Where did my storage's speed go?

Paolo Valente <paolo.valente@linaro.org>

What we will see

- Services as WEB hosting, video/audio streaming, cloud storage, containers, virtual machines, entertainment systems, ...
  - need to move data to/from storage
  - need to guarantee at least a minimum I/O bandwidth and a bounded latency to each client
    - For brevity we'll call client any entity that needs guarantees (so also a container, VM and so on)
- Without I/O control, these guarantees can be provided only at the price of throwing away **90%** of the storage speed
- But I/O control itself, with throttling, throws away **80%**
- The BFQ I/O scheduler solves this problem with all workloads, except for ... those made of mostly random I/O, where it loses **80%** too
- An improvement of BFQ now enables **~100%** of the speed to be always reached

Professional-quality guarantees

- Unexpected news for you: all of you here have (at least) one Internet connection!
- Your contract
  - Guarantees a minimum bandwidth (if professional service)
  - Limits maximum bandwidth
  - Provides you with an average bandwidth
- Average bandwidth likely to be at least half or one third of maximum
  - In any case, much higher than minimum guaranteed
- Would you be happy if average were about equal to minimum?
- Why is minimum much lower than average?
- Or, how can average be virtually always that higher than minimum?

The overbooking trick

- For the vast majority of services,
  - the average number of active clients is much lower than the total number of subscribers
  - the sum of the minimum bandwidths for the average active clients is much lower than the total bandwidth that the system can deliver
  - even the total average traffic generated is much lower than the total bandwidth available
- So, instead of wasting resources, providers
  - Size resources for the average total demand, and not for the maximum possible demand
    - In particular, resources are sized so as every client gets a very high average bandwidth
  - In other words, the bandwidth not used by inactive and low-traffic clients is constantly re-distributed among active clients
- Thus
  - Clients are happy, as they enjoy a high bandwidth most of the time
  - Providers are happy because they make clients happy without wasting resources

What about full load?

- If the total bandwidth demand occasionally happens to be higher than average
  - Minimum-bandwidth guarantees come into play

I/O-intensive services

- This same scheme holds for any service that involves moving data to/from storage
  - WEB hosting, video/audio streaming, cloud storage, containers, virtual machines, entertainment systems, ...
- Linux has I/O-control mechanisms to implement the above fundamental service scheme
  - Classical solution: the throttling I/O policy
- Bonus: the new blk-mq I/O stack fully exploits the speed of even the fastest storage available
  - Storage units can then deliver very high bandwidths to clients

We have a problem ...

- Throttling makes storage speed ... disappear
- Consider only throttling in this demo: https://youtu.be/lqpgghsuYUk
- This demo shows the results of one of the tests reported in this LWN article: https://lwn.net/Articles/763603/

So, how does the world go on?

- If throttling easily kills throughput
- How can companies provide the previous standard service scheme with Linux?
- They resort to *survival techniques* 🙂
  - Lack of strong I/O-control capabilities, combined with an impressive number of possible I/O scenarios, lead to an intricate jungle of combination of bandwidth, latency and throughput issues
  - Since it is hard to see far, companies solve, with *ad hoc* remedies, the problems they bump into, one at a time

Examples and drawbacks

- Examples
  - Throttle clients that happen to monopolize I/O
  - Use proportional-share with the CFQ I/O scheduler and increase the weights of suffering clients
- But our results clearly show that throttling
  - Is exactly the solution that easily kills throughput
  - May even lose control as workload is not homogenous
- As for CFQ, it fails to control I/O and get a high throughput with flash and queueing storage

Dedicated storage

- Given this nasty situation, companies often fallback to dedicated storage when they need reliable guarantees
- The I/O of clients that need bandwidth and latency guarantees is dispatched to *performant drives*, not shared with any other I/O
- How utilized are these drives?

Wild overprovisioning

- With some simple tests, it is easy to see that
  - Unless workload on dedicated storage is perfectly symmetric and homogeneous among clients
  - It is possible to guarantee bandwidth and latency to the unluckiest clients only if less than 10% of the speed of the storage is used !!!!
- So, this has basically nothing, or very little to do with redundancy

Intermediate wrap up

- In the face of
  - Faster and faster storage
  - Faster and faster I/O stack
- Throttling may waste 80% of the throughput!
- Dedicated storage may waste 90% of the throughput!
- In the end, one has to buy and manage five to ten times more storage resources than those sufficient to provide the total bandwidth needed
- This situation is particularly problematic with lower-end hardware
  - Some service levels may be practically impossible to reach!
- Fortunately, now there seems to be a new solution ...

Proportional share on BFQ

- Each group is assigned a weight, and receives a portion of the total throughput proportional to its weight
- This scheme guarantees minimum bandwidths in the same way that low limits do in throttling
  - It guarantees to each group a minimum bandwidth equal to the ratio between the weight of the group, and the sum of the weights of all the groups that may be active at the same time
- As shown in the demo, 100% of the maximum possible throughput reached
- But there is still a very nasty workload for BFQ ...

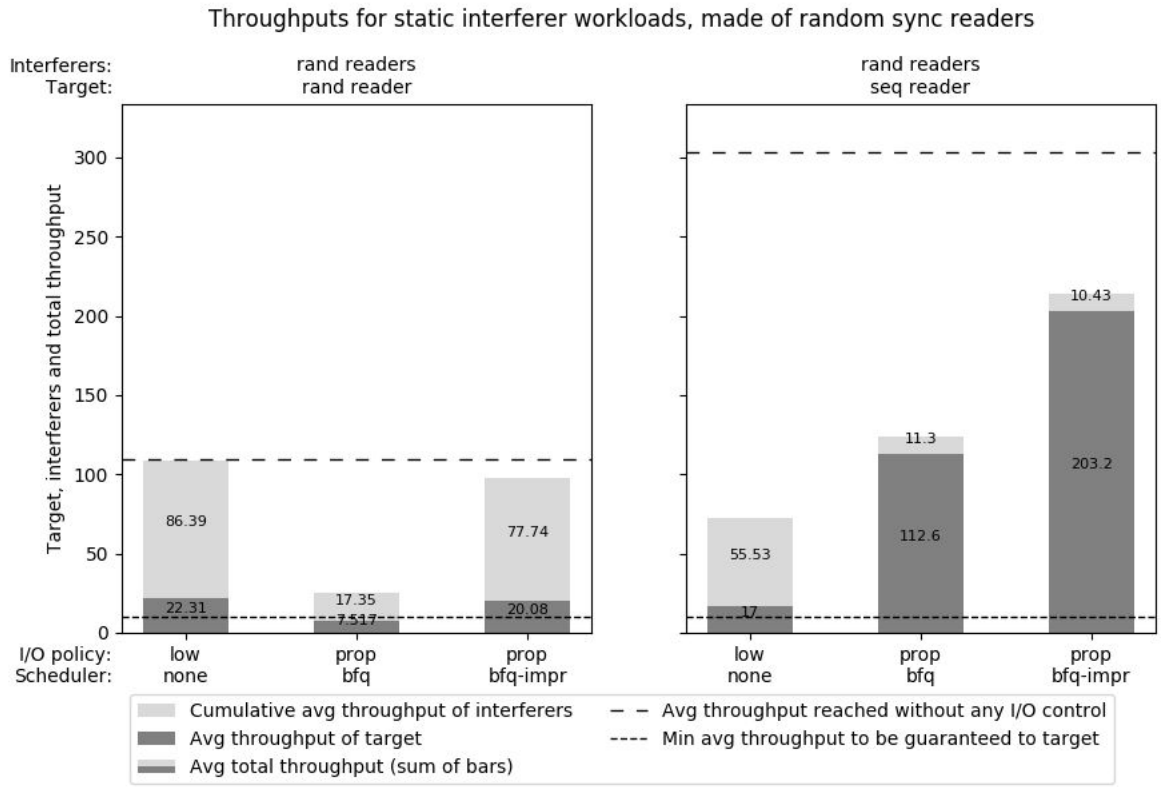
BFQ Achilles' heel

- *SSDs can handle many IOPS and tend to perform best with simple algorithm like noop or deadline while BFQ is well adapted to HDDs*

  Improving performance - ArchLinux

  - It still refers to noop and not none, but the gist is the same
- Why does this idea persists?
- Because BFQ sucks with random I/O
- Yet a new improvement is now under public testing
- Results in next slide
  - The test is the same as in the demo, but now all five groups do random I/O
  - The target is just one of the five groups
  - Consider only the left subplot

BFQ improvement



Conclusion

- After the last BFQ improvement, we have the following solution to implement professional-quality service guarantees
  - Minimum bandwidth -> guaranteed by BFQ
  - High average bandwidth -> reached because BFQ
    - Always keeps throughput close to 100% of the available speed
    - Systematically distributes throughput among the only active clients according to their weights
  - Maximum bandwidth -> very easy to limit with throttling
    - Natural use of throttling
    - No indirect throughput loss caused!
- This solution is general, standard, clean and reliable way
- Of course no silver bullet!
  - But now the ground in incomparably more solid than with a jungle of ad hoc solutions