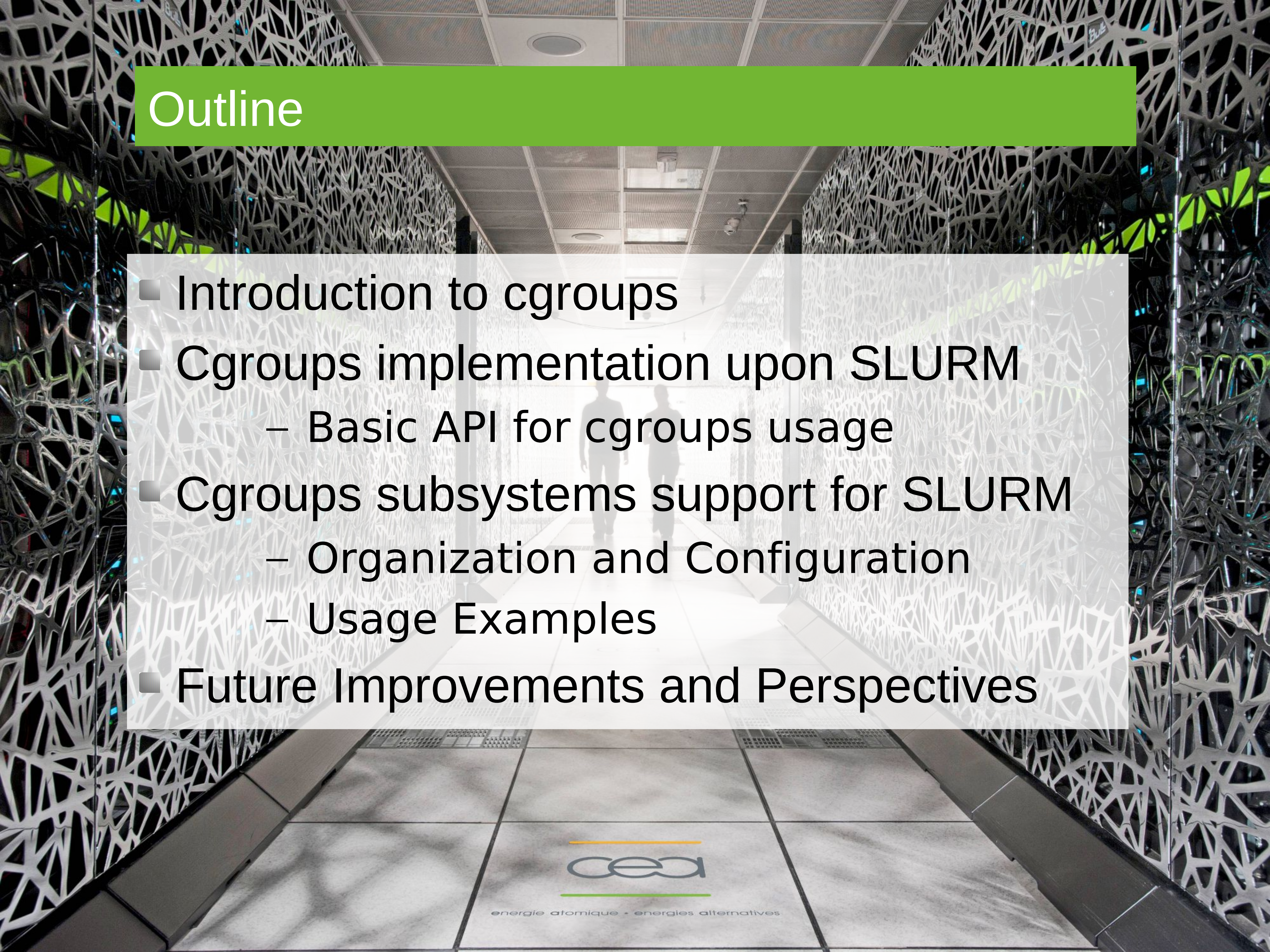
SLURM Resources isolation through cgroups

Yiannis Georgiou
email: yiannis.georgiou@bull.fr
Matthieu Hautreux
email: matthieu.hautreux@cea.fr

Architect of an Open World



Outline

- Introduction to cgroups
- Cgroups implementation upon SLURM
  - Basic API for cgroups usage
- Cgroups subsystems support for SLURM
  - Organization and Configuration
  - Usage Examples
- Future Improvements and Perspectives

Introduction to cgroups

Control Groups (cgroups) is a **Linux kernel mechanism** (appeared in 2.6.24) to limit, isolate and monitor resource usage (CPU, memory, disk I/O, etc.) of groups of processes.

Features

- ***Resource Limiting*** (i.e. not to exceed a memory limit)
- ***Prioritization*** (i.e. groups may have larger share of CPU)
- ***Isolation*** (i.e. isolate GPUs for particular processes)
- ***Accounting*** (i.e. montior resource usage for processes)
- ***Control*** (i.e. suspending and resuming processes)

Cgroups Model and Concepts

Model

Cgroups **similar** to Linux processes:

- Hierarchical
- Inheritance of attributes from parent to child

but **different** because:

- **multiple hierarchies** of cgroups may exist that are attached to one or more subsystems

Concepts

Cgroup – a group of processes with the same characteristics

- **Subsystem** – a module that applies parameters to a group of processes (cgroup)
- **Hierarchy** – a set of cgroups organized in a tree, plus one or more subsystems associated with that tree

Cgroups subsystems

- **cpuset** – assigns tasks to individual CPUs and memory nodes in a cgroup
- **cpu** – schedules CPU access to cgroups
- **cpuacct** – reports CPU resource usage of tasks of a cgroup
- **memory** – set limits on memory use and reports memory usage for a cgroup
- **devices** – allows or denies access to devices (i.e. gpus) for tasks of a cgroup
- **freezer** – suspends and resumes tasks in a cgroup
- **net_cls** – tags network packets in a cgroup to allow network traffic priorities
- **ns** – namespace subsystem
- **blkio** – tracks I/O ownership, allowing control of access to block I/O resources

Cgroups functionality rules

•Cgroups are represented as **virtual file systems**

- Hierarchies are directories, created by mounting subsystems, using the mount command; subsystem names specified as mount options
- Subsystem parameters are represented as files in each hierarchy with values that apply only to that cgroup

•**Interaction with cgroups** take place by manipulating directories and files in the cgroup virtual file system using standard shell commands and system calls (mkdir, mount, echo, etc)

- *tasks* file in each cgroup directory lists the tasks (pids) in that cgroup
- Tasks are automatically removed from a cgroup when they terminate or are added to a different cgroup in the same hierarchy
- Each task is present in only one cgroup in each hierarchy

•Cgroups have a mechanism for **automatic removal** of abandoned cgroups (release_agent)

Cgroups subsystems parameters

<u>cpuset subsystem</u>

cpuset.cpus: defines the set of cpus that the tasks in the cgroup are allowed to execute on

cpuset.mems: defines the set of memory zones that the tasks in the cgroup are allowed to use

<u>memory subsystem</u>

memory.limit_in_bytes: defines the memory limit for the tasks in the cgroup

memory.swappiness: controls kernel reclamation of memory from the tasks in the cgroup (swap priority)

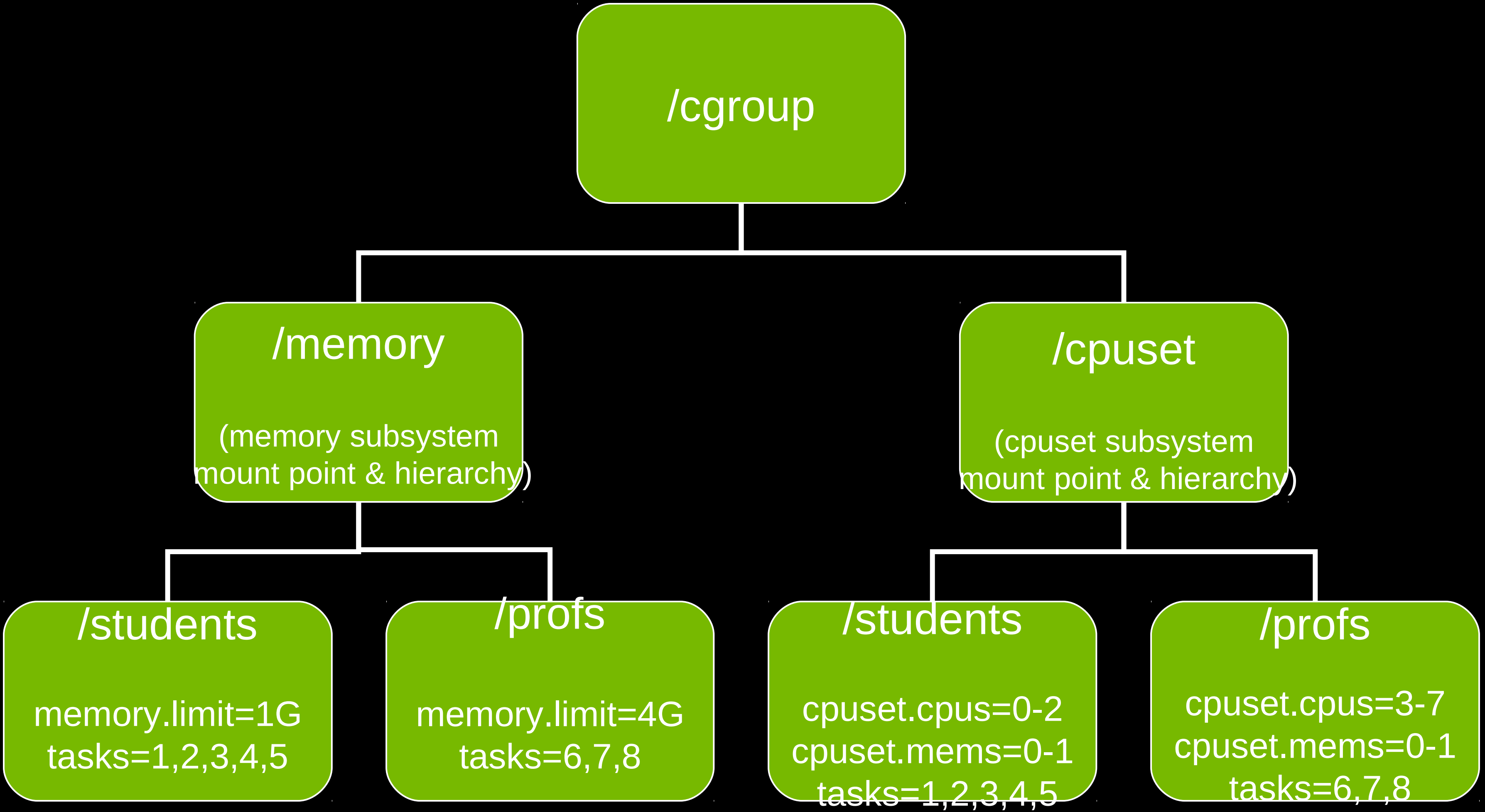
<u>freezer subsystem</u>

freezer.state: controls whether tasks in the cgroup are active (runnable) or suspended

<u>devices subsystem</u>

devices_allow: specifies devices to which tasks in a cgroup have acces

Cgroups functionality example



Cgroups functionality example

```
[root@mordor:~]# mkdir /cgroup
[root@mordor:~]# mkdir /cgroup/cpuset
[root@mordor:~]# mount -t cgroup -o cpuset none /cgroup/cpuset
[root@mordor:~]# ls /cgroup/cpuset/
cpuset.cpus  cpuset.mems tasks notify_on_release release_agent
[root@mordor:~]# mkdir /cgroup/cpuset/students
[root@mordor:~]# mkdir /cgroup/cpuset/profs
[root@mordor:~]# echo 0-2 > /cgroup/cpuset/students/cpuset.cpus
[root@mordor:~]# echo 0 > /cgroup/cpuset/students/cpuset.mems
[root@mordor:~]# echo $PIDS_st > /cgroup/cpuset/students/tasks
[root@mordor:~]# echo 3-7 > /cgroup/cpuset/profs/cpuset.cpus
[root@mordor:~]# echo 1 > /cgroup/cpuset/profs/cpuset.mems
[root@mordor:~]# echo $PIDS_pr > /cgroup/cpuset/profs/tasks
```

Why support cgroups on SLURM?

- To guarantee that every consumed resources is consumed the way it's planned to be (Authorization and Accounting of AAA)
  - leveraging linux latest features in terms of processes control and resource management
  - Enhancing nodes sharing (at the same time or sequentially)
- While enhancing SLURM connection with Linux systems
  - Improved **tasks isolation** upon resources
  - Improved **efficiency** of Slurm activities (e.g., process tracking, collection of accounting statistics)
  - Improved **robustness** (e.g. more reliable cleanup of jobs)
- And simplifying the addition of **new controled resources and features**
  - like management of network bandwidth or disks I/O as individual resources

Cgroups implementation upon SLURM

- A common API to manage cgroup hierarchies, directories and files
  - src/common/xcgroup.{h,c}
  - src/common/xcgroup_read_config.{h,c}

- A uniform syntax to declare slurm related cgroup subsystems directories
  - %cgroup_subsys_mount_point%/uid_%uid/job_%jobid/step_%stepid/

- A dedicated cgroup release_agent and subsystems release_agent naming schema
  - Lock/Unlock cgroup hierarchy when managing slurm related cgroups to avoid race conditions
  - Update uid_%uid entry to match subdirectory configurations

- 2 plugins that add cgroup related features to slurmd
  - Proctrack/cgroup : to track/suspend/resume job's tasks
  - Task/cgroup : to confine tasks to the allocated resources

Cgroups API

Ease cgroup init, directories and files management

- slurm_cgroup_conf_t
  - Stores cgroup related conf
  -
- xcgroup_ns_t
  - Structure associated to a cgroup hierarchy
  - Helps to initialize/mount/umount/search_into it
- xcgroup_t
  - Structure associated to a cgroup directory
  - Linked to the associated xcgroup_ns
  - Helps to add/get tasks, set/get params
  - Helps to lock/unlock the underlying directory

Cgroup **Proctrack** plugin: **freezer** subsystem

Track jobs processes using the freezer subsystem

- Every spawned process is tracked
  - Automatic inheritance of parent's cgroup
  - No way to escape the container
- Every processes can be frozen
  - Using the Thawed|Frozen state of the subsystem
  - No way to avoid the freeze action

Cgroup **Proctrack** plugin: **freezer** subsystem

```
[mat@leaf slurm]$ srun  sleep 300
```

```
[root@leaf ~]# cat /cgroup/freezer/uid_500/job_53/step_0/freezer.state
THAWED
[root@leaf ~]# scontrol suspend 53
[root@leaf ~]# ps -ef f | tail -n 2
root      15144      1  0 17:10 ?        Sl     0:00 slurmstepd: [53.0]
mat       15147 15144  0 17:10 ?        T      0:00  \_ /bin/sleep 300
[root@leaf ~]# cat /cgroup/freezer/uid_500/job_53/step_0/freezer.state
FREEZING
[root@leaf ~]# scontrol resume 53
[root@leaf ~]# ps -ef f | tail -n 2
root      15144      1  0 17:10 ?        Sl     0:00 slurmstepd: [53.0]
mat       15147 15144  0 17:10 ?        S      0:00  \_ /bin/sleep 300
[root@leaf ~]# cat /cgroup/freezer/uid_500/job_53/step_0/freezer.state
THAWED
[root@leaf ~]#
```

Cgroup **Task** plugin

Constrain jobs tasks to the allocated resources

- 3 independant layers of managed resources using 3 subsystems
  - Cores (**cpuset**), Memory (**memory**), GRES (**devices**)
- Every spawned process is tracked
  - Automatic inheritance of parent's cgroup
  - No way to use additional resources
- Each layer has its own additional parameters
- More resources could be added in the future

Cgroup **Task** plugin : **cpuset** subsystem

Constrain jobs tasks to the allocated cores

- Configurable feature
  - ConstrainCores=yes|no
- Use step's allocated cores with “exclusive steps”
  - Otherwise, let steps use job's allocated cores
- Basic affinity management as a configurable sub-feature
  - TaskAffinity=yes|no in cgroup.conf (rely on HWLOC)
  - Automatic block distribution of tasks only
    - Cyclic distribution not yet implemented

Cgroup **Task** plugin : **cpuset** subsystem

TaskAffinity binding logic

- Detect the number of allocated cores
- Look at the requested binding (--cpu_binding)
- Use the granularity that best matches the allocated resources versus the number of tasks to spawn * the req cores per task
  - If sockets are requested but less sockets than tasks, automatically switch to cores (1)
  - If cores are requested but less cores than required, automatically switch to PU (hyperthread) (2)
  - If less PU than required, disable affinity

Cgroup **Task** plugin : **cpuset** subsystem

TaskAffinity binding logic (following)

- Distribute the associated objects to tasks (socket|core(|PU))
- Relax constraint to match the requested binding if necessary
  - In (1), each task is then allowed to access other cores sharing the partial sockets already allowed
  - In (2), each task is then allowed to access the other hyperthreads sharing the partial cores already allowed

Cgroup **Task** plugin : **cpuset** subsystem

TaskAffinity binding logic in brief

- Distribute allocated cores to tasks in a basic block distribution
- If cpu_binding > cores (socket,ldom)
  - Allow adjacent cores on already allocated object to be used by task

- Need further optimization to provide more features
  - Cyclic distribution
  - Best-fit selection of cores to associate to tasks

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n1 --cpu_bind=none sleep 3000
salloc: Granted job allocation 55
```

```
 [root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:24:59] [55.0] task/cgroup: now constraining jobs allocated cores
[2011-09-16T17:24:59] [55.0] task/cgroup: loaded
[2011-09-16T17:24:59] [55.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:24:59] [55.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:24:59] [55.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:24:59] [55.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:24:59] [55.0] task/cgroup: task[0] is requesting no affinity
```

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n1 –exclusive --cpu_bind=none sleep 3000
salloc: Granted job allocation 56
```

```
 [root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:29:25] [56.0] task/cgroup: now constraining jobs allocated cores
[2011-09-16T17:29:25] [56.0] task/cgroup: loaded
[2011-09-16T17:29:25] [56.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:29:25] [56.0] task/cgroup: step abstract cores are '0'
[2011-09-16T17:29:25] [56.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:29:25] [56.0] task/cgroup: step physical cores are '0'
[2011-09-16T17:29:25] [56.0] task/cgroup: task[0] is requesting no affinity
```

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n1 --cpu_bind=cores sleep 3000
salloc: Granted job allocation 57
```

```
 [root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:31:17] [57.0] task/cgroup: now constraining jobs allocated cores
[2011-09-16T17:31:17] [57.0] task/cgroup: loaded
[2011-09-16T17:31:17] [57.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:31:17] [57.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:31:17] [57.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:31:17] [57.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:31:17] [57.0] task/cgroup: task[0] is requesting core level binding
[2011-09-16T17:31:17] [57.0] task/cgroup: task[0] using Core granularity
[2011-09-16T17:31:17] [57.0] task/cgroup: task[0] taskset '0x00000001' is set
```

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n1 --cpu_bind=socket sleep 3000
salloc: Granted job allocation 58
```

```
 [root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:33:31] [58.0] task/cgroup: now constraining jobs allocated cores
[2011-09-16T17:33:31] [58.0] task/cgroup: loaded
[2011-09-16T17:33:31] [58.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:33:31] [58.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:33:31] [58.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:33:31] [58.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:33:31] [58.0] task/cgroup: task[0] is requesting socket level binding
[2011-09-16T17:33:31] [58.0] task/cgroup: task[0] using Socket granularity
[2011-09-16T17:33:31] [58.0] task/cgroup: task[0] taskset '0x00000003' is set
```

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n1 --cpu_bind=ldom sleep 3000
salloc: Granted job allocation 59
```

```
 [root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:34:50] [59.0] task/cgroup: now constraining jobs allocated cores
[2011-09-16T17:34:50] [59.0] task/cgroup: loaded
[2011-09-16T17:34:50] [59.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:34:50] [59.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:34:50] [59.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:34:50] [59.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:34:50] [59.0] task/cgroup: task[0] is requesting ldom level binding
[2011-09-16T17:34:50] [59.0] task/cgroup: task[0] using Core granularity
[2011-09-16T17:34:50] [59.0] task/cgroup: task[0] higher level Machine found
[2011-09-16T17:34:50] [59.0] task/cgroup: task[0] taskset '0x00000003' is set
```

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n2 --cpu_bind=socket sleep 3000
salloc: Granted job allocation 60
```

```
 [root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup[2011-09-16T17:36:18] [60.0] task/cgroup:
 now constraining jobs allocated cores
[2011-09-16T17:36:18] [60.0] task/cgroup: loaded
[2011-09-16T17:36:18] [60.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:36:18] [60.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:36:18] [60.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:36:18] [60.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:36:18] [60.0] task/cgroup: task[0] is requesting socket level binding
[2011-09-16T17:36:18] [60.0] task/cgroup: task[1] is requesting socket level binding
[2011-09-16T17:36:18] [60.0] task/cgroup: task[1] using Core granularity
[2011-09-16T17:36:18] [60.0] task/cgroup: task[1] higher level Socket found
[2011-09-16T17:36:18] [60.0] task/cgroup: task[1] taskset '0x00000003' is set
[2011-09-16T17:36:18] [60.0] task/cgroup: task[0] using Core granularity
[2011-09-16T17:36:18] [60.0] task/cgroup: task[0] higher level Socket found
[2011-09-16T17:36:18] [60.0] task/cgroup: task[0] taskset '0x00000003' is set
```

Cgroup **Task** plugin : **memory** subsystem

Constrain jobs tasks to the allocated amount of memory

- Configurable feature
  - ConstrainRAMSpace=yes|no
  - ConstrainSwapSpace=yes|no
- Use step's allocated amount of memory with “exclusive steps”
  - Otherwise, let steps use job's allocated amount
- Both RSS and swap are monitored
- Trigger OOM killer on the cgroup's tasks when reaching limits
- Tolerant mechanism
  - AllowedRAMSpace , AllowedSwapSpace percents

Cgroup **Task** plugin : **memory** subsystem

```
[mat@leaf slurm]$ salloc --exclusive --mem-per-cpu 100 srun -n1 sleep 3000
salloc: Granted job allocation 67
```

```
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:55:20] [67.0] task/cgroup: now constraining jobs allocated memory
[2011-09-16T17:55:20] [67.0] task/cgroup: loaded
[2011-09-16T17:55:20] [67.0] task/cgroup: job mem.limit=3520MB memsw.limit=3840MB
[2011-09-16T17:55:20] [67.0] task/cgroup: step mem.limit=3520MB memsw.limit=3840MB
```

```
[mat@leaf slurm]$ salloc --exclusive --mem-per-cpu 100 srun –exclusive -n1 sleep 3000
salloc: Granted job allocation 68
```

```
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:57:31] [68.0] task/cgroup: now constraining jobs allocated memory
[2011-09-16T17:57:31] [68.0] task/cgroup: loaded
[2011-09-16T17:57:31] [68.0] task/cgroup: job mem.limit=3520MB memsw.limit=3840MB
[2011-09-16T17:57:31] [68.0] task/cgroup: step mem.limit=110MB memsw.limit=120MB
```

Cgroup **Task** plugin : **memory** subsystem

Limitations

- Automatic cleaning of cgroup directories
  - when last byte is unattached
  - Can take a long long long time
- Performances penalities on some systems
  - Depending on the kernel/cgroup version
  - Depending on the NUMA architecture of the nodes
- Next step
  - Detect and advertise OOM killer triggers

Cgroup **Task** plugin : **devices** subsystem

Constrain jobs tasks to the allocated system devices

- Based on the **GRES** allocated resources and built upon the cgroup task plugin
  - Each task is allowed to access to a number of devices by default (disk,network,etc)
  - Only the tasks that have granted allocation on the **GRES** devices will be allowed to have access on them.
  - Tasks with no granted allocation upon **GRES** devices will not be able to use them.

Cgroup **Task** plugin : **devices** subsystem

Configuration

1) Gres

- Configure gres plugin in the **slurm.conf**
- Create a **gres.conf** file on each computing node describing its resources

2) Cgroup Devices

- Configure **cgroup.conf** file to constrain devices
- Create file **allowed_devices.conf** file on each computing node listing the devices that the system should allow by default for all tasks

Cgroup **Task** plugin : **devices** subsystem

GRES Configuration Example

```
[root@mordor cgroup]# egrep "Gres" /etc/slurm/slurm.conf
GresTypes=gpu
NodeName=mordor NodeAddr=127.0.0.1 Gres=gpu:2 Sockets=1 Cor...
```

```
[root@mordor cgroup]# cat /etc/slurm/gres.conf
Name=gpu File=/dev/nvidia0
Name=gpu File=/dev/nvidia1
```

Note:
To declare different slurm.conf between nodes and controller you need to use option Debug_Flags=NO_CONF_HASH

Cgroup **Task** plugin : **devices** subsystem

Cgroup Devices Configuration Example

```
[root@mordor cgroup]# egrep "Devices" /etc/slurm/cgroup.conf
ConstrainDevices=yes
AllowedDevicesFile="/etc/slurm/allowed_devices.conf"
```

```
[root@mordor cgroup]# cat /etc/slurm/allowed_devices.conf
/dev/sda*
/dev/null
/dev/zero
/dev/urandom
/dev/cpu/*/*
```

Cgroup **Task** plugin : **devices** subsystem

Cgroup Devices Logic as implemented in task plugin

1) Initialization phase (information collection gres.conf file, major, minor, etc)

2) Allow all devices that should be allowed by default (allowed_devices.conf)

3) Lookup which gres devices are allocated for the job

- Write allowed gres devices to devices.allow file
- Write denied gres devices to devices.deny file

4) Execute **2** and **3** for job and steps tasks (different hierarchy level in cgroups)

Cgroups **devices** subsystem : Usage Example

```
[root@mordor cgroup]# egrep "Gres" /etc/slurm/slurm.conf
GresTypes=gpu
NodeName=cuzco[57,61] Gres=gpu:2 Procs=8 Sockets=2 CoresPerSocket=4
```

```
[root@cuzco51]# cat /etc/slurm/allowed_devices.conf
/dev/sda*
/dev/null
```

```
[gohn@cuzco0]$ cat gpu_test.sh
#!/bin/sh
sleep 10
echo 0 > /dev/nvidia0
echo 0 > /dev/nvidia1
```

Cgroups **devices** subsystem : Usage Example

```
[gohn@cuzco0]$ srun -n1 –gres=gpu:1 -o output ./gpu_test.sh
```

```
[root@cuzco51 ~]# tail -f /var/log/slurmd.cuzco51.log
[2011-09-20T03:10:02] [22.0] task/cgroup: manage devices jor job '22'
[2011-09-20T03:10:02] [22.0] device : /dev/nvidia0 major 195, minor 0
[2011-09-20T03:10:02] [22.0] device : /dev/nvidia1 major 195, minor 1
[2011-09-20T03:10:02] [22.0] device : /dev/sda2 major 8, minor 2
[2011-09-20T03:10:02] [22.0] device : /dev/sda1 major 8, minor 1
[2011-09-20T03:10:02] [22.0] device : /dev/sda major 8, minor 0
[2011-09-20T03:10:02] [22.0] device : /dev/null major 1, minor 3
[2011-09-20T03:10:02] [22.0] Default access allowed to device b 8:2 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'b 8:2 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Default access allowed to device b 8:1 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'b 8:1 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Default access allowed to device b 8:0 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'b 8:0 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Default access allowed to device c 1:3 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'c 1:3 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Allowing access to device c 195:0 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'c 195:0 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Not allowing access to device c 195:1 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.deny' set to 'c 195:1 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
```

Cgroups **devices** subsystem : Usage Example

```
[root@cuzco51 ~]# cat /cgroup/devices/uid_50071/job_22/step_0/tasks
4875
4879
4882
[root@cuzco51 ~]# cat /cgroup/devices/uid_50071/job_22/step_0/devices.list
b 8:2 rwm
b 8:1 rwm
b 8:0 rwm
c 1:3 rwm
c 195:0 rwm
```

```
[gohn@cuzco0]$ cat output
/home/GPU/./gputest.sh: line 4: echo: write error: Invalid argument
/home/GPU/./gputest.sh: line 5: /dev/nvidia1: Operation not permitted
```

Future Improvements: Cgroup task plugin

- User oriented cgroup hierarchies :
  - For the moment, only jobs and steps hierarchies are constrained. A user directory level is currently defined and could be used as the entry point to attach tasks to allowed resources
  - Main technical problem : how to merge resources from multiple directories in the cgroup hierarchies into a virtual directory automatically (no way to do that, a manual approach is used, but it's hard to keep things in sync -> slurm releage_agent)
    - Only done for cpusets subsystem in current version (slurm-2.3.0)

- A PAM module to leverage the user cgroup and help system daemons to bind user 's tasks to the locally allocated resources only
  - Eg : OpenSSH would use that PAM module to only allow remote log in to allocated resources
  - Eg : MPI implementations not aware of SLURM (using ssh) could be confined

Future Improvements: **devices** subsystem

- Improvements in cgroup/devices subsystem have been proposed to the kernel developers. The most important is related with the function of devices as whitelist and not as blacklist. The second would ease the procedure and no allowed_devices.conf file would be required.

Upcoming cgroups subsystem support: **cpuacct**

Monitoring cpu usage with cpuacct subsystem

(Already implemented, currently under testing)

- Implemented within job_acct_gather plugin of SLURM
- Collects information concerning the CPU cycles and the CPU time consumed for each task of the cgroup
- CPU cycles of a step are reported as a new attribute in the accounting database of SLURM
  - Value can be used for billing purposes

Bugs:
cpuacct.stat parameter which is supposed to collect CPU cycles actually returns CPU time and not cycles (bug reported)

Future Works

Limit the usage of disk and network bandwidth

- Control access to **I/O on hard disks** for tasks in cgroups through **blkio** subsystem
  - By specifying relative proportion (blkio.weight) of I/O access of devices available to a cgroup through the blkio.weight parameter with range from 100 to 1000
- Limit the **network bandwidth** for tasks in cgroups through **net_cls** subsystem
  - By specifying particular ids (net_cls.classids) and configure them appropriately through the filtering capabilities of the Linux network stack (tc command) to provide particular network bandwith to each cgroup
- Implementation as new parameters in the **task cgroup plugin**
- **Limitations: net_cls** currently works only for ethernet (not for infiniband) and **blkio** would work only for local hard disks

Future Works

Monitor and report the usage of additional resources

- Monitor and report **I/O access on hard disks** for tasks in cgroups **blkio subsystem**
  - Report may contain I/O time and I/O bytes transfered
- Monitor and report **memory usage** for tasks in cgroups **memory subsystem**
  - Report may be the max memory and the sum of memory plus swap used by tasks in cgroups
- Implementation in the **job_acct_gather plugin**

THANK YOU

Questions?