Subject: Re: [PATCH 3/3] i/o accounting and control
Posted by akpm on Thu, 26 Jun 2008 23:15:50 GMT
View Forum Message <> Reply to Message

On Fri, 27 Jun 2008 00:37:25 +0200
Andrea Righi <righi.andrea@gmail.com> wrote:

> > All this could be made cheaper if we were to reduce the sampling rate:
> > only call cgroup_io_throttle() on each megabyte of IO (for example).
> >
> > current->amount_of_io += bio->bi_size;
> > if (current->amount_of_io > 1024*1024) {
> >   cgroup_io_throttle(bio->bi_bdev, bio->bi_size);
> >   current->amount_of_io -= 1024 * 1024;
> > }
>
> What about ratelimiting the sampling based on i/o requests?
>
>   current->io_requests++;
>   if (current->io_requests > CGROUP_IOTHROTTLE_RATELIMIT) {
>   cgroup_io_throttle(bio->bi_bdev, bio->bi_size);
>   current->io_requests = 0;
>   }
>
> The throttle would fail for large bio->bi_size requests, but it would
> work also with multiple devices. And probably this would penalize tasks
> having a seeky i/o workload (many requests means more checks for
> throttling).

Yup. To a large degree, a 4k IO has a similar cost to a 1MB IO.
Certainly the 1MB IO is not 256 times as expensive!

Some sort of averaging fudge factor could be used here. For example, a
1MB IO might be considered, umm 3.4 times as expensive as a 4k IO. But
it varies a lot depending upon the underlying device. For a USB stick,
sure, we're close to 256x. For a slow-seek, high-bandwidth device
(optical?) it's getting closer to 1x. No single fudge-factor will suit
all devices, hence userspace-controlled tunability would be needed here
to avoid orders-of-magnitude inaccuracies.

The above

 cost ~= per-device-fudge-factor * io-size

can of course go horridly wrong because it doesn't account for seeks at
all. Some heuristic which incorporates per-cgroup seekiness (in some
weighted fashion) will help there.

I dunno. It's not a trivial problem, and I suspect that we'll need to
get very fancy in doing this if we are to avoid an implementation which
goes unusably badly wrong in some situations.

I wonder if we'd be better off with a completely different design.

<thinks for five seconds>

At present we're proposing that we look at the request stream and
a-priori predict how expensive it will be. That's hard. What if we
were to look at the requests post-facto and work out how expensive they
were? Say, for each request which this cgroup issued, look at the
start-time and the completion-time. Take the difference (elapsed time)
and divide that by wall time. Then we end up with a simple percentage:
"this cgroup is using the disk 10% of the time".

That's fine, as long as nobody else is using the device! If multiple
cgroups are using the device then we need to do, err, something.

Or do we? Maybe not - things will, on average, sort themselves out,
because everyone will slow everyone else down. It'll have inaccuracies
and failure scenarios and the smarty-pants IO schedulers will get in
the way.

Interesting project, but I do suspect that we'll need a lot of
complexity in there (and huge amounts of testing) to get something
which is sufficiently accurate to be generally useful.