

---

Subject: [PATCH -mm 1/3] i/o bandwidth controller documentation

Posted by [Andrea Righi](#) on Sat, 12 Jul 2008 11:31:29 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Documentation of the block device I/O bandwidth controller: description, usage, advantages and design.

Signed-off-by: Andrea Righi <righi.andrea@gmail.com>

---

```
Documentation/controllers/io-throttle.txt | 282 ++++++
1 files changed, 282 insertions(+), 0 deletions(-)
create mode 100644 Documentation/controllers/io-throttle.txt
```

```
diff --git a/Documentation/controllers/io-throttle.txt b/Documentation/controllers/io-throttle.txt
```

```
new file mode 100644
```

```
index 0000000..ab33633
```

```
--- /dev/null
```

```
+++ b/Documentation/controllers/io-throttle.txt
```

```
@ @ -0,0 +1,282 @ @
```

```
+
```

```
+      Block device I/O bandwidth controller
```

```
+
```

```
+1. Description
```

```
+
```

```
+This controller allows to limit the I/O bandwidth of specific block devices for
+specific process containers (cgroups) imposing additional delays on I/O
+requests for those processes that exceed the limits defined in the control
+group filesystem.
```

```
+
```

```
+Bandwidth limiting rules offer better control over QoS with respect to priority
+or weight-based solutions that only give information about applications'
+relative performance requirements. Nevertheless, priority based solutions are
+affected by performance bursts, when only low-priority requests are submitted
+to a general purpose resource dispatcher.
```

```
+
```

```
+The goal of the I/O bandwidth controller is to improve performance
+predictability and provide performance isolation of different control groups
+sharing the same block devices.
```

```
+
```

```
+NOTE #1: If you're looking for a way to improve the overall throughput of the
+system probably you should use a different solution.
```

```
+
```

```
+NOTE #2: The current implementation does not guarantee minimum bandwidth
+levels, the QoS is implemented only slowing down I/O "traffic" that exceeds the
+limits specified by the user; minimum I/O rate thresholds are supposed to be
+guaranteed if the user configures a proper I/O bandwidth partitioning of the
+block devices shared among the different cgroups (theoretically if the sum of
+all the single limits defined for a block device doesn't exceed the total I/O
```

+bandwidth of that device).

+

## +2. User Interface

+

+A new I/O bandwidth limitation rule is described using the file

+blockio.bandwidth.

+

+The same file can be used to set multiple rules for different block devices

+relative to the same cgroup.

+

### +2.1. Configure I/O limiting rules

+

+The syntax to configure a limiting rule is the following:

+

```
+# /bin/echo DEV:BW:STRATEGY:BUCKET_SIZE > CGROUP/blockio.bandwidth
```

+

+ DEV is the name of the device the limiting rule is applied to.

+

+ BW is the maximum I/O bandwidth on DEVICE allowed by CGROUP; bandwidth must be expressed in bytes/s. A generic I/O bandwidth limiting rule for a block device DEV can be removed setting the BW value to 0.

+

+ STRATEGY is the throttling strategy used to throttle the applications' I/O requests from/to device DEV. At the moment two different strategies can be used:

+

+ 0 = leaky bucket: the controller accepts at most B bytes ( $B = BW * \text{time}$ ); further I/O requests are delayed scheduling a timeout for the tasks that made those requests.

+

+ Different I/O flow

```
+   |||
+   |v|
+   | v
+   v
```

```
+   .....
+   \  /
```

+ \ / leaky-bucket

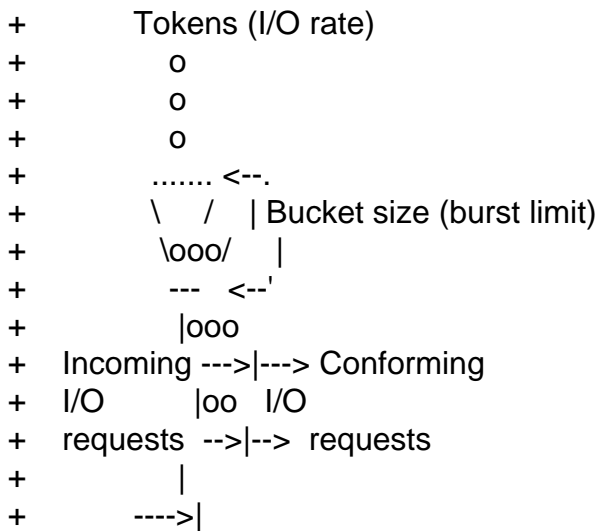
```
+   ---
+   |||
+   vvv
```

+ Smoothed I/O flow

+

+ 1 = token bucket: BW tokens are added to the bucket every seconds; the bucket can hold at the most BUCKET\_SIZE tokens; I/O requests are accepted if there are available tokens in the bucket; when a request of N bytes arrives N tokens are removed from the bucket; if fewer than N tokens are available the request is

+ delayed until a sufficient amount of token is available in  
+ the bucket.



+ Leaky bucket is more precise than token bucket to respect the bandwidth  
+ limits, because bursty workloads are always smoothed. Token bucket, instead,  
+ allows a small irregularity degree in the I/O flows (burst limit), and, for  
+ this, it is better in terms of efficiency (bursty workloads are not smoothed  
+ when there are sufficient tokens in the bucket).

+ BUCKET\_SIZE is used only with token bucket (STRATEGY == 1) and defines the  
+ size of the bucket in bytes.

+ CGROUP is the name of the limited process container.

+Also the following syntaxes are allowed:

+ remove an I/O bandwidth limiting rule  
+# /bin/echo DEV:0 > CGROUP/blockio.bandwidth

+ configure a limiting rule using leaky bucket throttling (ignore bucket size):  
+# /bin/echo DEV:BW:0 > CGROUP/blockio.bandwidth

## +2.2. Show I/O limiting rules

+All the defined rules and statistics for a specific cgroup can be shown reading  
+the file blockio.bandwidth. The following syntax is used:

+\$ cat CGROUP/blockio.bandwidth

+MAJOR MINOR BW STRATEGY LEAKY\_STAT BUCKET\_SIZE BUCKET\_FILL TIME\_DELTA

+ MAJOR is the major device number of DEV (defined above)

+ MINOR is the minor device number of DEV (defined above)

```

+
+- BW, STRATEGY and BUCKET_SIZE are the same parameters defined above
+
+- LEAKY_STAT is the amount of bytes currently allowed by the I/O bandwidth
+ controller (only used with leaky bucket strategy - STRATEGY == 0)
+
+- BUCKET_FILL represents the amount of tokens present in the bucket (only used
+ with token bucket strategy - STRATEGY == 1)
+
+- TIME_DELTA can be one of the following:
+ - the amount of jiffies elapsed from the last I/O request (token bucket)
+ - the amount of jiffies during which the bytes given by LEAKY_STAT have been
+ accumulated (leaky bucket)
+
+Multiple per-block device rules are reported in multiple rows
+(DEVi, i = 1 .. n):
+
+$ cat CGROUP/blockio.bandwidth
+MAJOR1 MINOR1 BW1 STRATEGY1 LEAKY_STAT1 BUCKET_SIZE1 BUCKET_FILL1
TIME_DELTA1
+MAJOR1 MINOR1 BW2 STRATEGY2 LEAKY_STAT2 BUCKET_SIZE2 BUCKET_FILL2
TIME_DELTA2
+...
+MAJORn MINORn BWn STRATEGYn LEAKY_STATn BUCKET_SIZEn BUCKET_FILLn
TIME_DELTAn
+
+2.3. Examples
+
+* Mount the cgroup filesystem (blockio subsystem):
+ # mkdir /mnt/cgroup
+ # mount -t cgroup -oblockio blockio /mnt/cgroup
+
+* Instantiate the new cgroup "foo":
+ # mkdir /mnt/cgroup/foo
+ --> the cgroup foo has been created
+
+* Add the current shell process to the cgroup "foo":
+ # /bin/echo $$ > /mnt/cgroup/foo/tasks
+ --> the current shell has been added to the cgroup "foo"
+
+* Give maximum 1MiB/s of I/O bandwidth on /dev/sda for the cgroup "foo", using
+ leaky bucket throttling strategy:
+ # /bin/echo /dev/sda:$((1024 * 1024)):0:0 > \
+ > /mnt/cgroup/foo/blockio.bandwidth
+ # sh
+ --> the subshell 'sh' is running in cgroup "foo" and it can use a maximum I/O
+ bandwidth of 1MiB/s on /dev/sda
+

```

```

+* Give maximum 8MiB/s of I/O bandwidth on /dev/sdb for the cgroup "foo", using
+ token bucket throttling strategy, bucket size = 8MB:
+ # /bin/echo /dev/sdb:$((8 * 1024 * 1024)):1:$((8 * 1024 * 1024)) > \
+ > /mnt/cgroup/foo/blockio.bandwidth
+ # sh
+ --> the subshell 'sh' is running in cgroup "foo" and it can use a maximum I/O
+ bandwidth of 1MiB/s on /dev/sda (controlled by leaky bucket throttling)
+ and 8MiB/s on /dev/sdb (controlled by token bucket throttling)
+
+* Run a benchmark doing I/O on /dev/sda and /dev/sdb; I/O limits and usage
+ defined for cgroup "foo" can be shown as following:
+ # cat /mnt/cgroup/foo/blockio.bandwidth
+ 8 16 8388608 1 0 8388608 -522560 48
+ 8 0 1048576 0 737280 0 0 216
+
+* Extend the maximum I/O bandwidth for the cgroup "foo" to 16MiB/s on /dev/sda:
+ # /bin/echo /dev/sda:$((16 * 1024 * 1024)):0:0 > \
+ > /mnt/cgroup/foo/blockio.bandwidth
+ # cat /mnt/cgroup/foo/blockio.bandwidth
+ 8 16 8388608 1 0 8388608 -84432 206436
+ 8 0 16777216 0 0 0 0 15212
+
+* Remove limiting rule on /dev/sdb for cgroup "foo":
+ # /bin/echo /dev/sdb:0:0:0 > /mnt/cgroup/foo/blockio.bandwidth
+ # cat /mnt/cgroup/foo/blockio.bandwidth
+ 8 0 16777216 0 0 0 0 110388
+
+3. Advantages of providing this feature
+
+* Allow I/O traffic shaping for block device shared among different cgroups
+* Improve I/O performance predictability on block devices shared between
+ different cgroups
+* Limiting rules do not depend of the particular I/O scheduler (anticipatory,
+ deadline, CFQ, noop) and/or the type of the underlying block devices
+* The bandwidth limitations are guaranteed both for synchronous and
+ asynchronous operations, even the I/O passing through the page cache or
+ buffers and not only direct I/O (see below for details)
+* It is possible to implement a simple user-space application to dynamically
+ adjust the I/O workload of different process containers at run-time,
+ according to the particular users' requirements and applications' performance
+ constraints
+* It is even possible to implement event-based performance throttling
+ mechanisms; for example the same user-space application could actively
+ throttle the I/O bandwidth to reduce power consumption when the battery of a
+ mobile device is running low (power throttling) or when the temperature of a
+ hardware component is too high (thermal throttling)
+
+4. Design

```

+

+The I/O throttling is performed imposing an explicit timeout, via `schedule_timeout_killable()` on the processes that exceed the I/O bandwidth dedicated to the cgroup they belong to. I/O accounting happens per cgroup.

+

+It just works as expected for read operations: the real I/O activity is reduced synchronously according to the defined limitations.

+

+Write operations, instead, are modeled depending of the dirty pages ratio (write throttling in memory), since the writes to the real block devices are processed asynchronously by different kernel threads (`pdflush`). However, the dirty pages ratio is directly proportional to the actual I/O that will be performed on the real block device. So, due to the asynchronous transfers through the page cache, the I/O throttling in memory can be considered a form of anticipatory throttling to the underlying block devices.

+

+Multiple re-writes in already dirtied page cache areas are not considered for accounting the I/O activity. This is valid for multiple re-reads of pages already present in the page cache as well.

+

+This means that a process that re-writes and/or re-reads multiple times the same blocks in a file (without re-creating it by `truncate()`, `ftruncate()`, `creat()`, etc.) is affected by the I/O limitations only for the actual I/O performed to (or from) the underlying block devices.

+

+Multiple rules for different block devices are stored in a linked list, using the `dev_t` number of each block device as key to uniquely identify each element of the list. RCU synchronization is used to protect the whole list structure, since the elements in the list are not supposed to change frequently (they change only when a new rule is defined or an old rule is removed or updated), while the reads in the list occur at each operation that generates I/O. This allows to provide zero overhead for cgroups that do not use any limitation.

+

+WARNING: per-block device limiting rules always refer to the `dev_t` device number. If a block device is unplugged (i.e. a USB device) the limiting rules defined for that device persist and they are still valid if a new device is plugged in the system and it uses the same major and minor numbers.

+

+NOTE: explicit sleeps are *not* imposed on tasks doing asynchronous I/O (AIO) operations; AIO throttling is performed returning `-EAGAIN` from `sys_io_submit()`. Userspace applications must be able to handle this error code opportunely.

+

+5. Todo

+

+\* Try to reduce the cost of calling `cgroup_io_throttle()` on every `submit_bio(READ, ...)`; this is not too much expensive, but the call of `task_subsys_state()` has surely a cost. A possible solution could be to temporarily account I/O in the current `task_struct` and call

+ cgroup\_io\_throttle() only on each X MB of I/O. Or on each Y number of I/O  
+ requests as well. Better if both X and/or Y can be tuned at runtime by a  
+ userspace tool.  
+  
+\* Think an alternative design for general purpose usage; special purpose usage  
+ right now is restricted to improve I/O performance predictability and  
+ evaluate more precise response timings for applications doing I/O. To a large  
+ degree the block I/O bandwidth controller should implement a more complex  
+ logic to better evaluate real I/O operations cost, depending also on the  
+ particular block device profile (i.e. USB stick, optical drive, hard disk,  
+ etc.). This would also allow to appropriately account I/O cost for seeky  
+ workloads, respect to large stream workloads. Instead of looking at the  
+ request stream and try to predict how expensive the I/O cost will be, a  
+ totally different approach could be to collect request timings (start time /  
+ elapsed time) and based on collected informations, try to estimate the I/O  
+ cost and usage (idea proposed by Andrew Morton <akpm@linux-foundation.org>).  
--  
1.5.4.3

---

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

---