Subject: Re: [PATCH 0/5] blk-throttle: writeback and swap IO control
Posted by Vivek Goyal on Wed, 23 Feb 2011 15:23:54 GMT
View Forum Message <> Reply to Message

> > Agreed. Granularity of per inode level might be accetable in many
> > cases. Again, I am worried faster group getting stuck behind slower
> > group.
> >
> > I am wondering if we are trying to solve the problem of ASYNC write throttling
> > at wrong layer. Should ASYNC IO be throttled before we allow task to write to
> > page cache. The way we throttle the process based on dirty ratio, can we
> > just check for throttle limits also there or something like that.(I think
> > that's what you had done in your initial throttling controller implementation?)
>
> Right. This is exactly the same approach I've used in my old throttling
> controller: throttle sync READs and WRITEs at the block layer and async
> WRITEs when the task is dirtying memory pages.
>
> This is probably the simplest way to resolve the problem of faster group
> getting blocked by slower group, but the controller will be a little bit
> more leaky, because the writeback IO will be never throttled and we'll
> see some limited IO spikes during the writeback.

Yes writeback will not be throttled. Not sure how big a problem that is.

- We have controlled the input rate. So that should help a bit.
- May be one can put some high limit on root cgroup to in blkio throttle controller to limit overall WRITE rate of the system.
- For SATA disks, try to use CFQ which can try to minimize the impact of WRITE.

It will atleast provide consistent bandwindth experience to application.

>However, this is always
> a better solution IMHO respect to the current implementation that is
> affected by that kind of priority inversion problem.
>
> I can try to add this logic to the current blk-throttle controller if
> you think it is worth to test it.

At this point of time I have few concerns with this approach.

- Configuration issues. Asking user to plan for SYNC ans ASYNC IO separately is inconvenient. One has to know the nature of workload.

- Most likely we will come up with global limits (atleast to begin with), and not per device limit. That can lead to contention on one single lock and scalability issues on big systems.

Having said that, this approach should reduce the kernel complexity a lot. So if we can do some intelligent locking to limit the overhead then it will boil down to reduced complexity in kernel vs ease of use to user. I guess at this point of time I am inclined towards keeping it simple in kernel.

Couple of people have asked me that we have backup jobs running at night and we want to reduce the IO bandwidth of these jobs to limit the impact on latency of other jobs, I guess this approach will definitely solve that issue.

IMHO, it might be worth trying this approach and see how well does it work. It might not solve all the problems but can be helpful in many situations.

I feel that for proportional bandwidth division, implementing ASYNC control at CFQ will make sense because even if things get serialized in higher layers, consequences are not very bad as it is work conserving algorithm. But for throttling serialization will lead to bad consequences.

May be one can think of new files in blkio controller to limit async IO per group during page dirty time.

blkio.throttle.async.write_bps_limit
blkio.throttle.async.write_iops_limit

Thanks
Vivek

_____

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