Subject: [RFC][v2][patch 0/12][CFQ-cgroup]Yet another I/O bandwidth controlling subsystem for CGroups based o
Posted by Satoshi UCHIDA on Thu, 03 Apr 2008 07:09:12 GMT

This patchset modified a name of subsystem (from "cfq_cgroup" to "cfq")
and a checking in create function.

This patchset introduce "Yet Another" I/O bandwidth controlling
subsystem for cgroups based on CFQ (called 2 layer CFQ).

The idea of 2 layer CFQ is to build fairness control per group on the top of existing CFQ control.
We add a new data structure called CFQ meta-data on the top of
cfqd in order to control I/O bandwidth for cgroups.
CFQ meta-data control cfq_datas by service tree (rb-tree) and
CFQ algorithm when synchronous I/O.
An active cfqd controls queue for cfq by service tree.
Namely, the CFQ meta-data control traditional CFQ data.
the CFQ data runs conventionally.

```
       cfqmd    cfqmd    (cfqmd = cfq meta-data)
         |       |
  cfqc  -- cfqd ----- cfqd    (cfqd = cfq data,
         |       |        cfqc = cfq cgroup data)
  cfqc  --[cfqd]----- cfqd
          $B", (B
 $B!!!!!!!!!! (Bconventional control.
```


This patchset is gainst 2.6.25-rc2-mm1.

Last week, we found a patchset from Vasily Tarasov (Open VZ) that
posted to LKML.
  [RFC][PATCH 0/9] cgroups: block: cfq: I/O bandwidth controlling subsystem for CGroups based
on CFQ
  http://lwn.net/Articles/274652/

Our subsystem and Vasily's one are similar on the point of modifying
the CFQ subsystem, but they are different on the point of the layer of
implementation. Vasily's subsystem add a new layer for cgroup between
cfqd and cfqq, but our subsystem add a new layer for cgroup on the top
of cfqd.

The different of implementation from OpenVZ's one are:
    * top layer algorithm is also based on service tree, and
    * top layer program is stored in the different file (block/cfq-cgroup.c).

We hope to discuss not which is better implementation, but what is the best way to implement I/O bandwidth control based on CFQ here.

Please give us your comments, questions and suggestions.

Finally, we introduce a usage of our implementation.

* Preparation for using 2 layer CFQ

 1. Adopt this patchset to kernel 2.6.25-rc2-mm1.

 2. Build kernel with CFQ-CGROUP option.

 3. Restart new kernel.

 4. Mount cfq_cgroup special device to device directory.
    ex.
      mkdir /dev/cgroup
      mount -t cgroup -o cfq cfq /dev/cgroup

* Usage of grouping control.
 - Create New group
     Make new directory under /dev/cgroup.
     For example, the following command genrerates a 'test1' group.
        mkdir /dev/cgroup/test1

 - Insert task to group
     Write process id(pid) on "tasks" entry in the corresponding group.
     For example, the following command sets task with pid 1100 into test1 group.
        echo 1100 > /dev/cgroup/test1/tasks
     Child tasks of this tasks is also inserted into test1 group.

 - Change I/O priority of group
     Write priority on "cfq.ioprio" entry in corresponding group.
     For example, the following command sets priority of rank 2 to 'test1' group.
        echo 2 > /dev/cgroup/test1/tasks
     I/O priority for cgroups takes the value from 0 to 7. It is same as
     existing per-task CFQ.

 - Change I/O priority of task
     Use existing "ionice" command.

* Example
 Two I/O load (dd command) runs some conditions.

 - When they are same group and same priority,

   program
    #!/bin/sh
    echo $$ > /dev/cgroup/tasks
    echo $$ > /dev/cgroup/test/tasks
    ionice -c 2 -n 3 dd if=/internal/data1 of=/dev/null bs=1M count=1K &
    ionice -c 2 -n 3 dd if=/internal/data2 of=/dev/null bs=1M count=1K &
    echo $$ > /dev/cgroup/test2/tasks
    echo $$ > /dev/cgroup/tasks

   result
    1024+0 records in
    1024+0 records out
    1073741824 bytes (1.1 GB) copied, 27.7676 s, 38.7 MB/s
    1024+0 records in
    1024+0 records out
    1073741824 bytes (1.1 GB) copied, 28.8482 s, 37.2 MB/s

   These tasks was fair, therefore they finished at similar time.

 - When they are same group and different priorities (0 and 7),

   program
    #!/bin/sh
    echo $$ > /dev/cgroup/tasks
    echo $$ > /dev/cgroup/test/tasks
    ionice -c 2 -n 0 dd if=/internal/data1 of=/dev/null bs=1M count=1K &
    ionice -c 2 -n 7 dd if=/internal/data2 of=/dev/null bs=1M count=1K &
    echo $$ > /dev/cgroup/test2/tasks
    echo $$ > /dev/cgroup/tasks

   result
    1024+0 records in
    1024+0 records out
    1073741824 bytes (1.1 GB) copied, 18.8373 s, 57.0 MB/s
    1024+0 records in
    1024+0 records out
    1073741824 bytes (1.1 GB) copied, 28.108 s, 38.2 MB/s

   The first task (copy data1) had high priority, therefore it finished at fast.

 - When they are different groups and different priorities (0 and 7),

program
```
#!/bin/sh
echo $$ > /dev/cgroup/tasks
echo $$ > /dev/cgroup/test/tasks
ionice -c 2 -n 0 dd if=/internal/data1 of=/dev/null bs=1M count=1K
echo $$ > /dev/cgroup/test2/tasks
ionice -c 2 -n 7 dd if=/internal/data2 of=/dev/null bs=1M count=1K
echo $$ > /dev/cgroup/tasks
```

result
```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 28.1661 s, 38.1 MB/s
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 28.8486 s, 37.2 MB/s
```

The first task (copy data1) had high priority, but they finished at similar time.
Because their groups had same priority.

- When they are different groups with different priorities (7 and 0)
  and same priority,

program
```
#!/bin/sh
echo $$ > /dev/cgroup/tasks
echo 7 > /dev/cgroup/test/cfq.ioprio
echo $$ > /dev/cgroup/test/tasks
ionice -c 2 -n 0 dd if=/internal/data1 of=/dev/null bs=1M count=1K >& test1.log &
echo 0 > /dev/cgroup/test2/cfq.ioprio
echo $$ > /dev/cgroup/test2/tasks
ionice -c 2 -n 7 dd if=/internal/data2 of=/dev/null bs=1M count=1K >& test2.log &
echo $$ > /dev/cgroup/tasks
```

result
```
=== test1.log ===
  1024+0 records in
  1024+0 records out
  1073741824 bytes (1.1 GB) copied, 27.3971 s, 39.2 MB/s
=== test2.log ===
  1024+0 records in
  1024+0 records out
  1073741824 bytes (1.1 GB) copied, 17.3837 s, 61.8 MB/s
```

This first task (copy data1) had high priority, but they finished at late.
Because its group had low priority.

=====
 Satoshi UHICDA
   NEC Corporation.