A comparison between BFQ and io.latency

Paolo Valente
paolo.valente at linaro.org

August 16, 2019

1 Motivation and goal of this comparison

The io.latency controller is arguably the most effective solution to the non-trivial problems reported in the patches and articles on the controller [1, 2]. The most challenging testbed considered for the io.latency controller is that of high-priority workloads seriously troubled by a low-priority web server with a slow memory leak [2].

Still, the io.latency controller is a general latency-control solution. So what about other, common use cases? For instance, what about servers or cloud nodes serving multiple clients, without memory-leak issues?

2 Test hardware and software

In particular, how does the io.latency controller perform, compared with BFQ, on hardware ranging from modern, harder-to-control SSDs, to higher-latency devices as HDDs? To find out, we compared the two solutions on the following three drives:

- SAMSUNG NVMe SSD 970 PRO: hard to control because of deep, multiple queues, and a sophisticated logic to handle and reorder enqueued commands;
- PLEXTOR SATA PX-256M5S SSD: lower queue depth, but higher latencies;
- HITACHI HTS72755 HDD: high latencies.

The OS was Ubuntu 18.04, on a Linux 5.3-rc4.

3 Scenario

For brevity, hereafter we write io.latency to mean io.latency controller.

We considered two workloads defined as follows:

- One process, the *target*, performing 4 KB random sync reads, with the time pattern defined below. We opted for random I/O for the target, because this it the type of I/O that incurs highest in-drive latencies, and that suffers from the highest interference by concurrent I/O, if the latter is sequential.
- While the target does its I/O, 15 processes, the *interferers*, perform sequential reads in the first workload, and sequential writes in the second workload. We opted for sequential I/O because this is the type of I/O that generates most interference on concurrent I/O (as sequential I/O si privileged in both the OS and the drive, and easily consumes most of the drive's bandwidth).
- In the write workload, writes are *fdatasynced*, again to generate maximum interference.
- When I/O is controlled by io.latency, both the target process and each interferer are encapsulated in a separate, single-process group, as this controller is to be configured on a per-group basis.
- In all configurations but that of HDD as storage device, and no control or io.latency as I/O-control solution (see below), the target does I/O only for two seconds for each test run. The duration of the I/O grows to ten seconds in the above HDD special cases. We consider this time pattern because it both matches that of latency-sensitive applications, and causes latency to depend on the I/O policy or schedule enforced in block layer. A detailed explanation is available in the appendix.
- Each workload finishes when the target finishes doing I/O.

For each workload, we measured the time taken by each read operation performed by the *target* process, while the workload was controlled by one of the following configurations of I/O policies and schedulers:

none-none No I/O policy enforced and no I/O scheduling performed. This combination is used as a reference, to have an idea of the latency incurred in case of no I/O control.

lat-none io.latency as I/O policy; and none as I/O scheduler, so that io.latency is in full control. The target latency for the group containing the target process is set to just 10 μs, i.e., to have the controller try to guarantee the lowest-possible latency on each of the three drives.

none-bfq No I/O policy enforced, and BFQ as I/O scheduler. The target process is assigned to the real-time I/O-priority class, so as to aim at the lowest-possible latency also in this case.

Hereafter we call just latency the time taken by a read operation performed by the target process, and we call *latency-measurement run* the taking of latency measurements during the execution of one of the above workloads, with one of the above configurations selected as I/O-control configuration.

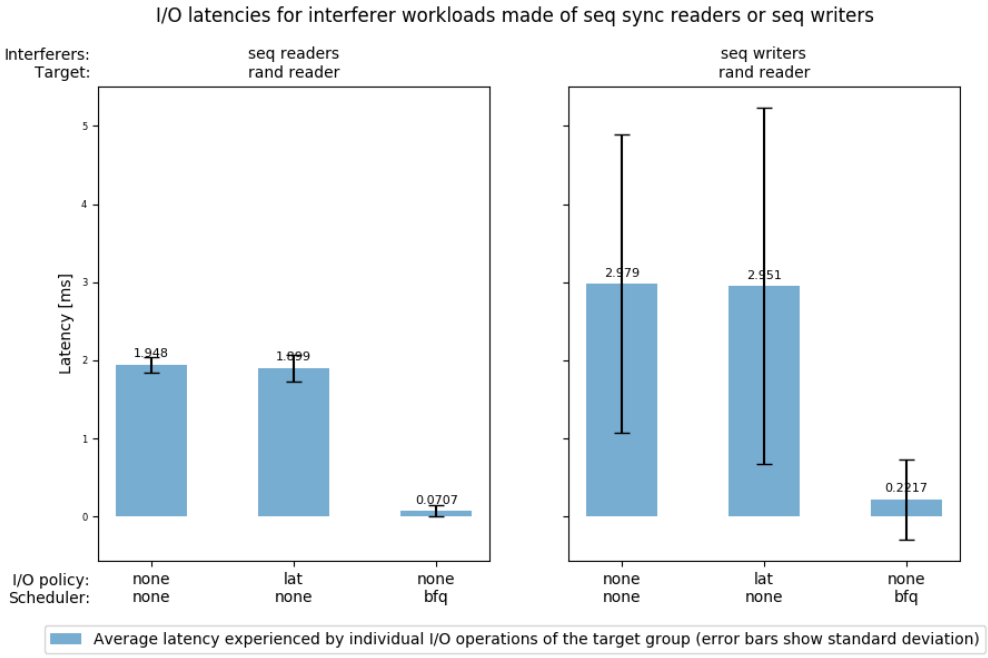

Figure 1: Individual-I/O latency on the SAMSUNG SSD (lower is better).

4 Results

For each combination of the workloads and I/O-control configurations reported in the previous section, we repeated the latency-measurement run ten times. Our full set of measurements can be reproduced with the S benchmark suite [3], by executing this command [4].

Statistics values (minimum, maximum, average, standard deviation) did not vary much across repetitions. More importantly, the relative performance of the I/O-control configuration at hand (compared with that of the other two configurations) was about the same across all repetitions. As a consequence, we report, for each combination of workload and I/O-control configuration, the average latency and standard deviation obtained for just one representative latency-measurement run.

Figure 1 shows results for the SAMSUNG SSD. With both readers and writers as interferers, io.latency has the same performance as no I/O control, with an average latency ranging from ∼2 to ∼3 ms. With BFQ, average latency is ∼27 times as low, and equal to ∼70 μs with readers as interferers; it is ∼13 times as low, and equal to ∼220 μs with writers.

The situation gets a little worse for io.latency with the PLEXTOR SSD, as shown in Figure 2. With readers as interfers, average latency with io.latency is even higher than without I/O control: ∼57 ms against ∼39 ms. With writers, average latency is ∼3 ms for both configurations. As with the SAMSUNG SSD, average latency is much lower with BFQ. With readers as interferers, BFQ's average latency is equal to ∼74 μs, and

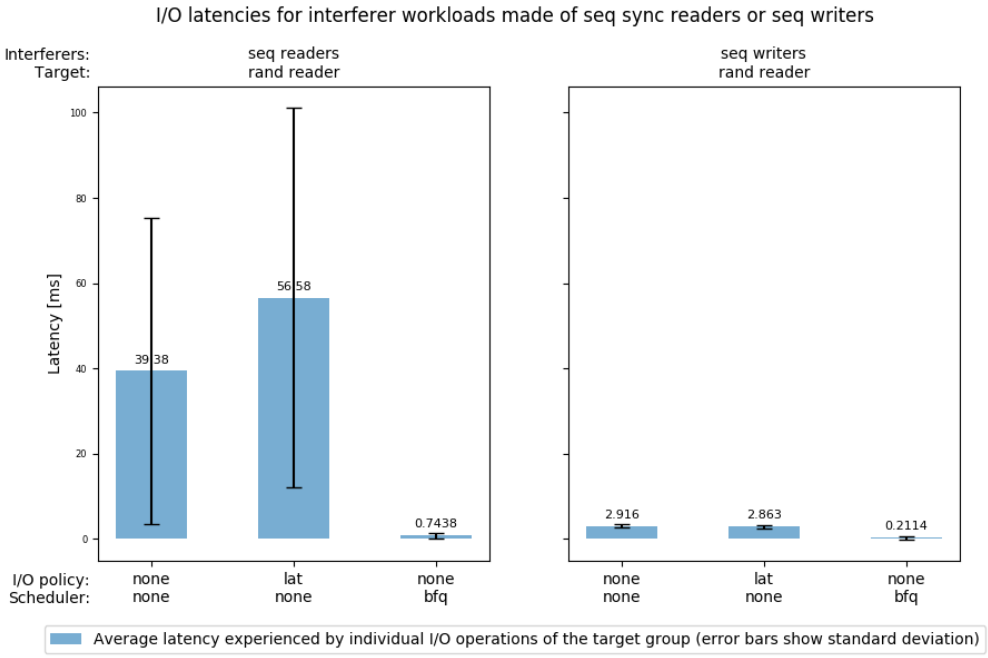

Figure 2: Individual-I/O latency on the PLEXTOR SSD (lower is better).

is from ∼53 to ∼76 times as low than that of the other two solutions; with readers as interferers, it is ∼21 μs, and ∼14 times as low as that of the other two solutions.

Yet the worst scenario for io.latency is that with the HDD and readers as interferers. As reported in Figure 3, average latency reaches ∼4.5 seconds, as it happens without I/O control. Latency is much lower with writers as interferers, ∼78 ms, even if it is still higher than without I/O control (∼56 ms). As on non-rotational devices, latency is much lower with BFQ: ∼11 ms, i.e., ∼420 times as low, in case of readers as interferers, and ∼13 ms, i.e., from ∼4 to ∼6 times as low.

Appendix: time pattern of the target process' I/O

The target does I/O only for two or ten seconds, as this is an effective time pattern for measuring the latency guaranteed by BFQ or the io.latency controller to time-sensitive tasks. This fact follows, first, from that latency-sensitive tasks typically generate *occasional* I/O, i.e., generate small batches of I/O *sporadically*, or just once. More precisely, a batch is sporadic if it starts after the previous batch has been completed. Consider, e.g., the reading of the frames/samples of a video/audio, or the loading of an application. In contrast, tasks needing continuous I/O for a relatively long time (such as a large-file copy), are unlikely to be sensitive to the latency experienced by single I/O operations or small batches of operations.

Secondly, occasional I/O could be generated, at a finer grain, in two ways. First, by

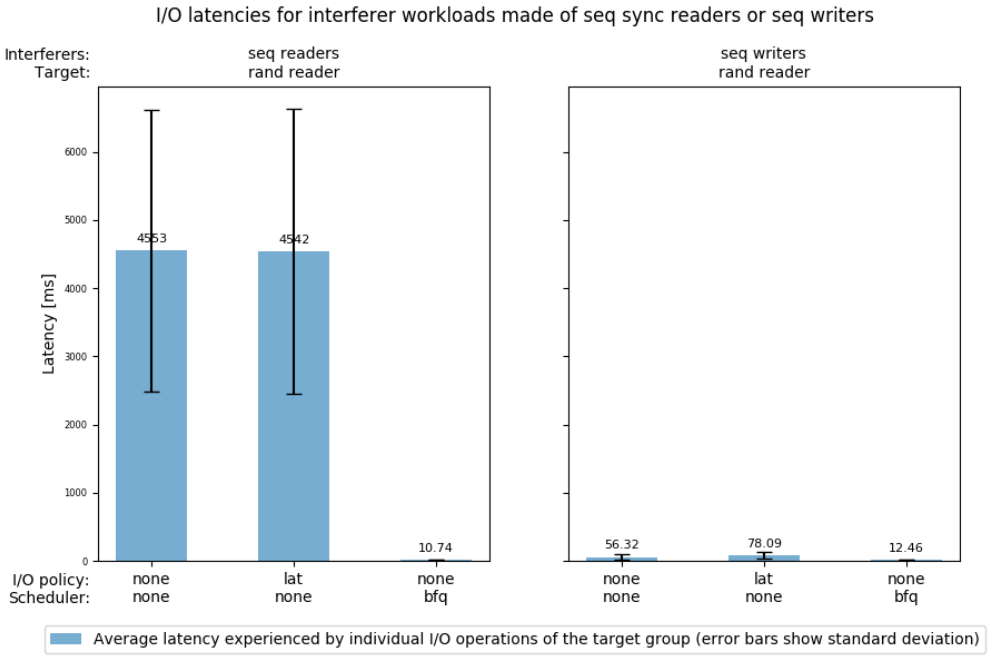

Figure 3: Individual-I/O latency on the HITACHI HDD (lower is better).

generating single I/O operations sporadically. Second, by doing I/O continuously for a short time, and then restarting at some later time, after the last batches of I/O has been completed. With the first pattern, the SAMSUNG SSD (re)schedules commands internally so as to guarantee an average latency in the order of 1 ms to target I/O, regardless of the I/O policy enforced in the block layer. So, on the bright side, no I/O control is needed to bound latencies. On the downside, there is actually no chance to control I/O so as to reduce latency below 1 ms. The same situation, of course with different values, occurs on the PLEXTOR SSD, in case interferers do writes.

Concerning the HDD, the worst-case latency of single random reads in an HDD is so high that the target should issue I/O at a negligible rate for single I/O operations to be sporadic. Such a rate should be so low to be unlikely to represent many significant workloads.

If, instead, a process generates a continuous flow of I/O for a while, then latency does happen to depend on the I/O policy enforced in the block layer. For this reason, in this article we report results for this second pattern. In this respect, one second of reads would have been enough, but with more variable results, and, as we verified, with the same average values as with two seconds. The duration of the I/O grows to ten seconds in the two extra HDD cases, because latency happens to be so high, that at least ten seconds are needed to collect more than just one latency sample.

References

[1] [Online]. Available: https://lwn.net/ml/linux-kernel/20180703151503.2549-13-josef@toxicpanda.com/

[2] [Online]. Available: https://lwn.net/Articles/782876/

[3] [Online]. Available: https://github.com/Algodev-github/S

[4] `cd S/run_multiple_benchmarks.sh && sudo ./run_main_benchmarks.sh latency "none-none lat-none none-bfq" "" "" "" "" 10`.