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