Subject: Summary of resource management discussion
Posted by Srivatsa Vaddagiri on Mon, 12 Mar 2007 12:42:26 GMT
View Forum Message <> Reply to Message

I happened to read the entire thread (@ http://lkml.org/lkml/2007/3/1/159)
all over again and felt it may be usefull to summarize the discussions so far.

If I have missed any imp. points or falsely represented someone's view
(unintentionally of course!), then I would be glad to be corrected.

1. Which task-grouping mechanism?

[This question is the most vital one that needs a consensus]

Resource management generally works by apply resource controls over a -group-
of tasks (tasks of a user, tasks in a vserver/container etc).

What mechanism do we use to group tasks for res mgmt purposes?

Options:

a. Paul Menage's container(/uh-a-diff-name-pls?) patches

The patches introduce a new pointer in task_struct, struct
container_group *containers, and a new structure 'struct container'.

Tasks pointing to the same 'struct container' object (via their
tsk->containers->container[] pointer) are considered to form
a group associated with that container. The attributes associated
with a container (ex: cpu_limit, rss_limit, cpus/mems_allowed) are
decided by the options passed to mount command (which binds
one/more/all resource controllers to a hierarchy).

+ For workload management, where it is desirable to manage resource
consumption of a run-time defined (potentially arbitrary) group of
tasks, then this patch is handy, as no existing pointers in
task_struct can be used to form such a run-time decided group.

- (subjective!) If there is a existing grouping mechanism already (say
tsk->nsproxy[->pid_ns]) over which res control needs to be applied,
then the new grouping mechanism can be considered redundant (it can
eat up unnecessary space in task_struct)

What may help avoid this redundancy is to re-build existing
grouping mechanism (say tsk->nsproxy) using the container patches.
Serge however expressed some doubts on such a implementation
(for ex: how will one build hierarchical cpusets and non-hierarchical
namespaces using that single 'grouping' pointer in task_struct) and

also felt it may slow down things a bit from namespaces pov (more dereferences reqd to get to a task's namespace).

b. Reuse existing pointers in task_struct, tsk->nsproxy or better perhaps tsk->nsproxy->pid_ns, as the means to group tasks (rcfs patches)

This is based on the observation that the group of tasks whose resource consumption need to be managed is already defined in the kernel by existing pointers (either tsk->nsproxy or tsk->nsproxy->pid_ns)

+ reuses existing grouping mechanism in kernel

- mixes resource and name spaces (?)

c. Introduce yet-another new structure ('struct res_ctl?') which houses resource control (& possibly pid_ns?) parameters and a new pointer to this structure in task_struct (Herbert Poetzl).

Tasks that have a pointer to the same 'struct res_ctl' are considered to form a group for res mgmt purpose

+ Accessing res ctl information in scheduler fast path is optimized (only two-dereferences required)

- If all resource control parameters (cpu, memory, io etc) are lumped together in same structure, it makes it hard to have resource classes (cpu, mem etc) that are independent of each other.

- If we introduce several pointers in task_struct to allow separation of resource classes, then it will increase storage space in task_struct and also fork time (we have to take ref count on more than one object now). Herbert thinks this is worthy tradeoff for the benefit gained in scheduler fast paths.

2. Where do we put resource control parameters for a group?

This depends on 1. So the options are:

a. Paul Menage's patches:

(tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit

An optimized version of the above is:
(tsk->containers->subsys[cpu_ctlr.subsys_id] - X)->cpu_limit

b. rcfs
 tsk->nsproxy->ctlr_data[cpu_ctlr.subsys_id]->cpu_limit

c. Herbert's proposal
 tsl->res_ctl->cpu_limit

3. How are cpusets related to vserver/containers?

 Should it be possible to, lets say, create exclusive cpusets and
 attach containers to different cpusets?

4. Interface
 Filesystem vs system call

 Filesystem:
  + natural way to represent hierarchical data
  + File permission model convenient to delegate
      management of part of a tree to one user
  + Ease of use with scripts

  (from Herbet Poetzl):

  - performance of filesystem interfaces is quite bad
  - you need to do a lot to make the fs consistant for
    e.g. find and friends (regarding links and filesize)
  - you have a quite hard time to do atomic operations
    (except for the ioctl interface, which nobody likes)
  - vfs/mnt namespaces complicate the access to this
    new filesystem once you start moving around (between
          the spaces)

5. If we use filesystem interface, then should it be in /proc? (Eric)

 - /proc doesn't allow the flexibility of say creating multiple
   hierarchies and binding different resource controllers to each
   hierarchy

6. As tasks move around namespaces/resource-classes, their
   tsk->nsproxy/containers object will change. Do we simple create
   a new nsproxy/containers object or optimize storage by searching
   for one which matches the task's new requirements?

 - Linux Vserver follows former approach i.e simply creates
   a new nsproxy with pointers to required namespace objects

7. Hierarchy

- For res mgmt, do we need to worry about hierarchy at all?

- If we consider cpuset to be a resource controller,
  then we have one resource controller who already
  supports hierarchy

- If we don't support hierarchy in res controllers today
  but were to add that support later, then
  user-interface shouldn't change. That's why
  designining -atleast- the user interface to support
  hierarchy may make sense

- Do we let resource classes to be split independent of each?

  For ex: CPU resource classes are independent of memory resource
  classes. This inturn affect whether the Paul Menage's patches
  need to support multiple hierarchy feature.

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by Herbert Poetzl on Tue, 13 Mar 2007 16:24:59 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 06:12:26PM +0530, Srivatsa Vaddagiri wrote:
> I happened to read the entire thread (@ http://lkml.org/lkml/2007/3/1/159)
> all over again and felt it may be usefull to summarize the discussions so far.
>
> If I have missed any imp. points or falsely represented someone's view
> (unintentionally of course!), then I would be glad to be corrected.
>
> 1. Which task-grouping mechanism?
>
> [This question is the most vital one that needs a consensus]
>
> Resource management generally works by apply resource controls over a -group-
> of tasks (tasks of a user, tasks in a vserver/container etc).
>

> What mechanism do we use to group tasks for res mgmt purposes?
>
> Options:
>
> a. Paul Menage's container(/uh-a-diff-name-pls?) patches
>
> The patches introduce a new pointer in task_struct, struct
> container_group *containers, and a new structure 'struct container'.
>
> Tasks pointing to the same 'struct container' object (via their
> tsk->containers->container[] pointer) are considered to form
> a group associated with that container. The attributes associated
> with a container (ex: cpu_limit, rss_limit, cpus/mems_allowed) are
> decided by the options passed to mount command (which binds
> one/more/all resource controllers to a hierarchy).
>
> + For workload management, where it is desirable to manage resource
>   consumption of a run-time defined (potentially arbitrary) group of
>   tasks, then this patch is handy, as no existing pointers in
>   task_struct can be used to form such a run-time decided group.
>
> - (subjective!) If there is a existing grouping mechanism already (say
>   tsk->nsproxy[->pid_ns]) over which res control needs to be applied,
>   then the new grouping mechanism can be considered redundant (it can
>   eat up unnecessary space in task_struct)
>
>   What may help avoid this redundancy is to re-build existing
>   grouping mechanism (say tsk->nsproxy) using the container patches.
>   Serge however expressed some doubts on such a implementation
>   (for ex: how will one build hierarchical cpusets and non-hierarchical
>   namespaces using that single 'grouping' pointer in task_struct) and
>   also felt it may slow down things a bit from namespaces pov (more
>   dereferences reqd to get to a task's namespace).
>
> b. Reuse existing pointers in task_struct, tsk->nsproxy or better perhaps
>   tsk->nsproxy->pid_ns, as the means to group tasks (rcfs patches)
>
> This is based on the observation that the group of tasks whose resource
> consumption need to be managed is already defined in the kernel by
> existing pointers (either tsk->nsproxy or tsk->nsproxy->pid_ns)
>
> + reuses existing grouping mechanism in kernel
>
> - mixes resource and name spaces (?)
>
> c. Introduce yet-another new structure ('struct res_ctl?') which houses
>   resource control (& possibly pid_ns?) parameters and a new pointer to this
>   structure in task_struct (Herbert Poetzl).

>
>   Tasks that have a pointer to the same 'struct res_ctl' are considered
> to form a group for res mgmt purpose
>
> + Accessing res ctl information in scheduler fast path is
>   optimized (only two-dereferences required)
>
> - If all resource control parameters (cpu, memory, io etc) are
>   lumped together in same structure, it makes it hard to
>   have resource classes (cpu, mem etc) that are independent of
>   each other.
>
> - If we introduce several pointers in task_struct to allow
>   separation of resource classes, then it will increase storage space
>   in task_struct and also fork time (we have to take ref count
>   on more than one object now). Herbert thinks this is worthy
>   tradeoff for the benefit gained in scheduler fast paths.

what about identifying different resource categories and
handling them according to the typical usage pattern?

like the following:

 - cpu and scheduler related accounting/limits
 - memory related accounting/limits
 - network related accounting/limits
 - generic/file system related accounting/limits

I don't worry too much about having the generic/file stuff
attached to the nsproxy, but the cpu/sched stuff might be
better off being directly reachable from the task
(the memory related stuff might be placed in a zone or so)

> 2. Where do we put resource control parameters for a group?
>
> This depends on 1. So the options are:
>
> a. Paul Menage's patches:
>
> (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit
>
>   An optimized version of the above is:
> (tsk->containers->subsys[cpu_ctlr.subsys_id] - X)->cpu_limit
>
>
> b. rcfs
> tsk->nsproxy->ctlr_data[cpu_ctlr.subsys_id]->cpu_limit
>

> c. Herbert's proposal
> tsk->res_ctl->cpu_limit

see above, but yes ...

> 3. How are cpusets related to vserver/containers?
>
> Should it be possible to, lets say, create exclusive cpusets and
> attach containers to different cpusets?

that is what Linux-VServer does atm, i.e. you can put
an entire guest into a specific cpu set

> 4. Interface
> Filesystem vs system call
>
> Filesystem:
>   + natural way to represent hierarchical data
>   + File permission model convenient to delegate
>   management of part of a tree to one user
>   + Ease of use with scripts
>
>   (from Herbet Poetzl):
>
>   - performance of filesystem interfaces is quite bad
>   - you need to do a lot to make the fs consistant for
>   e.g. find and friends (regarding links and filesize)
>   - you have a quite hard time to do atomic operations
>   (except for the ioctl interface, which nobody likes)
>   - vfs/mnt namespaces complicate the access to this
>   new filesystem once you start moving around (between
>   the spaces)
>
>
> 5. If we use filesystem interface, then should it be in /proc? (Eric)
>
> - /proc doesn't allow the flexibility of say creating multiple
>   hierarchies and binding different resource controllers to each
>   hierarchy
>
> 6. As tasks move around namespaces/resource-classes, their
>   tsk->nsproxy/containers object will change. Do we simple create
>   a new nsproxy/containers object or optimize storage by searching
>   for one which matches the task's new requirements?
>
> - Linux Vserver follows former approach i.e simply creates
>   a new nsproxy with pointers to required namespace objects

which I consider suboptimal, but it was straight forward
to implement ...

> 7. Hierarchy
>
> - For res mgmt, do we need to worry about hierarchy at all?
>
> - If we consider cpuset to be a resource controller,
> then we have one resource controller who already
> supports hierarchy
>
> - If we don't support hierarchy in res controllers today
> but were to add that support later, then
> user-interface shouldn't change. That's why
> designining -atleast- the user interface to support
> hierarchy may make sense
>
> - Do we let resource classes to be split independent of each?
>
> For ex: CPU resource classes are independent of memory resource
> classes. This inturn affect whether the Paul Menage's patches
> need to support multiple hierarchy feature.

thanks,
Herbert

> --
> Regards,
> vatsa
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by Srivatsa Vaddagiri on Tue, 13 Mar 2007 17:58:20 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 05:24:59PM +0100, Herbert Poetzl wrote:
> what about identifying different resource categories and
> handling them according to the typical usage pattern?
>
> like the following:

>
> - cpu and scheduler related accounting/limits
> - memory related accounting/limits
> - network related accounting/limits
> - generic/file system related accounting/limits
>
> I don't worry too much about having the generic/file stuff
> attached to the nsproxy, but the cpu/sched stuff might be
> better off being directly reachable from the task

I think we should experiment with both combinations (a direct pointer to cpu_limit structure from task_struct and an indirect pointer), get some numbers and then decide. Or do you have results already with respect to that?

> > 3. How are cpusets related to vserver/containers?
> >
> > Should it be possible to, lets say, create exclusive cpusets and
> > attach containers to different cpusets?
>
> that is what Linux-VServer does atm, i.e. you can put
> an entire guest into a specific cpu set

Interesting. What abt /dev/cpuset view? Is that same for all containers or do you restrict that view to the containers cpuset only?

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by Herbert Poetzl on Tue, 13 Mar 2007 23:50:51 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 11:28:20PM +0530, Srivatsa Vaddagiri wrote:
> On Tue, Mar 13, 2007 at 05:24:59PM +0100, Herbert Poetzl wrote:
> > what about identifying different resource categories and
> > handling them according to the typical usage pattern?
> >
> > like the following:
> >
> > - cpu and scheduler related accounting/limits
> > - memory related accounting/limits

> > - network related accounting/limits
> > - generic/file system related accounting/limits
> >
> > I don't worry too much about having the generic/file stuff
> > attached to the nsproxy, but the cpu/sched stuff might be
> > better off being directly reachable from the task
>
> I think we should experiment with both combinations (a direct pointer
> to cpu_limit structure from task_struct and an indirect pointer), get
> some numbers and then decide. Or do you have results already with
> respect to that?

nope, no numbers for that, but I appreciate some testing
and probably can do some testing in this regard too
(although I want to get some testing done for the resource
 sharing between guests first)

> > > 3. How are cpusets related to vserver/containers?
> > >
> > > Should it be possible to, lets say, create exclusive cpusets and
> > > attach containers to different cpusets?
> >
> > that is what Linux-VServer does atm, i.e. you can put
> > an entire guest into a specific cpu set
>
> Interesting. What abt /dev/cpuset view?

host only for now

best,
Herbert

> Is that same for all containers or do you restrict that view
> to the containers cpuset only?
>
> --
> Regards,
> vatsa
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by Paul Menage on Thu, 15 Mar 2007 11:24:37 GMT
View Forum Message <> Reply to Message

On 3/12/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> - (subjective!) If there is a existing grouping mechanism already (say
> tsk->nsproxy[->pid_ns]) over which res control needs to be applied,
> then the new grouping mechanism can be considered redundant (it can
> eat up unnecessary space in task_struct)

If there really was a grouping that was always guaranteed to match the way you wanted to group tasks for e.g. resource control, then yes, it would be great to use it. But I don't see an obvious candidate. The pid namespace is not it, IMO. Resource control (and other kinds of task grouping behaviour) shouldn't require virtualization.

>
> a. Paul Menage's patches:
>
> (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit

Additionally, if we allow mature container subsystems to have an id declared in a global enum, then we can make the cpu_ctlr.subsys_id into a constant.

>
> b. rcfs
> tsk->nsproxy->ctlr_data[cpu_ctlr.subsys_id]->cpu_limit

So what's the '-X' that you're referring to

> 3. How are cpusets related to vserver/containers?
>
> Should it be possible to, lets say, create exclusive cpusets and
> attach containers to different cpusets?

Sounds reasonable.

>
> 6. As tasks move around namespaces/resource-classes, their
> tsk->nsproxy/containers object will change. Do we simple create
> a new nsproxy/containers object or optimize storage by searching
> for one which matches the task's new requirements?

I think the latter.

>
> - If we don't support hierarchy in res controllers today
> but were to add that support later, then

> user-interface shouldn't change. That's why
> designining -atleast- the user interface to support
> hierarchy may make sense

Right - having support for a hierarchy in the API doesn't mean that individual controllers have to support being in a hierarchy.

Paul

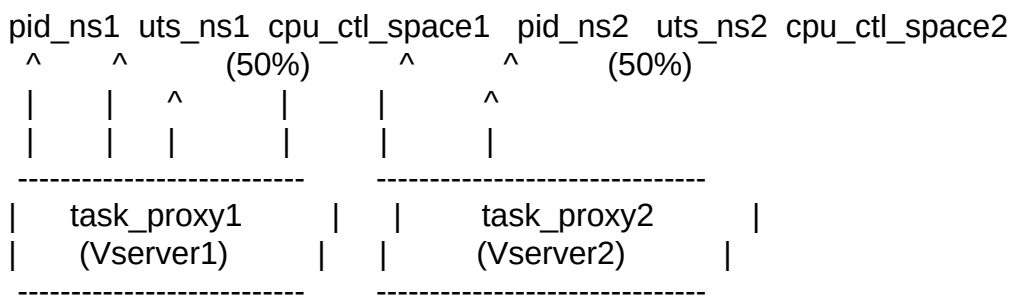
Subject: Re: Summary of resource management discussion
Posted by Srivatsa Vaddagiri on Thu, 15 Mar 2007 17:04:35 GMT
View Forum Message <> Reply to Message

On Thu, Mar 15, 2007 at 04:24:37AM -0700, Paul Menage wrote:
> If there really was a grouping that was always guaranteed to match the
> way you wanted to group tasks for e.g. resource control, then yes, it
> would be great to use it. But I don't see an obvious candidate. The
> pid namespace is not it, IMO.

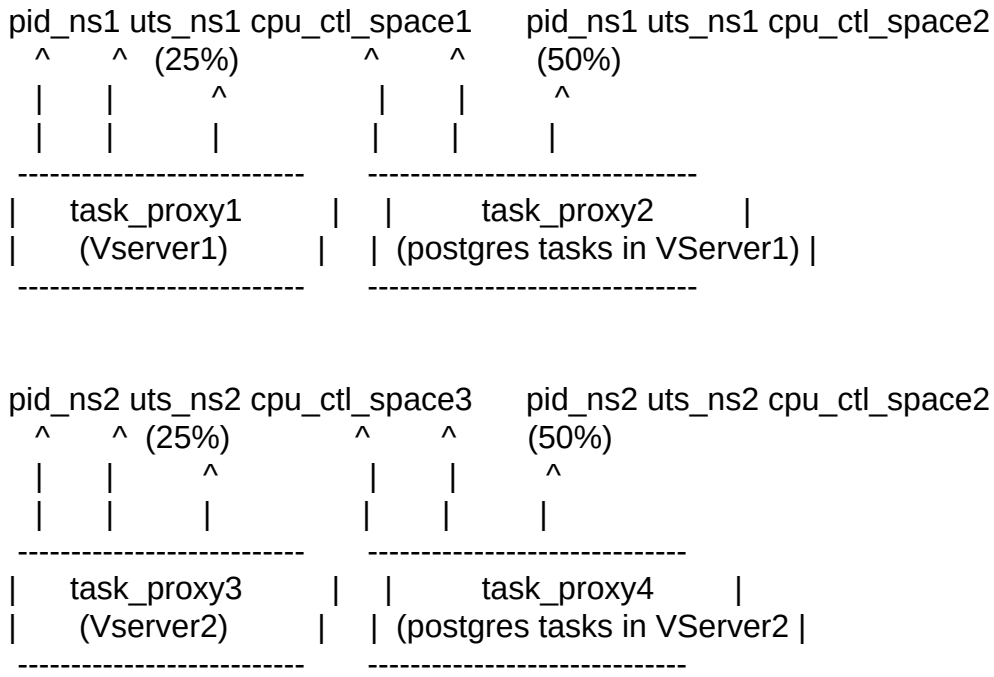
In vserver context, what is the "normal" case then? Atleast for Linux Vserver pid namespace seems to be normal unit of resource control (as per Herbert).

Even if one wanted to manage a arbitrary group of tasks in vserver context, IMHO its still possible to construct that arbitrary group using the existing pointer, ns[/task]proxy, and not break existing namespace semantics/functionality.

So the normal case I see is:



But, if someone wanted to manage cpu resource differently, and say that postgres tasks from both vservers should be in same cpu resource class, the above becomes:



(the best I could draw using ASCII art!)

The benefit I see of this approach is it will avoid introduction of additional pointers in struct task_struct and also additional structures (struct container etc) in the kernel, but we will still be able to retain same user interfaces you had in your patches.

Do you see any drawbacks of doing like this? What will break if we do this?

> Resource control (and other kinds of task grouping behaviour) shouldn't
> require virtualization.

Certainly. AFAICS, nsproxy[.c] is unconditionally available in the kernel (even if virtualization support is not enabled). When reused for pure resource control purpose, I see that as a special case of virtualization where only resources are virtualized and namespaces are not.

I think an interesting question would be : what more task-grouping behavior do you want to implement using an additional pointer that you can't reusing ->task_proxy?

> >a. Paul Menage's patches:
> >
> > (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit
>
> So what's the '-X' that you're referring to

Oh ..that's to seek pointer to begining of the cpulimit structure (subsys pointer in 'struct container' points to a structure embedded in a larger structure. -X gets you to point to the larger structure).

> >6. As tasks move around namespaces/resource-classes, their
> > tsk->nsproxy/containers object will change. Do we simple create
> > a new nsproxy/containers object or optimize storage by searching
> > for one which matches the task's new requirements?
>
> I think the latter.

Yes me too. But maybe to keep in simple in initial versions, we should avoid that optimisation and at the same time get statistics on duplicates?.

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by Paul Menage on Thu, 15 Mar 2007 19:12:50 GMT
View Forum Message <> Reply to Message

On 3/15/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> On Thu, Mar 15, 2007 at 04:24:37AM -0700, Paul Menage wrote:
> > If there really was a grouping that was always guaranteed to match the
> > way you wanted to group tasks for e.g. resource control, then yes, it
> > would be great to use it. But I don't see an obvious candidate. The
> > pid namespace is not it, IMO.
>
> In vserver context, what is the "normal" case then? Atleast for Linux
> Vserver pid namespace seems to be normal unit of resource control (as per
> Herbert).

Yes, for vserver the pid namespace is a good proxy for resource control groupings. But my point was that it's not universally suitable.

>
> (the best I could draw using ASCII art!)

Right, I think those diagrams agree with the point I wanted to make - that resource control shouldn't be tied to the pid namespace.

>
> The benefit I see of this approach is it will avoid introduction of
> additional pointers in struct task_struct and also additional structures
> (struct container etc) in the kernel, but we will still be able to retain
> same user interfaces you had in your patches.
>
> Do you see any drawbacks of doing like this? What will break if we do
> this?

There are some things that benefit from having an abstract
container-like object available to store state, e.g. "is this
container deleted?", "should userspace get a callback when this
container is empty?". But this indirection object wouldn't need to be
on the fast path for subsystem access to their per-taskgroup state.

> > >a. Paul Menage's patches:
> > >
> > > (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit
> >
> > So what's the '-X' that you're referring to
>
> Oh ..that's to seek pointer to begining of the cpulimit structure (subsys
> pointer in 'struct container' points to a structure embedded in a larger
> structure. -X gets you to point to the larger structure).

OK, so shouldn't that be listed as an overhead for your rcfs version
too? In practice, most subsystems that I've written tend to have the
subsys object at the beginning of the per-subsys state, so X = 0 and
is optimized out by the compiler. Even if it wasn't, X is constant and
so won't hurt much or at all.

>
> Yes me too. But maybe to keep in simple in initial versions, we should
> avoid that optimisation and at the same time get statistics on duplicates?.

That's an implementation detail - we have more important points to
agree on right now ...

Paul

Subject: Re: Summary of resource management discussion

On Thu, Mar 15, 2007 at 12:12:50PM -0700, Paul Menage wrote:
> There are some things that benefit from having an abstract
> container-like object available to store state, e.g. "is this
> container deleted?", "should userspace get a callback when this
> container is empty?".

IMO we can still get these bits of information using nsproxy itself (I admit I haven't looked at the callback requirement yet).

But IMO a bigger use of 'struct container' object in your patches is to store hierarchical information and avoid /repeating/ that information in each resource object (struct cpuset, struct cpu_limit, struct rss_limit etc) a 'struct container' is attached to (as pointed out here : http://lkml.org/lkml/2007/3/7/356). However I don't know how many controllers will ever support such hierarchical res mgmt and thats why I said option 3 [above URL] may not be a bad compromise.

Also if you find a good answer for my earlier question "what more task-grouping behavior do you want to implement using an additional pointer that you can't reusing ->task_proxy", it would drive home the need for additional pointers/structures.

> >> >a. Paul Menage's patches:
> >> >
> >> > (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit
> >>
> >> So what's the '-X' that you're referring to
> >
> >Oh ..that's to seek pointer to begining of the cpulimit structure (subsys
> >pointer in 'struct container' points to a structure embedded in a larger
> >structure. -X gets you to point to the larger structure).
>
> OK, so shouldn't that be listed as an overhead for your rcfs version
> too?

X shouldn't be needed in rcfs patches, because "->ctlr_data" in nsproxy can directly point to the larger structure (there is no 'struct container_subsys_state' equivalent in rcfs patches).

Container patches:

 (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit

rcfs:

 tsk->nsproxy->ctlr_data[cpu_ctlr.subsys_id]->cpu_limit

> >Yes me too. But maybe to keep in simple in initial versions, we should
> >avoid that optimisation and at the same time get statistics on duplicates?.
>
> That's an implementation detail - we have more important points to
> agree on right now ...

yes :)

Eric, did you have any opinion on this thread?

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by Herbert Poetzl on Fri, 16 Mar 2007 14:19:16 GMT
View Forum Message <> Reply to Message

On Thu, Mar 15, 2007 at 10:34:35PM +0530, Srivatsa Vaddagiri wrote:
> On Thu, Mar 15, 2007 at 04:24:37AM -0700, Paul Menage wrote:
> > If there really was a grouping that was always guaranteed to match the
> > way you wanted to group tasks for e.g. resource control, then yes, it
> > would be great to use it. But I don't see an obvious candidate. The
> > pid namespace is not it, IMO.
>
> In vserver context, what is the "normal" case then? Atleast for Linux
> Vserver pid namespace seems to be normal unit of resource control (as per
> Herbert).

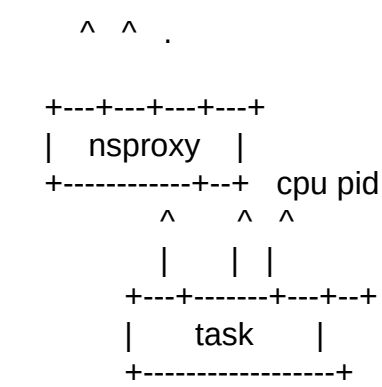
it is, but mainly because a 'context' in the Linux-VServer
case is a struct, which defines all the properties for a
'guest' excluding separate (name)spaces. as the pid space
doesn't exist yet, it is part of the task grouping context

> Even if one wanted to manage a arbitrary group of tasks in vserver
> context, IMHO its still possible to construct that arbitrary group using
> the existing pointer, ns[/task]proxy, and not break existing namespace
> semantics/functionality.
>
> So the normal case I see is:
>
> pid_ns1 uts_ns1 cpu_ctl_space1 pid_ns2 uts_ns2 cpu_ctl_space2

>   ^   ^   (50%)   ^   ^   (50%)
> | | ^ | | ^
> | | | | | |
>   --------------------------   -------------------------------
> | task_proxy1 | | task_proxy2 |
> | (Vserver1) | | (Vserver2) |
>   --------------------------   -------------------------------
>
>
> But, if someone wanted to manage cpu resource differently, and say that
> postgres tasks from both vservers should be in same cpu resource class,
> the above becomes:
>
>
>   pid_ns1 uts_ns1 cpu_ctl_space1   pid_ns1 uts_ns1 cpu_ctl_space2
>   ^   ^   (25%)   ^   ^   (50%)
> | | ^ | | ^
> | | | | | |
>   --------------------------   -------------------------------
> | task_proxy1 | | task_proxy2 |
> | (Vserver1) | | (postgres tasks in VServer1) |
>   --------------------------   -------------------------------
>
>
>   pid_ns2 uts_ns2 cpu_ctl_space3   pid_ns2 uts_ns2 cpu_ctl_space2
>   ^   ^ (25%)   ^   ^   (50%)
> | | ^ | | ^
> | | | | | |
>   --------------------------   -------------------------------
> | task_proxy3 | | task_proxy4 |
> | (Vserver2) | | (postgres tasks in VServer2 |
>   --------------------------   -------------------------------
>
> (the best I could draw using ASCII art!)
>
> The benefit I see of this approach is it will avoid introduction of
> additional pointers in struct task_struct and also additional structures
> (struct container etc) in the kernel, but we will still be able to retain
> same user interfaces you had in your patches.
>
> Do you see any drawbacks of doing like this? What will break if we do
> this?

looks good to me, except for the potential issue with
the double indirection introducing too much overhear
(compared to something like this:

      ipc uts



don't forget, accounting for cpu is probably very closely
tied to tasks, while this doesn't matter much for other
resources like number of tasks or file handles ...

> > Resource control (and other kinds of task grouping behaviour) shouldn't
> > require virtualization.
>
> Certainly. AFAICS, nsproxy[.c] is unconditionally available in the
> kernel (even if virtualization support is not enabled). When reused for
> pure resource control purpose, I see that as a special case of virtualization
> where only resources are virtualized and namespaces are not.
>
> I think an interesting question would be : what more task-grouping
> behavior do you want to implement using an additional pointer that you
> can't reusing ->task_proxy?
>
> > >a. Paul Menage's patches:
> > >
> > >   (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit
> >
> > So what's the '-X' that you're referring to
>
> Oh ..that's to seek pointer to begining of the cpulimit structure (subsys
> pointer in 'struct container' points to a structure embedded in a larger
> structure. -X gets you to point to the larger structure).
>
> > >6. As tasks move around namespaces/resource-classes, their
> > >   tsk->nsproxy/containers object will change. Do we simple create
> > >   a new nsproxy/containers object or optimize storage by searching
> > >   for one which matches the task's new requirements?
> >
> > I think the latter.
>
> Yes me too. But maybe to keep in simple in initial versions, we should
> avoid that optimisation and at the same time get statistics on duplicates?.

I agree here, just let us keep some way to actually

check _how_ much overhead we add with nsproxy, etc

best,
Herbert

> --
> Regards,
> vatsa
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by Herbert Poetzl on Fri, 16 Mar 2007 14:26:45 GMT
View Forum Message <> Reply to Message

On Thu, Mar 15, 2007 at 12:12:50PM -0700, Paul Menage wrote:
> On 3/15/07, Srivatsa Vaddagiri <vatsa@in.ibm.com> wrote:
> > On Thu, Mar 15, 2007 at 04:24:37AM -0700, Paul Menage wrote:
> > > If there really was a grouping that was always guaranteed to match
> > > the way you wanted to group tasks for e.g. resource control, then
> > > yes, it would be great to use it. But I don't see an obvious
> > > candidate. The pid namespace is not it, IMO.
> >
> > In vserver context, what is the "normal" case then? Atleast for
> > Linux Vserver pid namespace seems to be normal unit of resource
> > control (as per Herbert).
>
> Yes, for vserver the pid namespace is a good proxy for resource
> control groupings. But my point was that it's not universally
> suitable.
>
> >
> > (the best I could draw using ASCII art!)
>
> Right, I think those diagrams agree with the point I wanted to make -
> that resource control shouldn't be tied to the pid namespace.

first, strictly speaking they aren't (see previous mail)
it is more the lack of a separate pid space for now which
basically makes pid space == context, and in turn, the
resource limits are currently tied to the context too,

which again addresses the very same group of tasks

I'm fine with having a separate pid space, and another
(possibly different) cpu limit space or resource limit
space(s) as long as they do not complicate the entire
solution without adding any _real_ benefit ...

for example, it might be really nice to have a separate
limit for VM and RSS and MEMLOCK and whatnot, but IMHO
there is no real world scenario which would require you
to have those limits for different/overlaping groups
of tasks ... let me know if you have some examples

best,
Herbert

> > The benefit I see of this approach is it will avoid introduction
> > of additional pointers in struct task_struct and also additional
> > structures (struct container etc) in the kernel, but we will still
> > be able to retain same user interfaces you had in your patches.

> > Do you see any drawbacks of doing like this? What will break if we
> > do this?
>
> There are some things that benefit from having an abstract
> container-like object available to store state, e.g. "is this
> container deleted?", "should userspace get a callback when this
> container is empty?". But this indirection object wouldn't need to be
> on the fast path for subsystem access to their per-taskgroup state.
>
> > > >a. Paul Menage's patches:
> > > >
> > > > (tsk->containers->container[cpu_ctlr.subsys_id] - X)->cpu_limit
> > >
> > > So what's the '-X' that you're referring to
> >
> > Oh ..that's to seek pointer to begining of the cpulimit structure (subsys
> > pointer in 'struct container' points to a structure embedded in a larger
> > structure. -X gets you to point to the larger structure).
>
> OK, so shouldn't that be listed as an overhead for your rcfs version
> too? In practice, most subsystems that I've written tend to have the
> subsys object at the beginning of the per-subsys state, so X = 0 and
> is optimized out by the compiler. Even if it wasn't, X is constant and
> so won't hurt much or at all.
>
> >
> > Yes me too. But maybe to keep in simple in initial versions, we should

> > avoid that optimisation and at the same time get statistics on duplicates?.
>
> That's an implementation detail - we have more important points to
> agree on right now ...
>
> Paul
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by Srivatsa Vaddagiri on Fri, 16 Mar 2007 14:57:49 GMT
View Forum Message <> Reply to Message

On Fri, Mar 16, 2007 at 03:19:16PM +0100, Herbert Poetzl wrote:
> > Do you see any drawbacks of doing like this? What will break if we do
> > this?
>
> looks good to me, except for the potential issue with
> the double indirection introducing too much overhear

Sure. I plan to get some numbers with and without the indirection in
nsproxy.

I was planning to get these numbers with a prelimnary CPU controller I wrote a
while back (http://lkml.org/lkml/2006/9/28/236 and
http://lkml.org/lkml/2007/1/26/12). Do you have plans to publish any CPU
controller in the short term as well?

--
Regards,
vatsa

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Summary of resource management discussion
Posted by ebiederm on Fri, 16 Mar 2007 20:03:03 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri <vatsa@in.ibm.com> writes:

> On Thu, Mar 15, 2007 at 12:12:50PM -0700, Paul Menage wrote:
>> >Yes me too. But maybe to keep in simple in initial versions, we should
>> >avoid that optimisation and at the same time get statistics on duplicates?.
>>
>> That's an implementation detail - we have more important points to
>> agree on right now ...
>
> yes :)
>
> Eric, did you have any opinion on this thread?

A little. Sorry for the delay this is my low priority discussion to follow. Here comes a brain dump...

I do know of one case outside the context of containers/vservers where I would have liked a nice multi-process memory limit. HPC jobs doing rdma have a bad habit of wanting to mlock all of memory and thus send the memory allocator into a tail spin. So the separate use would be nice.

I do suspect that we are likely to look at having a hierarchy for most of the limits. For limits if we can stand the cost having a hierarchy makes sense. However I don't see the advantage of sharing the hierarchy between different classes of resource groups.

I do know that implementing hierarchical counters are inefficient for tracking resource consumption, and if we support deep hierarchies I expect we will want to optimize that, in non-trivial ways. At which point the hierarchy will be composed of more that just a pointer to it's parent. We could easily end up with children taking a lease from their parents saying you can have this many more before you look upwards in the hierarchy again. So with non-trivial hierarchy information adding a pointer at a generic level doesn't seem to be much of a help.

Further my gut feel is that in general we will want to limit all resources on a per container basis. At the same time I expect we will want to limit other resources to select process groups or users even farther inside of a container. So a single hierarchy for multiple resources seems a little questionable.

Which suggests that we want what I think was called multi-hierarchy support.

I guess also with hierarchies and entering we probably need a limit such that if you enter another resource group you can't do anything except stay at the same level or descend into the hierarchy. But

you can never remove your parent of that resource type.

With the whole shared subtree concept the mount namespace stores
things you can enter into in the mount namespace. This overcomes
the difficulty of having to find a process who has the resources
you want when you want to enter someplace. I think that concept
is a benefit of a filesystem interface. Using mount and unmount to
pin magic subsystem state seems a little more natural to me then
having to do other filesystem manipulations. Especially since
the mount namespace is reference counted so you can be certain
everything you have pinned will either remain visible to someone
or automatically disappear. I don't like interfaces like
sysvipc that require manual destruction. I do think there is some
sense in layout out a palette into the mount namespace we can enter
into.

There is another issue I'm not quite certain how to address. To some
extent it makes sense to be able to compile out the resource
controllers ensuring their overhead goes away completely. However for
the most part I think it makes sense to assume that when they are
compiled in we have an initial set of resource controllers that either
doesn't limit us at all or has very liberal limits that have the same
effect. Dealing with the case of possibly limiting things when we
have the support compiled in does not seem to make a lot of sense to
me.

I hope that helps a little. I'm still coming to terms with the issues
brought on by resource groups and controlling filesystems.

Eric

_____

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

---

Subject: Re: Summary of resource management discussion
Posted by Paul Jackson on Fri, 16 Mar 2007 21:23:56 GMT
View Forum Message <> Reply to Message

Herbert wrote:
> looks good to me, except for the potential issue with
> the double indirection introducing too much overhear

It's not the indirection count that I worry about.

It's the scalability of the locking. We must avoid as
much as possible adding any global locks on key code paths.

This means:
 1) be reluctant to add them to fork/exit
 2) just RCU locks on per-job (or finer grain) data when on
 the normal page allocation path
 3) nothing outside the current task context for the normal
 task scheduling code path.

A global lock on the wrong code path is fatal for scaling
big NUMA boxes.

... now whether or not that is an issue here, I don't claim
to know. I'm just worried that it could be.

Atomic data, such as global counters, is just as bad.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401