facebook
IO controllers and cgroupv2, or why I wake up crying in the middle of the night.

Josef Bacik
Software Engineer
Why do we need a io controller?
Background

- CFQ’s io.weight
  - Required using CFQ for disk scheduling (bad).
- io.max
  - Not work conserving (not bad if that’s what you want).
  - Requires queue lock (bad).
The io stack (multi-queue)
io.latency

- Applies per peer group.
- Winner takes all, no proportionality at all.
- Meant to be work conserving.
  - As long as threshold requirements are met, anybody can do whatever they want.
- Winner takes all, no proportional throttling.
Challenges and solutions

• No feedback system when under pressure.
• Cgroup wide io pressure indicator.
• System doesn’t expect latency at submit_bio() time.
  • Delayed throttling at user space return time.
• Ext4 by design has a priority inversion.
  • Use btrfs (or xfs in theory, but we haven’t tried that.)
Memory leak on spinning rust
Memory leak on SSD

Graph showing performance metrics over time, with labels indicating specific events and performance characteristics.
IO storm with tar on spinning rust
Fbtax2 slices

- workload.slice
- Workload
- Container job daemon.
- system.slice
- chef
- hostcritical.slice
- ssh
Fbtax2 components

- memory.low
  - Minimum amount the job needs to run safely.
- io.latency
  - Protect the workload at all costs.
- Oomd
  - Kill anything that exceeds the pressure thresholds.
io.weight

- Currently in testing
- Provides good proportional balancing after a given amount of time.
- Worst case scenario you lose 10% of your overall capacity with saturation level IO.
- Work conserving (as long as we’re below saturation point, everybody is free to go as they will.)
Learn more

https://facebookmicrosites.github.io/cgroup2/
Questions?