will 1.3G of free pages in above example)

Thanks,
-Kame

Subject: Re: Supporting overcommit with the memory controller
Posted by Paul Menage on Thu, 06 Mar 2008 02:54:52 GMT
View Forum Message <> Reply to Message

On Wed, Mar 5, 2008 at 5:01 PM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > But to make this more interesting, there are plenty of jobs that will
> > happily fill as much pagecache as they have available. Even a job
> > that's just writing out logs will continually expand its pagecache
> > usage without anything to stop it, and so just keeping the reserved
> > pool at a fixed amount of free memory will result in the job expanding
> > even if it doesn't need to.
> It's current memory management style. "reclaim only when necessary".
>

Exactly - if the high-priority latency-sensitive job really needs that
extra memory, we want it to be able to automatically squash/kill the
low-priority job when memory runs low, and not suffer any latency
spikes. But if it doesn't actually need the memory, we'd rather use it
for low-priority batch stuff. The "no latency spikes" bit is important
- we don't want the high-priority job to get bogged down in
try_to_free_pages() and out_of_memory() loops when it needs to
allocate memory.

> >
> Can Balbir's soft-limit patches help ?
>
> It reclamims each cgroup's pages to soft-limit if the system needs.
>
> Make limitation like this
>
> Assume 4G server.
>   Limit   soft-limit
> Not important Apss: 2G   100M
> Important Apps   :   3G   2.7G
>
> When the system memory reachs to the limit, each cgroup's memory usages will
> goes down to soft-limit. (And there will 1.3G of free pages in above example)
>

Yes, that could be a useful part of the solution - I suspect we'd need
to have kswapd do the soft-limit push back as well as in
try_to_free_pages(), to avoid the high-priority jobs getting stuck in
the reclaim code. It would also be nice if we had:

- a way to have the soft-limit pushing kick in substantially *before*
the machine ran out of memory, to provide a buffer for the
high-priority jobs.

- a way to measure the actual working set of a cgroup (which may be
smaller than its allocated memory if it has plenty of stale pagecache
pages allocated). Maybe refaults, or maybe usage-based information.

Paul

Subject: Re: Supporting overcommit with the memory controller
Posted by KAMEZAWA Hiroyuki on Thu, 06 Mar 2008 03:19:52 GMT
View Forum Message <> Reply to Message

On Wed, 5 Mar 2008 18:54:52 -0800
"Paul Menage" <menage@google.com> wrote:

> On Wed, Mar 5, 2008 at 5:01 PM, KAMEZAWA Hiroyuki
> <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > > But to make this more interesting, there are plenty of jobs that will
> > > happily fill as much pagecache as they have available. Even a job
> > > that's just writing out logs will continually expand its pagecache
> > > usage without anything to stop it, and so just keeping the reserved
> > > pool at a fixed amount of free memory will result in the job expanding
> > > even if it doesn't need to.
> > It's current memory management style. "reclaim only when necessary".
> >
>
> Exactly - if the high-priority latency-sensitive job really needs that
> extra memory, we want it to be able to automatically squash/kill the
> low-priority job when memory runs low, and not suffer any latency
> spikes. But if it doesn't actually need the memory, we'd rather use it
> for low-priority batch stuff. The "no latency spikes" bit is important
> - we don't want the high-priority job to get bogged down in
> try_to_free_pages() and out_of_memory() loops when it needs to
> allocate memory.
>
In our measurements(on RHEL5), setting dirty_ratio to suitable value can

help us to avoid *long* latency in most of *usual* situation.
(I'm sorry that I can't show the numbers, please try.)
Some mm people are trying to improve the kernel behavior under *unusual*
situation. If you don't want any latency spikes for high priority processes,
we'll have to try to make global page allocator handle priority of process/pages.

It seems what you really want is priority based file-cache control.
I have no objectio to using cgroup as controller interface of it.

For avoiding spike, I'm now considering to support dirty_ratio
for memcg. (Now, it seems difficut.)

> > >
> > Can Balbir's soft-limit patches help ?
> >
> > It reclamims each cgroup's pages to soft-limit if the system needs.
> >
> > Make limitation like this
> >
> > Assume 4G server.
> >   Limit   soft-limit
> > Not important Apss: 2G   100M
> > Important Apps   :   3G   2.7G
> >
> > When the system memory reachs to the limit, each cgroup's memory usages will
> > goes down to soft-limit. (And there will 1.3G of free pages in above example)
> >
>
> Yes, that could be a useful part of the solution - I suspect we'd need
> to have kswapd do the soft-limit push back as well as in
> try_to_free_pages(), to avoid the high-priority jobs getting stuck in
> the reclaim code. It would also be nice if we had:
>
> - a way to have the soft-limit pushing kick in substantially *before*
> the machine ran out of memory, to provide a buffer for the
> high-priority jobs.
>
Maybe background-reclaim thread can be a help. (I'm now maintaining a patch.)

> - a way to measure the actual working set of a cgroup (which may be
> smaller than its allocated memory if it has plenty of stale pagecache
> pages allocated). Maybe refaults, or maybe usage-based information.
>
Hmm, current memory resource controller shows

- failcnt
- active/inactive
- rss/cache

I think we have enough infrastructure to account additional parameters.
But I think support all vmstat members for memcg is a bit overkill.
We'll have to choice what is necessary.

Thanks,
-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Supporting overcommit with the memory controller
Posted by Pavel Emelianov on Thu, 06 Mar 2008 08:55:47 GMT
View Forum Message <> Reply to Message

>> Can Balbir's soft-limit patches help ?

[snip]

>
> Yes, that could be a useful part of the solution - I suspect we'd need
> to have kswapd do the soft-limit push back as well as in
> try_to_free_pages(), to avoid the high-priority jobs getting stuck in
> the reclaim code. It would also be nice if we had:

BTW, one of the way OpenVZ users determine how much memory they
need for containers is the following: they set the limits to
maximal values and then check the "maxheld" (i.e. the maximal level
of consumption over the time) value.

Currently, we don't have such in res_counters and I'm going to
implement this. Objections?

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Supporting overcommit with the memory controller
Posted by KAMEZAWA Hiroyuki on Thu, 06 Mar 2008 09:04:24 GMT
View Forum Message <> Reply to Message

On Thu, 06 Mar 2008 11:55:47 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> >> Can Balbir's soft-limit patches help ?
>
> [snip]
>
> >
> > Yes, that could be a useful part of the solution - I suspect we'd need
> > to have kswapd do the soft-limit push back as well as in
> > try_to_free_pages(), to avoid the high-priority jobs getting stuck in
> > the reclaim code. It would also be nice if we had:
>
> BTW, one of the way OpenVZ users determine how much memory they
> need for containers is the following: they set the limits to
> maximal values and then check the "maxheld" (i.e. the maximal level
> of consumption over the time) value.
>
> Currently, we don't have such in res_counters and I'm going to
> implement this. Objections?
>
Basically, no objection.

BTW, which does it means ?
- create a new cgroup to accounting max memory consumption, etc...
or
- add new member to mem_cgroup
or
- add new member to res_counter

Thanks,
-Kame

Subject: Re: Supporting overcommit with the memory controller
Posted by Pavel Emelianov on Thu, 06 Mar 2008 09:07:57 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Thu, 06 Mar 2008 11:55:47 +0300
> Pavel Emelyanov <xemul@openvz.org> wrote:
>
>>>> Can Balbir's soft-limit patches help ?
>> [snip]
>>

>>> Yes, that could be a useful part of the solution - I suspect we'd need
>>> to have kswapd do the soft-limit push back as well as in
>>> try_to_free_pages(), to avoid the high-priority jobs getting stuck in
>>> the reclaim code. It would also be nice if we had:
>> BTW, one of the way OpenVZ users determine how much memory they
>> need for containers is the following: they set the limits to
>> maximal values and then check the "maxheld" (i.e. the maximal level
>> of consumption over the time) value.
>>
>> Currently, we don't have such in res_counters and I'm going to
>> implement this. Objections?
>>
> Basically, no objection.
>
> BTW, which does it means ?
> - create a new cgroup to accounting max memory consumption, etc...
> or
> - add new member to mem_cgroup
> or
> - add new member to res_counter

The third one - new member on res_counter. This will cost us 8 more
bytes on mem_cgroup and no performance impact, since the new field
is about to be touched only together with the limit and usage ones,
and thus is in one cacheline.

> Thanks,
> -Kame
>
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Supporting overcommit with the memory controller
Posted by Balbir Singh on Thu, 06 Mar 2008 18:42:18 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> We want to be able to use the memory controller in the following way,
> and I'd like to know how practical this is currently, and will be in
> the future.
>
> Users are poor at determining how much memory their jobs will actually
> use (partly due to poor estimation, partly due to high variance of

> memory usage on some jobs). So, we want to overcommit machines, i.e.
> we want the total limits granted to all cgroups add up to more than
> the total size of the machine.
>
> Our central scheduler will try to ensure that the jobs that are packed
> on to the same machine are unlikely to all hit their peak usage at
> once, so the machine as a whole is unlikely to actually run out of
> memory. But sometimes it will be over-optimistic, and the machine will
> run out of memory. We will try to ensure that there's a mixture of
> high and low priority jobs on a machine, so that when the machine runs
> out of memory the OOM killer can nuke the low-priority jobs and we can
> reschedule them elsewhere.
>
> The tricky bit is that we don't want this OOM process to impact the
> high-priority jobs on the machine. I.e. even while the low-priority
> job is OOM-killing itself, the high priority job shouldn't have any
> difficulty in doing regular memory allocations. And if the
> high-priority job gets a spike in its memory usage, we want the
> low-priority jobs to get killed quickly and cleanly to free up memory
> for the high-priority job, without stalling the high-priority job.
>
> So for each job we need a (per-job configurable) amount of memory
> that's essentially reserved for that job. That way the high-priority
> job can carry on allocating from its reserved pool even while the
> low-priority job is OOMing; the low-priority job can't touch the
> reserved pool of the high-priority job.
>
> But to make this more interesting, there are plenty of jobs that will
> happily fill as much pagecache as they have available. Even a job
> that's just writing out logs will continually expand its pagecache
> usage without anything to stop it, and so just keeping the reserved
> pool at a fixed amount of free memory will result in the job expanding
> even if it doesn't need to. Therefore we want to be able to include in
> the "reserved" pool, memory that's allocated by the job, but which can
> be freed without causing performance penalties for the job. (e.g. log
> files, or pages from a large on-disk data file with little access
> locality of reference) So suppose we'd decided to keep a reserve of
> 200M for a particular job - if it had 200M of stale log file pages in
> the pagecache then we could treat those as the 200M reserve, and not
> have to keep on expanding the reserve pool.
>
> We've been approximating this reasonably well with a combination of
> cpusets, fake numa, and some hacks to determine how many pages in each
> node haven't been touched recently (this is a bit different from the
> active/inactive distinction). By assigning physical chunks of memory
> (fake numa nodes) to different jobs, we get the pre-reservation that
> we need. But using fake numa is a little inflexible, so it would be
> nice to be able to use a page-based memory controller.

>
> Is this something that would be possible to set up with the current
> memory controller? My impression is that this isn't quite possible
> yet, but maybe I've not just thought hard enough. I suspect that we'd
> need at least the addition of page refault data, and the ability to
> pre-reserve pages for a group.

I have some patches for implementing soft-limits. Have you explored to see if they can sort your problem? I am thinking of adding additional statistics like page-in, page-out rates and eventually refault statistics.

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