How fast is fast storage under Linux?

Paolo Valente

University of Modena and Reggio Emilia - Italy

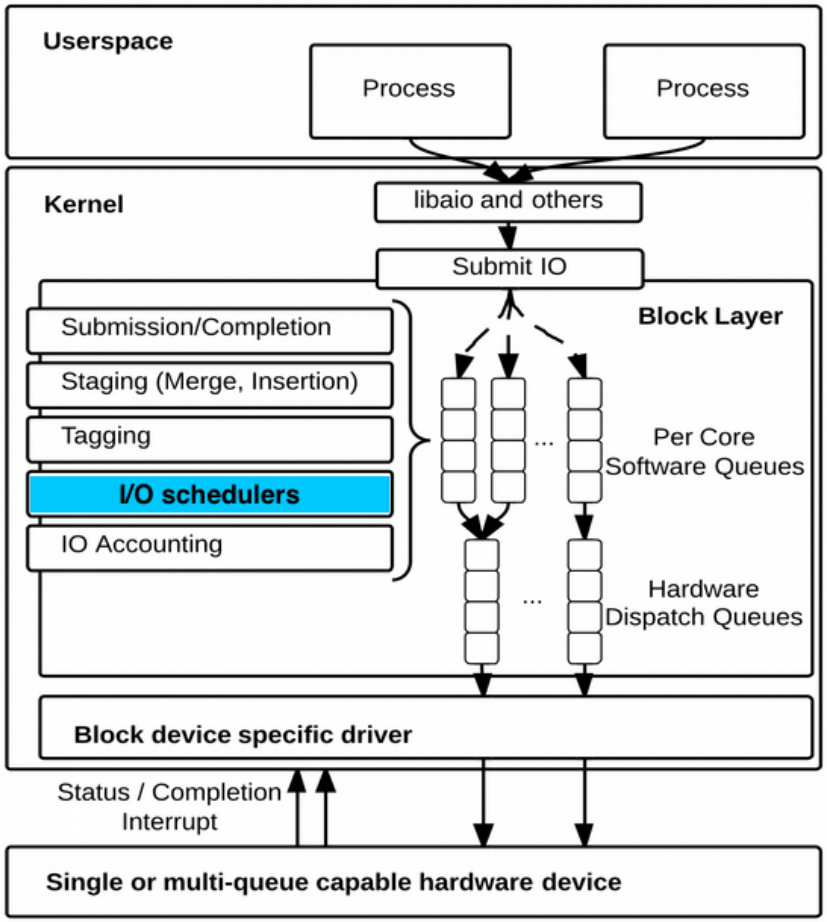
November 3, 2018

Roadmap

- Driving drives at full speed with *blk-mq*
- But crashing with real server applications
- Or losing 80% of the speed if trying to control I/O
- BFQ: a solution to get back 100% of the speed
  - And full control
- What about personal systems?
  - Without BFQ, very low responsiveness
    - Even with fastest storage
  - BFQ comes to the rescue: high responsiveness recovered

blk-mq

- Highest-possible parallelism for maximum throughput
- We'll talk about I/O schedulers in a few slides



Test drive

- PLEXTOR PX-256M5S SSD
  - 520 MB/s sequential read
  - 73 KIOPS random read

- Mounted on an (old) ThinkPad W520
  - 2.4GHz Intel Core i7-2760QM as CPU
  - 1.3 GHz DDR3 DRAM

Running fast even with *sync* random I/O ...

- Even with buffered *sync* 4KB random reads
  - I.e., every process issuing a single new read request only after the previous one is completed
- Throughput: 200 MB/s (50 KIOPS)

Great, let's use this speed for something useful

- Servers
  - Guarantee I/O bandwidth to
    - a set of clients (of any kind of service)
    - containers
    - virtual machines
    - ...
  - For brevity, we will consider the case of a set of clients
- Personal systems
  - Start applications quickly
  - Playback audios/videos

Serving clients at maximum total speed!

- With, e.g.,
  - four clients doing sequential reads
  - one client doing random reads
- Total throughout: 513 MB/s

What about control?

- Let's check it with the first part of this demo: `https://youtu.be/V6pXtwvfh4U`

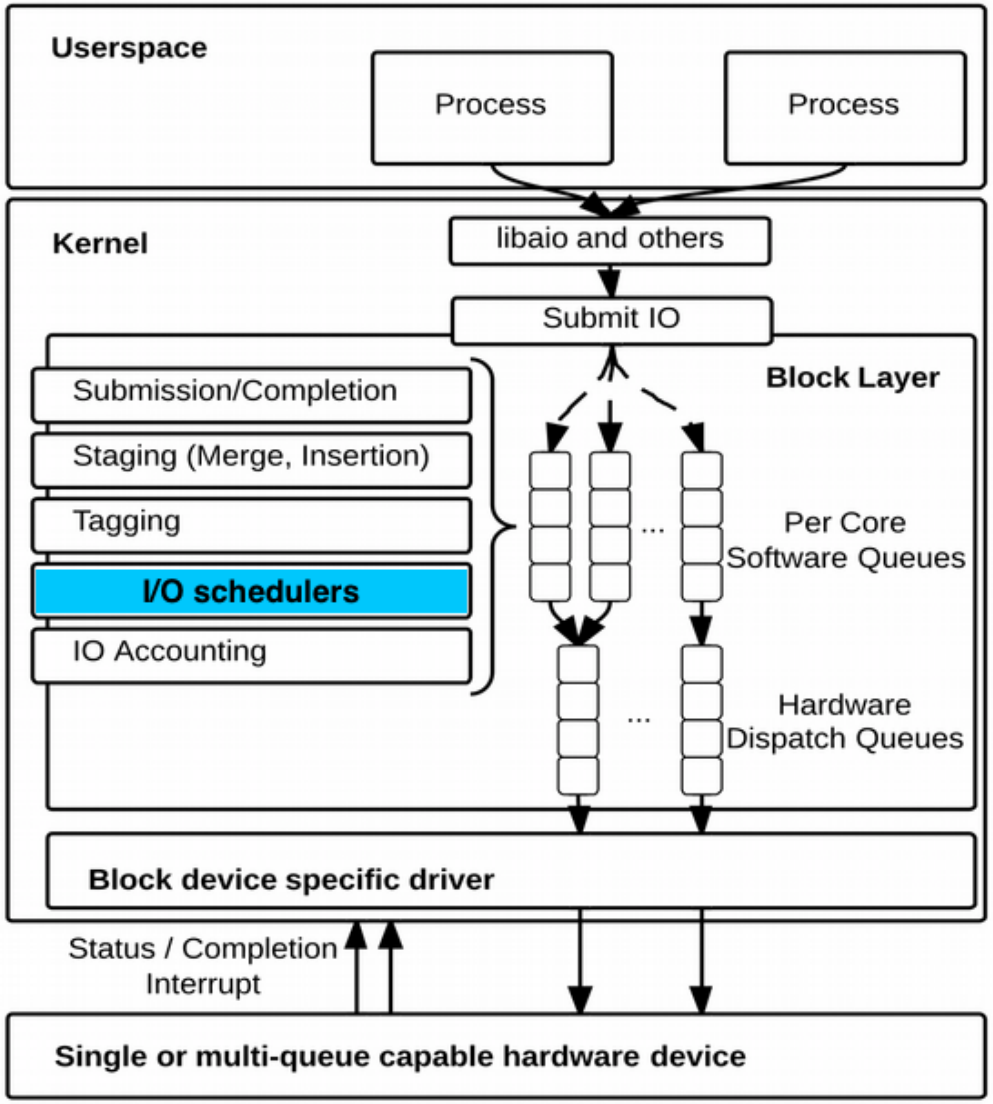
... ooops

"Rowan Atkinson crashes McLaren F1 supercar into tree"
BBC News, 5 August 2011

Why?

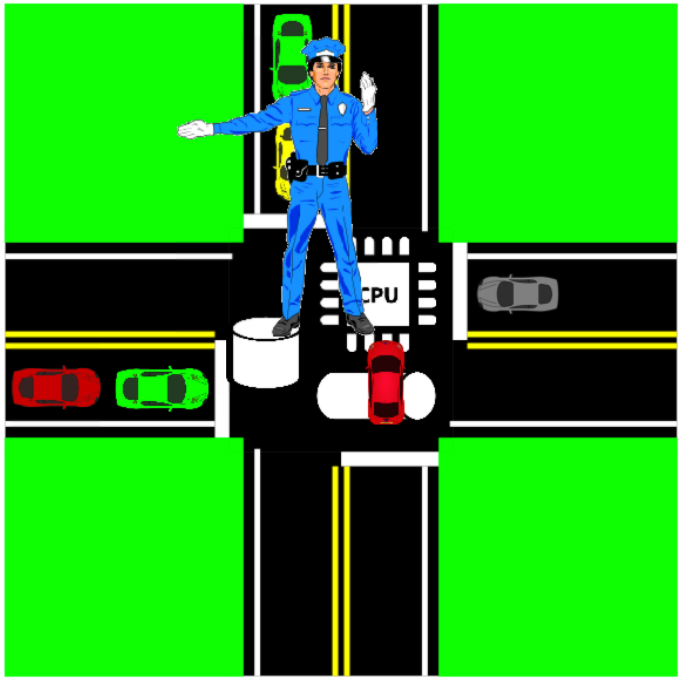
- If there is no I/O control, then
  - In case of contention, every I/O request of an unlucky process may have to wait for a lot of time before being served
  - Because the I/O stack and the drive itself prefer other requests
  - E.g., sequential I/O is highly preferred against random I/O

So *blk-mq* equipped with I/O schedulers



I/O schedulers

- *blk-mq* tries to provide
  - low-latency
  - responsiveness
  - fairness
- Through I/O schedulers: BFQ, MQ-DEADLINE, NONE, KYBER



I/O policies

- I/O schedulers try to do the right thing automatically
- For some goals, this may not be enough
- One notable example is exactly
  - Guaranteeing a minimum bandwidth to every client of a service
- I/O policies are used for a goal like this
  - *throttling* and *proportional share*
  - they work on top of I/O schedulers
    - *throttling* is an independent mechanism
    - *proportional share* expects an appropriate proportional-share scheduler to implement the policy

Standard solution to control I/O

- BFQ implements the proportional-share policy in *blk-mq*
- But BFQ is still a newcomer
- Virtually no company knows much about proportional share on top of BFQ
- With *blk-mq*, throttling is the only Linux mechanism used for controlling I/O
- So let's solve our control problem with throttling
- Second part of the demo: `https://youtu.be/V6pXtwvfh4U`
  - Neglect BFQ results for the moment

What happened to the speed of the drive???

- 80% of the speed is thrown away by throttling
- Like if the drive was five times as slow!

Unsolvable problem? Let's see with BFQ

- Let's focus on BFQ performance in the demo: `https://youtu.be/V6pXtwvfh4U`

Newfound speed and control with BFQ

- That's all about servers
- Let's move to personal systems

Personal systems

- Servers evidently need proper configuration
- But, on a personal system,
  - do we have to create groups and configure weights
  - to enjoy the above I/O-control benefits?
- Not with BFQ
  - BFQ tries to provide the best possible quality of experience
  - With **no configuration** needed

BFQ main features

- High application and system **responsiveness**
  - Low **lag** in typical Android terminology
- Low **drop rate** in audio/video playback
- High **throughput**

Both

- the high need for these features, and
- the actual performance of BFQ

are shown in many diagrams, presentations and, especially, demos

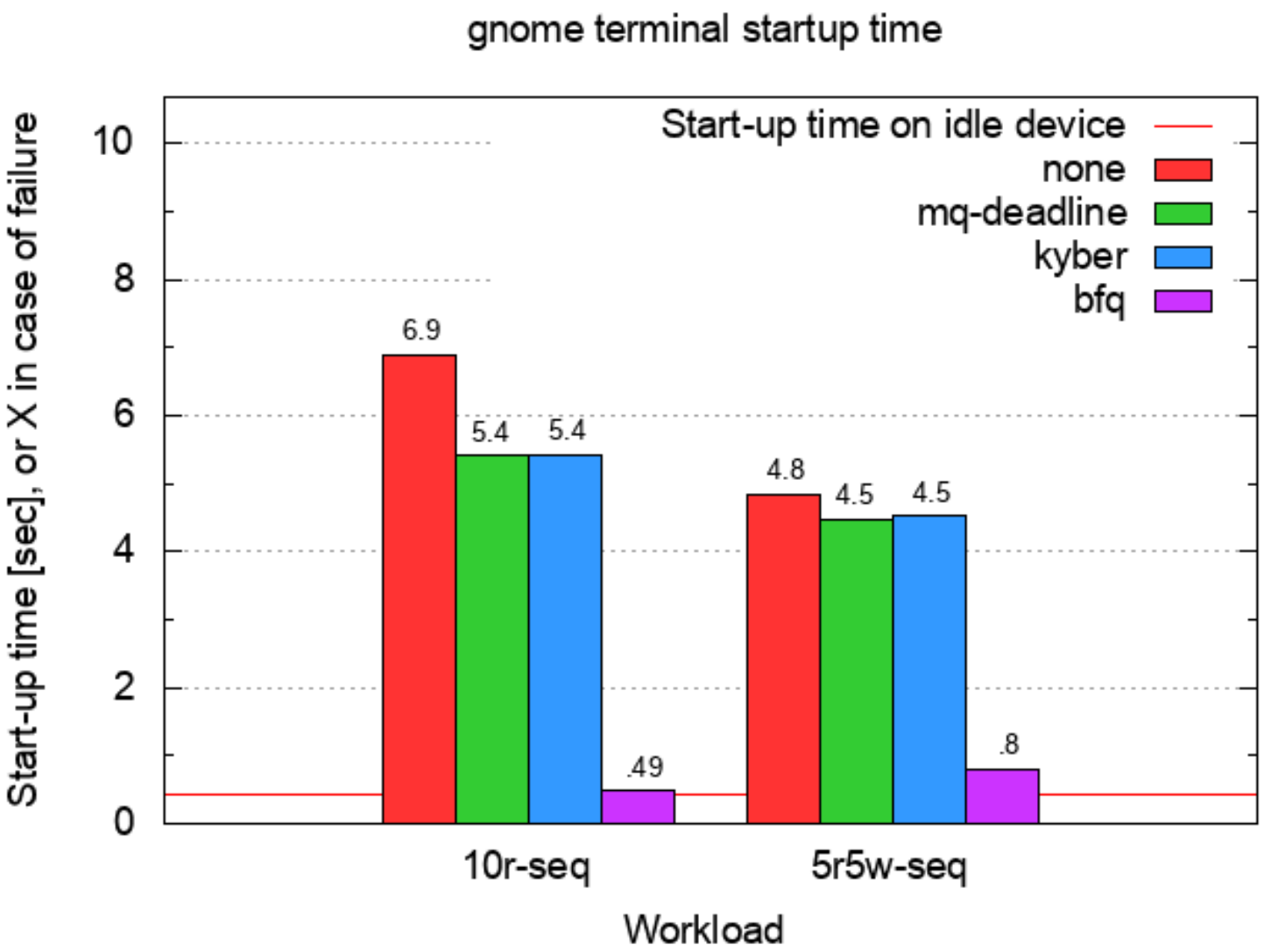
Demos

- PC (Linux distribution) demo with an HDD:
  `https://youtu.be/ZeNbS0rzpoY`
- PC (Linux distribution) demo with an SSD:
  `https://youtu.be/1cjZeaCXIyM`
- Embedded system (Hikey running Android) demo:
  `https://youtu.be/ANfqNiJVoVE`
- Embedded system (Hikey running Debian) demo:
  `https://youtu.be/gyM_JJtIvP0`
- Android Phone (Pixel 2) demo:
  `https://youtu.be/Ai3EPDpdsvY`

Responsiveness (lag) and playback quality

- Let's see BFQ at work with the SSD demo
  - Same storage device as in the demo on I/O control
  - Old Linux and BFQ version, still in legacy *blk*
  - Relative performance of BFQ now much better
    - Shown in a diagram in next slide
  - The goal of the demo is to give you a feeling of what the numbers in the next slide mean
- So, let's watch the PC (Linux distribution) demo with an SSD: `https://youtu.be/1cjZeaCXIyM`

Comparison on Linux 4.18 with *blk-mq*



That's all folks!

Thanks for your attention!