The status quo & challenge of Linux IO Isolation

Zheng Liu From Alibaba Cloud

Agenda

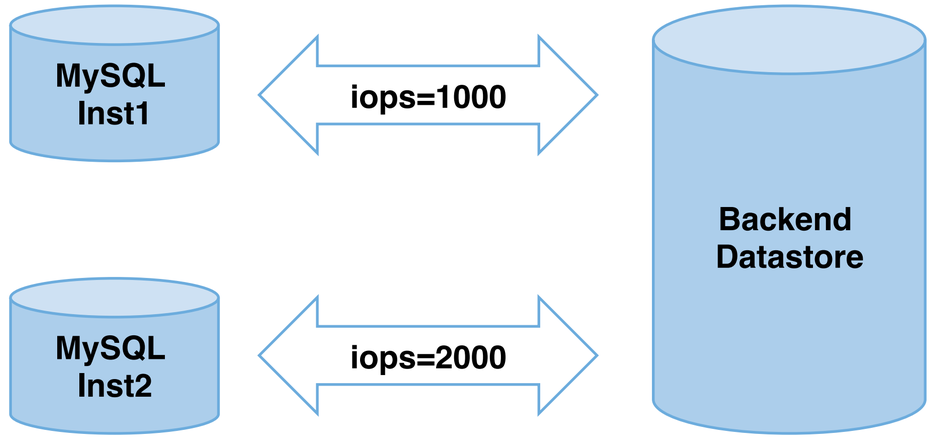
1. Why Need IO Isolation
2. Introduction of Existing Solutions
3. The Insufficiency of Existing Solutions
3. Optimizing Work & Challenge

Agenda

1. **Why Need IO Isolation**
2. **Introduction of Existing Solutions**
3. **The Insufficiency of Existing Solutions**
3. **Optimizing Work & Challenge**

Why Need IO Isolation

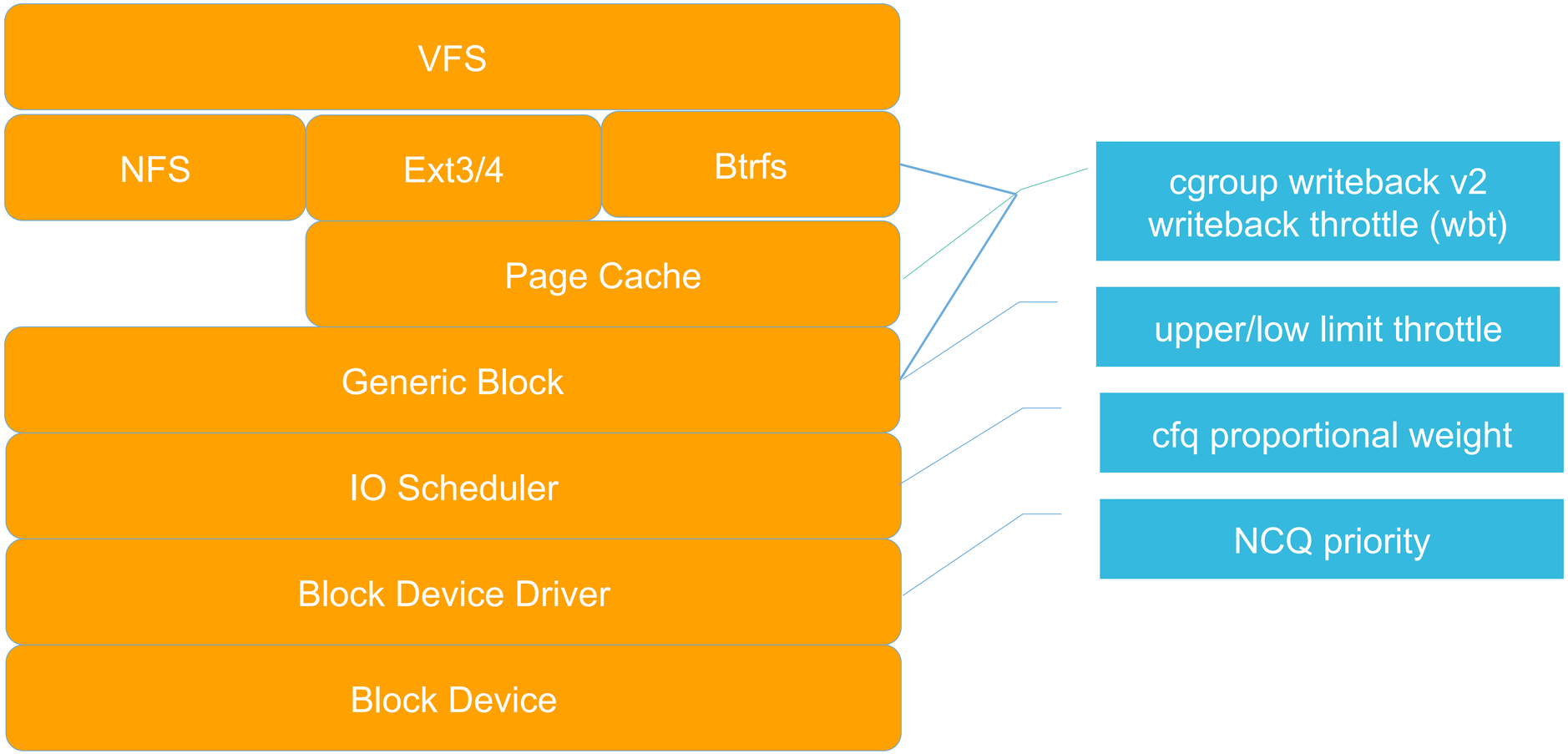
- Mission critical application vs normal application, e.g. Online vs Offline.
- Differentiate spec of VMs.
- Differentiate users, e.g. VIP vs normal.
- Differentiate spec of MySQL instances.
- ……



Agenda

1. Why Need IO Isolation
2. Introduction of Existing Solutions
3. The Insufficiency of Existing Solutions
3. Optimizing Work & Challenge

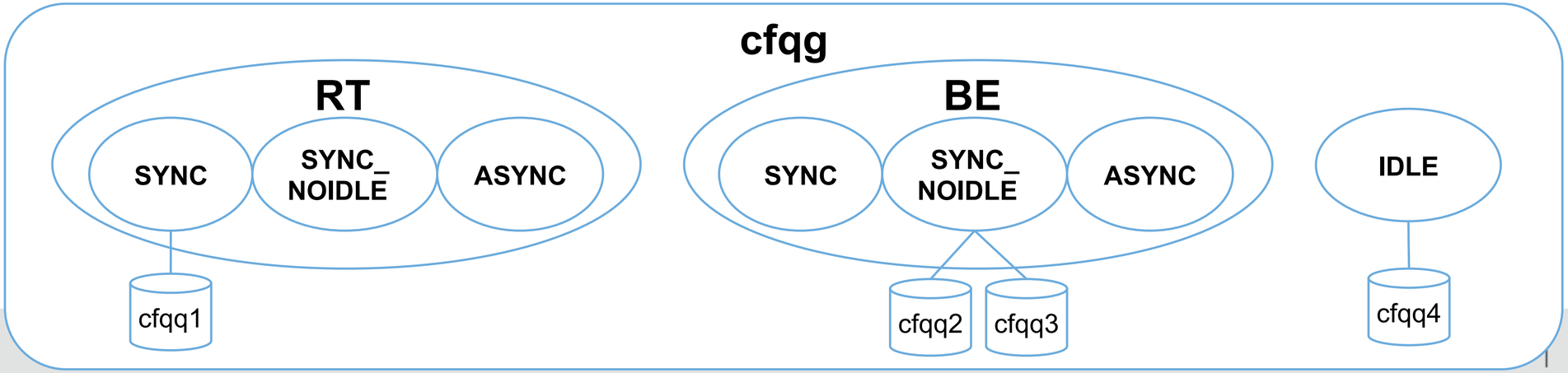
Existing Solutions Hierarchies



CFQ Proportional Weight

- Since v2.6.33, authored by Vivek Goyal.
- Proportional policy of cgroup blkio controller, based on cfq io scheduler, time slice/iops accounting.
- Each cfqg contains several cfqq service trees.
- All cfqgs are on the same global cfqg service tree.
- Each cfqg contains a configured weight value, and cfqg's vdisktime decides its position on the cfqg service tree, charged according to the weight (larger weight, smaller vdisktime).
- cfqq schedule policy:

a) choose the most left (smallest vdisktime) cfqg on the cfqg service tree;
b) choose cfqq service tree: RT > BE > IDLE; contains lowest rb_key cfqq;
c) choose cfqq.



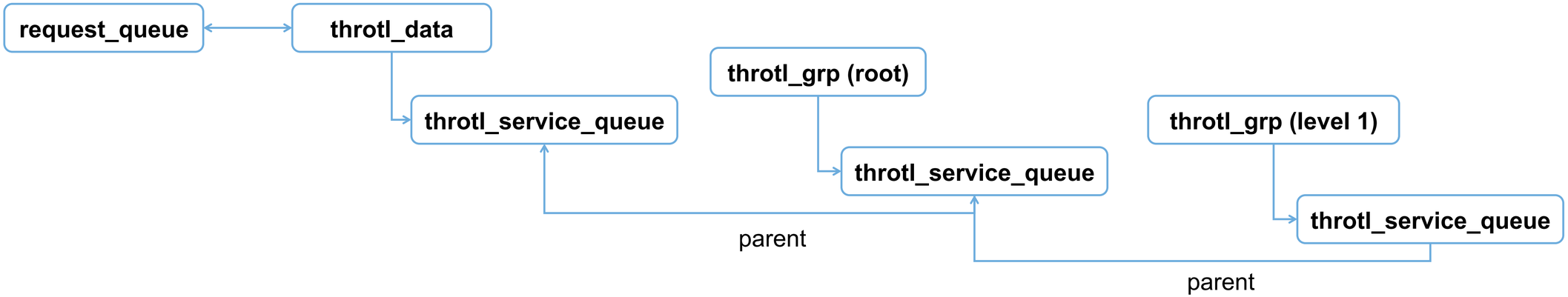
Block Throttle – max limit

- Since v2.6.37, authored by Vivek Goyal.
- Upper limit policy of cgroup blkio controller, when doing make request checks before IO scheduler.
- Bandwidth/iops accounting.
- Check the dispatched ios/bytes in current slice:

a) if within limits, charge and dispatch directly;

b) if above limits, queue to throttle group and calculate sleep time, then try to schedule dispatch in the next slice.

- * Can only throttle sync/direct io.



Cgroup Writeback v2

- Since v4.2, authored by Tejun Heo.
- For throttling buffered write.
- Convert writeback code so that wb (bdi_writeback) operates as an independent writeback domain instead of bdi (backing_dev_info), and a single bdi can have multiple per-cgroup wb's working.
- Introduce inode_wb_link so that an inode can be associated against multiple wb's as it gets dirtied by different cgroups, and use boyer-moore algorithm to account to the right cgroup.
- With the cooperation between memcg and blkcg, now we can know the original process of the writeback io instead of kworker.

Writeback Throttle (WBT)

- Since v4.10, authored by Jens Axboe.
- For throttling writeback ios based on latency.
- Introduce a new request flag REQ_BACKGROUND to indicate it is backgroud (non-urgent) IO.
- Take inspiration in the CoDel networking scheduling algorithm, monitor latencies of in a defined window.

a) If the minimum latency in the above window exceeds some target, increment scaling step and scale down queue depth by a factor of 2x. The monitoring window is then shrunk to 100 / sqrt(scaling step + 1).
b) If latencies look good, decrement scaling step.
c) If we‘re only doing writes, allow the scaling step to go negative. This will temporarily boost write performance, snapping back to a stable scaling step of 0 if reads show up or the heavy writers finish. Unlike positive scaling steps where we shrink the monitoring window, a negative scaling step retains the default step==0 window size.

Block Throttle – low limit

- Since v4.12, authored by Shaohua Li.
- “Best effort throttling”, with guaranteed low limit and then try to use free bandwidth as more as possible.
- Introduce state machine with 2 states MAX and LOW, and dynamic adjust limit value and switch state if necessay according to running situations.

a) If no low limit configured, initialize state to MAX.
b) If has low limit configured, state switches to LOW and all cgroups run targeting to the their low limit. If all cgroups reach to their low limit, switch state to MAX and increase limit value targeting to max limit as well.
c) In state MAX, if has cgroup run below its low limit, decrease limit value till all cgroups run above low limit, and finally switch state to LOW if necessary.
d) Introduce idle detection and latency target mechanisms to identify the case that a cgroup cannot dispatch enough io, for the sake of using free bandwidth.

NCQ Priority

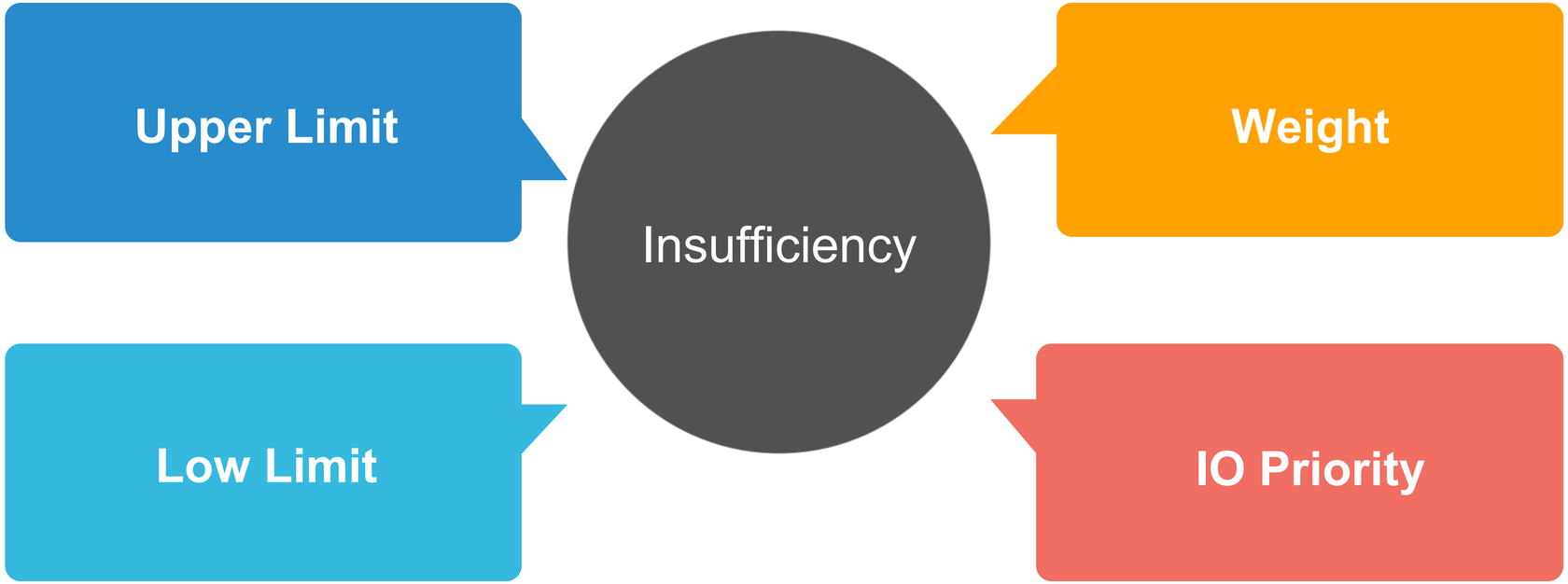
- Since v4.10, authored by Adam Manzanares.
- For improving tail latencies of workloads that use higher queue depths.
- Add iocontext priority to request and build ATA commands with high priority.
- Device should has NCO priority information support.
- Check if device support:

hdparm -I /dev/<device> I grep NCQ

Agenda

1. Why Need IO Isolation
2. Introduction of Existing Solutions
3. The Insufficiency of Existing Solutions
3. Optimizing Work & Challenge

Insufficiency of Existing Solutions



Upper Limit

- Hard limit:

a) If configure a high upper limit, isolation result won't be as good as we expect (disk bottleneck), e.g. disk bandwidth 400MB/s, cg1 400MB/s, cg2 200MB/s

b) If configure a proper low upper limit, cgroup cannot dispatch more io than its limit, even other cgroups are idle (waste bandwidth), e.g. disk bandwidth 400MB/s, cg1 300MB/s, cg2 100MB/s.

- Only under cgroup v2 can throttle buffered write.

Low Limit

- It is very hard to identify an idle cgroup:

a) think time: can only detect the cgroup which dispatches little io;

b) latency target: not friendly with end user; IOW, user cannot easily know the proper target latency to be configured

- Currently no consideration of cooperation with writeback (periodically io).
- It is currently an experimental feature implemented under cgroup v2.

IO Weight

- Tied to cfq io scheduler, not generic implementation.
- In many real user scenarios on SSD, we do not use cfq as the io scheduler:

a) SATA SSD – deadline as default;
b) NVMe SSD – none or kyber.

IO Priority

- Rely on device support, only part of SATA device support NCQ priority now.
- Read latency can be impacted a lot with write in mixed workload, e.g. read tail latency on NVMe SSD.

Agenda

1. Why Need IO Isolation
2. Introduction of Existing Solutions
3. The Insufficiency of Existing Solutions
3. Optimizing Work & Challenge

Optimizing Work & Challenge

- Low limit

a) Continuously improve idle detection in real user scenarios, e.g. automatically learn how to configure proper latency target.

b) Cooperation with writeback io throttle, optimizing the upgrade/downgrade logic with periodically flushed writeback io.

Optimizing Work & Challenge

- IO Weight

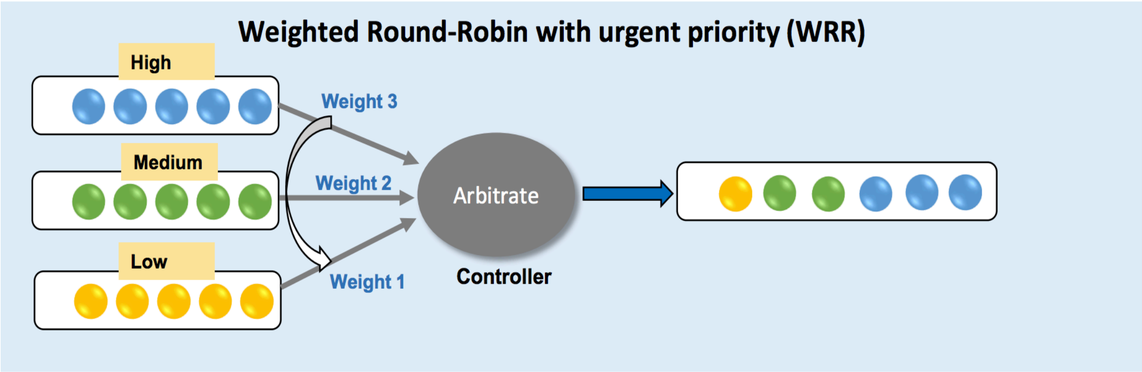
a) Implement weight control independent of io scheduler (work for blk-mq);
b) Shaohua Li tries the unified work, choose block throttle as the candidate to implement proportion policy;
c) Estimate bandwidth: bandwidth = current bandwidth / disk utilization, and always slightly over estimate to utilize disk more;
d) In a cgroup, share = weight / total weight, bandwidth limit = share * estimated disk bandwidth;
e) Feedback and dynamic adjust cgroup share to solve inactive cgroup.

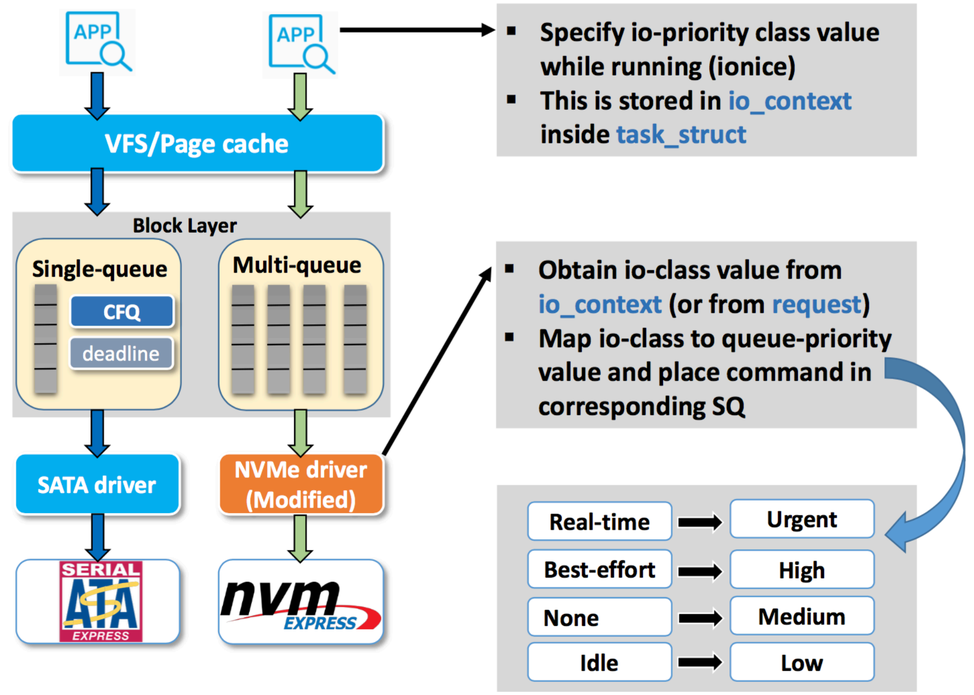
Optimizing Work & Challenge

- IO Priority

a) Support io priority control on more kinds of devices;

b) Improve read tail latency through WRR (Weighted-Round-Robin-with-urgent-priority) on NVMe SSD (already has paper posted in HostStorage ’17)：





MORE THAN JUST CLOUD | Alibaba Cloud