blkio-controller.txt

Block IO Controller

Overview

cgroup subsys "blkio" implements the block io controller. There seems to be a need of various kinds of IO control policies (like proportional BW, max BW) both at leaf nodes as well as at intermediate nodes in a storage hierarchy. Plan is to use the same cgroup based management interface for blkio controller and based on user options switch IO policies in the background.

In the first phase, this patchset implements proportional weight time based division of disk policy. It is implemented in CFQ. Hence this policy takes effect only on leaf nodes when CFQ is being used.

HOWTO

You can do a very simple testing of running two dd threads in two different cgroups. Here is what you can do.

- Enable Block IO controller
  CONFIG_BLK_CGROUP=y

- Enable group scheduling in CFQ
  CONFIG_CFQ_GROUP_IOSCHED=y

- Compile and boot into kernel and mount IO controller (blkio).

  mount -t cgroup -o blkio none /cgroup

- Create two cgroups
  mkdir -p /cgroup/test1/ /cgroup/test2

- Set weights of group test1 and test2
  echo 1000 > /cgroup/test1/blkio.weight
  echo 500 > /cgroup/test2/blkio.weight

- Create two same size files (say 512MB each) on same disk (file1, file2) and launch two dd threads in different cgroup to read those files.

  sync
  echo 3 > /proc/sys/vm/drop_caches

  dd if=/mnt/sdb/zerofile1 of=/dev/null &
  echo $! > /cgroup/test1/tasks
  cat /cgroup/test1/tasks

  dd if=/mnt/sdb/zerofile2 of=/dev/null &
  echo $! > /cgroup/test2/tasks
  cat /cgroup/test2/tasks

- At macro level, first dd should finish first. To get more precise data, keep on looking at (with the help of script), at blkio.disk_time and blkio.disk_sectors files of both test1 and test2 groups. This will tell how much disk time (in milli seconds), each group got and how many secotors each group dispatched to the disk. We provide fairness in terms of disk time, so ideally io.disk_time of cgroups should be in proportion to the weight.

Various user visible config options
==========
CONFIG_BLK_CGROUP
	- Block IO controller.

CONFIG_DEBUG_BLK_CGROUP
	- Debug help. Right now some additional stats file show up in cgroup
	 if this option is enabled.

CONFIG_CFQ_GROUP_IOSCHED
	- Enables group scheduling in CFQ. Currently only 1 level of group
	 creation is allowed.

Details of cgroup files
==========
- blkio.weight
	- Specifies per cgroup weight. This is default weight of the group
	 on all the devices until and unless overridden by per device rule.
	 (See blkio.weight_device).
	 Currently allowed range of weights is from 100 to 1000.

- blkio.weight_device
	- One can specify per cgroup per device rules using this interface.
	 These rules override the default value of group weight as specified
	 by blkio.weight.

	 Following is the format.

	 #echo dev_maj:dev_minor weight > /path/to/cgroup/blkio.weight_device
	 Configure weight=300 on /dev/sdb (8:16) in this cgroup
	 # echo 8:16 300 > blkio.weight_device
	 # cat blkio.weight_device
	 dev weight
	 8:16 300

	 Configure weight=500 on /dev/sda (8:0) in this cgroup
	 # echo 8:0 500 > blkio.weight_device
	 # cat blkio.weight_device
	 dev weight
	 8:0 500
	 8:16 300

	 Remove specific weight for /dev/sda in this cgroup
	 # echo 8:0 0 > blkio.weight_device
	 # cat blkio.weight_device
	 dev weight
	 8:16 300

- blkio.time
	- disk time allocated to cgroup per device in milliseconds. First
	 two fields specify the major and minor number of the device and
	 third field specifies the disk time allocated to group in
	 milliseconds.

- blkio.sectors

	- number of sectors transferred to/from disk by the group. First
	 two fields specify the major and minor number of the device and
	 third field specifies the number of sectors transferred by the
	 group to/from the device.

- blkio.io_service_bytes
	- Number of bytes transferred to/from the disk by the group. These
	 are further divided by the type of operation - read or write, sync
	 or async. First two fields specify the major and minor number of the
	 device, third field specifies the operation type and the fourth field
	 specifies the number of bytes.

- blkio.io_serviced
	- Number of IOs completed to/from the disk by the group. These
	 are further divided by the type of operation - read or write, sync
	 or async. First two fields specify the major and minor number of the
	 device, third field specifies the operation type and the fourth field
	 specifies the number of IOs.

- blkio.io_service_time
	- Total amount of time between request dispatch and request completion
	 for the IOs done by this cgroup. This is in nanoseconds to make it
	 meaningful for flash devices too. For devices with queue depth of 1,
	 this time represents the actual service time. When queue_depth > 1,
	 that is no longer true as requests may be served out of order. This
	 may cause the service time for a given IO to include the service time
	 of multiple IOs when served out of order which may result in total
	 io_service_time > actual time elapsed. This time is further divided by
	 the type of operation - read or write, sync or async. First two fields
	 specify the major and minor number of the device, third field
	 specifies the operation type and the fourth field specifies the
	 io_service_time in ns.

- blkio.io_wait_time
	- Total amount of time the IOs for this cgroup spent waiting in the
	 scheduler queues for service. This can be greater than the total time
	 elapsed since it is cumulative io_wait_time for all IOs. It is not a
	 measure of total time the cgroup spent waiting but rather a measure of
	 the wait_time for its individual IOs. For devices with queue_depth > 1
	 this metric does not include the time spent waiting for service once
	 the IO is dispatched to the device but till it actually gets serviced
	 (there might be a time lag here due to re-ordering of requests by the
	 device). This is in nanoseconds to make it meaningful for flash
	 devices too. This time is further divided by the type of operation -
	 read or write, sync or async. First two fields specify the major and
	 minor number of the device, third field specifies the operation type
	 and the fourth field specifies the io_wait_time in ns.

- blkio.io_merged
	- Total number of bios/requests merged into requests belonging to this
	 cgroup. This is further divided by the type of operation - read or
	 write, sync or async.

- blkio.io_queued
	- Total number of requests queued up at any given instant for this
	 cgroup. This is further divided by the type of operation - read or

write, sync or async.

- blkio.avg_queue_size
	- Debugging aid only enabled if CONFIG_DEBUG_BLK_CGROUP=y.
	 The average queue size for this cgroup over the entire time of this
	 cgroup's existence. Queue size samples are taken each time one of the
	 queues of this cgroup gets a timeslice.

- blkio.group_wait_time
	- Debugging aid only enabled if CONFIG_DEBUG_BLK_CGROUP=y.
	 This is the amount of time the cgroup had to wait since it became busy
	 (i.e., went from 0 to 1 request queued) to get a timeslice for one of
	 its queues. This is different from the io_wait_time which is the
	 cumulative total of the amount of time spent by each IO in that cgroup
	 waiting in the scheduler queue. This is in nanoseconds. If this is
	 read when the cgroup is in a waiting (for timeslice) state, the stat
	 will only report the group_wait_time accumulated till the last time it
	 got a timeslice and will not include the current delta.

- blkio.empty_time
	- Debugging aid only enabled if CONFIG_DEBUG_BLK_CGROUP=y.
	 This is the amount of time a cgroup spends without any pending
	 requests when not being served, i.e., it does not include any time
	 spent idling for one of the queues of the cgroup. This is in
	 nanoseconds. If this is read when the cgroup is in an empty state,
	 the stat will only report the empty_time accumulated till the last
	 time it had a pending request and will not include the current delta.

- blkio.idle_time
	- Debugging aid only enabled if CONFIG_DEBUG_BLK_CGROUP=y.
	 This is the amount of time spent by the IO scheduler idling for a
	 given cgroup in anticipation of a better request than the exising ones
	 from other queues/cgroups. This is in nanoseconds. If this is read
	 when the cgroup is in an idling state, the stat will only report the
	 idle_time accumulated till the last idle period and will not include
	 the current delta.

- blkio.dequeue
	- Debugging aid only enabled if CONFIG_DEBUG_BLK_CGROUP=y. This
	 gives the statistics about how many a times a group was dequeued
	 from service tree of the device. First two fields specify the major
	 and minor number of the device and third field specifies the number
	 of times a group was dequeued from a particular device.

- blkio.reset_stats
	- Writing an int to this file will result in resetting all the stats
	 for that cgroup.

CFQ sysfs tunable
=================
/sys/block/<disk>/queue/iosched/group_isolation

If group_isolation=1, it provides stronger isolation between groups at the
expense of throughput. By default group_isolation is 0. In general that
means that if group_isolation=0, expect fairness for sequential workload
only. Set group_isolation=1 to see fairness for random IO workload also.

Generally CFQ will put random seeky workload in sync-noidle category. CFQ will disable idling on these queues and it does a collective idling on group of such queues. Generally these are slow moving queues and if there is a sync-noidle service tree in each group, that group gets exclusive access to disk for certain period. That means it will bring the throughput down if group does not have enough IO to drive deeper queue depths and utilize disk capacity to the fullest in the slice allocated to it. But the flip side is that even a random reader should get better latencies and overall throughput if there are lots of sequential readers/sync-idle workload running in the system.

If group_isolation=0, then CFQ automatically moves all the random seeky queues in the root group. That means there will be no service differentiation for that kind of workload. This leads to better throughput as we do collective idling on root sync-noidle tree.

By default one should run with group_isolation=0. If that is not sufficient and one wants stronger isolation between groups, then set group_isolation=1 but this will come at cost of reduced throughput.

What works
==========
- Currently only sync IO queues are support. All the buffered writes are still system wide and not per group. Hence we will not see service differentiation between buffered writes between groups.