Subject: [patch 0/5]-Containers: Introduction
Posted by Rohit Seth on Fri, 15 Sep 2006 01:37:44 GMT
View Forum Message <> Reply to Message

Containers:

Commodity HW is becoming more powerful. This is giving opportunity to run different workloads on the same platform for better HW resource utilization. To run different workloads efficiently on the same platform, it is critical that we have a notion of limits for each workload in Linux kernel. Current cpuset feature in Linux kernel provides grouping of CPU and memory support to some extent (for NUMA machines).

We use the term container to indicate a structure against which we track and charge utilization of system resources like memory, tasks etc for a workload. Containers will allow system admins to customize the underlying platform for different applications based on their performance and HW resource utilization needs. Containers contain enough infrastructure to allow optimal resource utilization without bogging down rest of the kernel. A system admin should be able to create, manage and free containers easily.

At the same time, changes in kernel are minimized so as this support can be easily integrated with mainline kernel.

The user interface for containers is through configfs. Appropriate file system privileges are required to do operations on each container. Currently implemented container resources are automatically visible to user space through /configfs/container/<container_name> after a container is created.

Signed-off-by: Rohit Seth <rohitseth@google.com>

Diffstat for the patch (against linux-2.6.18-rc6-mm2):

```
diffstat patch.rc6mm2.0914.02
 Documentation/containers.txt |   42 ++
 fs/inode.c                   |    3
 include/linux/container.h    |  163 ++++++++++
 include/linux/fs.h           |    5
 include/linux/mm_inline.h    |    4
 include/linux/mm_types.h     |    4
 include/linux/sched.h        |    6
 kernel/Makefile              |    1
 kernel/container_configfs.c  |  426 ++++++++++++++++++++++++++++++++++++++++++
 kernel/exit.c                |    2
 kernel/fork.c                |    8
```

```
 mm/Kconfig              |    8
 mm/Makefile             |    2
 mm/container.c          |  620 ++++++++++++++++++++++++++++++++++++++++++++++++
 mm/container_mm.c       |  455 ++++++++++++++++++++++++++++++++++++++
 mm/filemap.c            |    4
 mm/page_alloc.c         |    3
 mm/rmap.c               |    8
 mm/swap.c               |    1
 mm/vmscan.c             |    1
 20 files changed, 1765 insertions(+), 1 deletion(-)
```

This patch set has basic container support that includes:

- Create a container using mkdir command in configfs

- Free a container using rmdir command

- Dynamically adjust memory and task limits for container.

- Add/Remove a task to container (given a pid)

- Files are currently added as part of open from a task that already belongs to a container.

- Keep track of active, anonymous, mapped and pagecache usage of container memory

- Does not allow more than task_limit number of tasks to be created in the container.

- Over the limit memory handler is called when number of pages (anon + pagecache) exceed the limit. It is also called when number of active pages exceed the page limit. Currently, this memory handler scans the mappings and tasks belonging to container (file and anonymous) and tries to deactivate pages. If the number of page cache pages is also high then it also invalidate mappings. The thought behind this scheme is, it is okay for containers to go over limit as long they run in degraded manner when they are over their limit. Also, if there is any memory pressure then pages belonging to over the limit container(s) become prime candidates for kernel reclaimer. Container mutex is also held during the time this handler is working its way through to prevent any further addition of resources (like tasks or mappings) to this container. Though it also blocks removal of same resources from the container for the same time. It is possible that over the limit page handler takes lot of time if memory pressure on a container is continuously very high. The limits, like how long a task should schedule out when it hits memory limit, is also on the lower side at

present (particularly when it is memory hogger). But should be easy to change if need be.

- Indicate the number of times the page limit and task limit is hit

Below is a one line description for patches that will follow:

[patch01]: Documentation on how to use containers (Documentation/container.txt)

[patch02]: Changes in the generic part of kernel code

[patch03]: Container's interface with configfs

[patch04]: Core container support

[patch05]: Over the limit memory handler.

TODO:
1- mm/Kconfig is not the best place to set the CONFIG_CONTAINERS option.
2- some code(like container_add_task) in mm/container.c should go elsewhere.
3- Support adding/removing a file name to container through configfs
4- /proc/pid/container to show the container id (or name)
5- Wider testing for memory controller. Currently it is possible that limits are exceeded. See if a call to reclaim can be easily integrated.
6- Kernel memory tracking (based on patches from BC)
7- Limit on user locked memory
8- Huge memory support
9- Stress testing with containers
10- One shot view of all containers
11- CKRM folks are interested in seeing all processes belonging to a container. Add the attribute show_tasks to container.
12- Add logic so that the sum of limits are not exceeding appropriate system requirements.
13- Extend it with other controllers (CPU and Disk I/O)
14- Add flags bits for supporting different actions (like in some cases provide a hard memory limit and in some cases it could be soft).
15- Capability to kill processes for the extreme cases.
16- ...

This is based on lot of discussions over last month or so. I hope this patch set is something that we can agree and more support can be added on top of this. Please provide feedback and add other extensions that are useful in the TODO list.

Thanks,
-rohit