Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Andrea Righi on Tue, 22 Feb 2011 22:41:41 GMT
View Forum Message <> Reply to Message

On Tue, Feb 22, 2011 at 02:34:03PM -0500, Vivek Goyal wrote:
> On Tue, Feb 22, 2011 at 06:12:51PM +0100, Andrea Righi wrote:
> > Currently the blkio.throttle controller only support synchronous IO requests.
> > This means that we always look at the current task to identify the "owner" of
> > each IO request.
> >
> > However dirty pages in the page cache can be wrote to disk asynchronously by
> > the per-bdi flusher kernel threads or by any other thread in the system,
> > according to the writeback policy.
> >
> > For this reason the real writes to the underlying block devices may
> > occur in a different IO context respect to the task that originally
> > generated the dirty pages involved in the IO operation. This makes the
> > tracking and throttling of writeback IO more complicate respect to the
> > synchronous IO from the blkio controller's perspective.
> >
> > The same concept is also valid for anonymous pages involed in IO operations
> > (swap).
> >
> > This patch allow to track the cgroup that originally dirtied each page in page
> > cache and each anonymous page and pass these informations to the blk-throttle
> > controller. These informations can be used to provide a better service level
> > differentiation of buffered writes swap IO between different cgroups.
> >
>
> Hi Andrea,
>
> Thanks for the patches. Before I look deeper into patches, had few
> general queries/thoughts.
>
> - So this requires memory controller to be enabled. Does it also require
>   these to be co-mounted?

No and no. The blkio controller enables and uses the page_cgroup
functionality, but it doesn't depend on the memory controller. It
automatically selects CONFIG_MM_OWNER and CONFIG_PAGE_TRACKING (last
one added in PATCH 3/5) and this is sufficient to make page_cgroup
usable from any generic controller.

>
> - Currently in throttling there is no limit on number of bios queued
>   per group. I think this is not necessarily a very good idea because
>   if throttling limits are low, we will build very long bio queues. So
>   some AIO process can queue up lots of bios, consume lots of memory

> without getting blocked. I am sure there will be other side affects
> too. One of the side affects I noticed is that if an AIO process
> queues up too much of IO, and if I want to kill it now, it just hangs
> there for a really-2 long time (waiting for all the throttled IO
> to complete).
>
> So I was thinking of implementing either per group limit or per io
> context limit and after that process will be put to sleep. (something
> like request descriptor mechanism).

io context limit seems a better solution for now. We can also expect some help from the memory controller, if we'll have the dirty memory limit per cgroup in the future the max amount of bios queued will be automatically limited by this functionality.

>
> If that's the case, then comes the question of what do to about kernel
> threads. Should they be blocked or not. If these are blocked then a
> fast group will also be indirectly throttled behind a slow group. If
> they are not then we still have the problem of too many bios queued
> in throttling layer.

I think kernel threads should be never forced to sleep, to avoid the classic "priority inversion" problem and create potential DoS in the system.

Also for this part the dirty memory limit per cgroup could help a lot, because a cgroup will never exceed its "quota" of dirty memory, so it will not be able to submit more than a certain amount of bios (corresponding to the dirty memory limit).

>
> - What to do about other kernel thread like kjournald which is doing
> IO on behalf of all the filesystem users. If data is also journalled
> then I think again everything got serialized and a faster group got
> backlogged behind a slower one.

This is the most critical issue IMHO.

The blkio controller should need some help from the filesystems to understand which IO request can be throttled and which cannot. At the moment critical IO requests (with critical I mean that are dependency for other requests) and non-critical requests are mixed together in a way that throttling a single request may stop a lot of other requests in the system, and at the block layer it's not possible to retrieve such informations.

I don't have a solution for this right now. Except looking at each

filesystem implementation and try to understand how to pass these informations to the block layer.

>
> - Two processes doing IO to same file and slower group will throttle
> IO for faster group also. (flushing is per inode).
>

I think we should accept to have an inode granularity. We could redesign the writeback code to work per-cgroup / per-page, etc. but that would add a huge overhead. The limit of inode granularity could be an acceptable tradeoff, cgroups are supposed to work to different files usually, well.. except when databases come into play (ouch!).

Thanks,
-Andrea

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs