STORAGE DEVELOPER CONFERENCE

SDC21

BY Developers FOR Developers

Virtual Conference
September 28-29, 2021

A SNIA Event

Be On Time: Command Duration Limits Feature Support in Linux™

Damien Le Moal, Western Digital Research

Outline

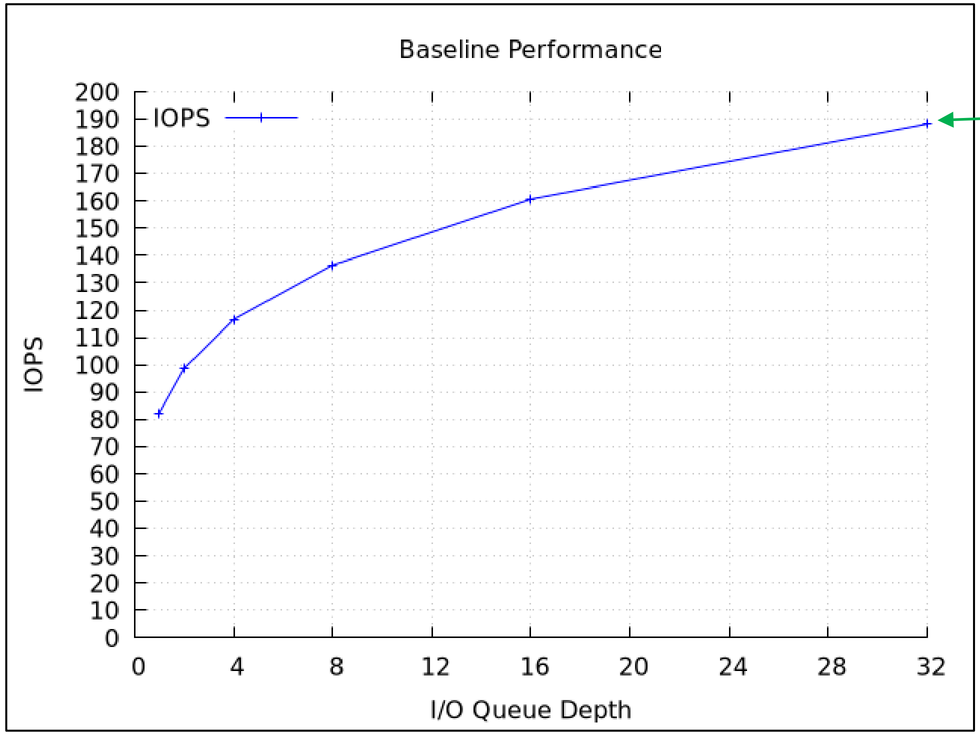
- HDD Performance and I/O Latency
  - HDD performance characteristics
  - I/O latency control
    - Controlling latency with the queue depth
    - ATA NCQ Priority feature
- Command duration limits
  - Feature overview and Linux integration
  - Experimental results
- Conclusion

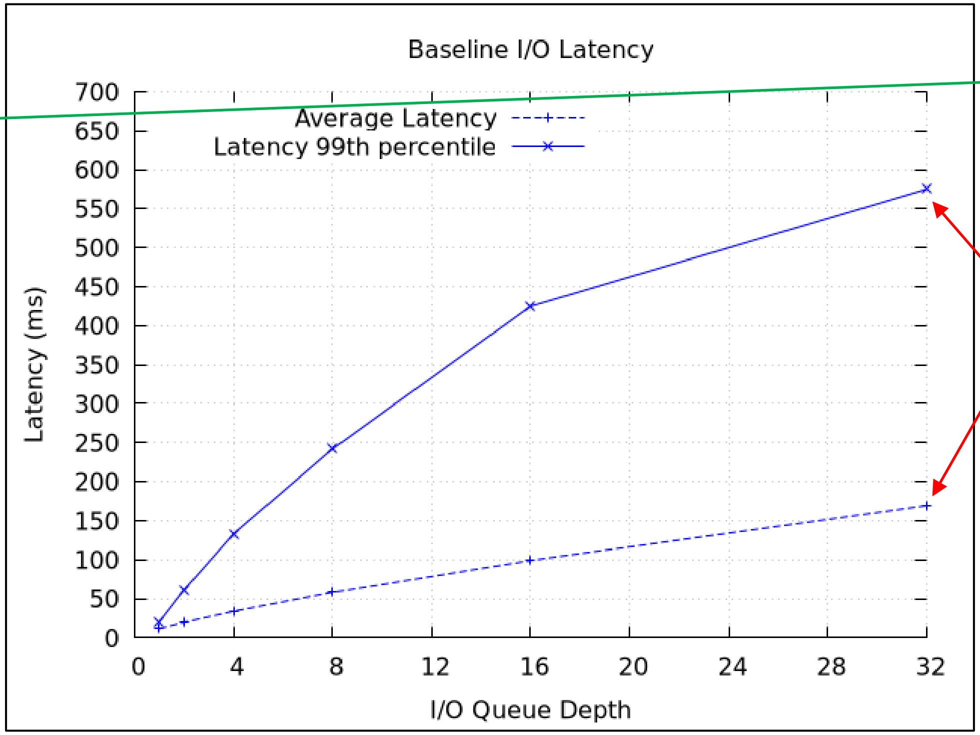
HDD Performance and I/O Latency

IOPS and I/O tail latency increase with queue depth

HDD Performance Characteristics

- Using HDDs at high queue depth increases performance
  - E.g. I/O rate increases from 81 IOPS at QD=1 up to 188 IOPS at QD=32
- But this comes at the cost of a significant increase in tail latency
  - E.g. Latency 99th percentile increases up to 575 ms at QD=32





Significantly higher IOPS at high QD

But higher average and tail latency at high QD

128KB Random Read Workload

STORAGE DEVELOPER CONFERENCE SDC 21

Controlling I/O Latency

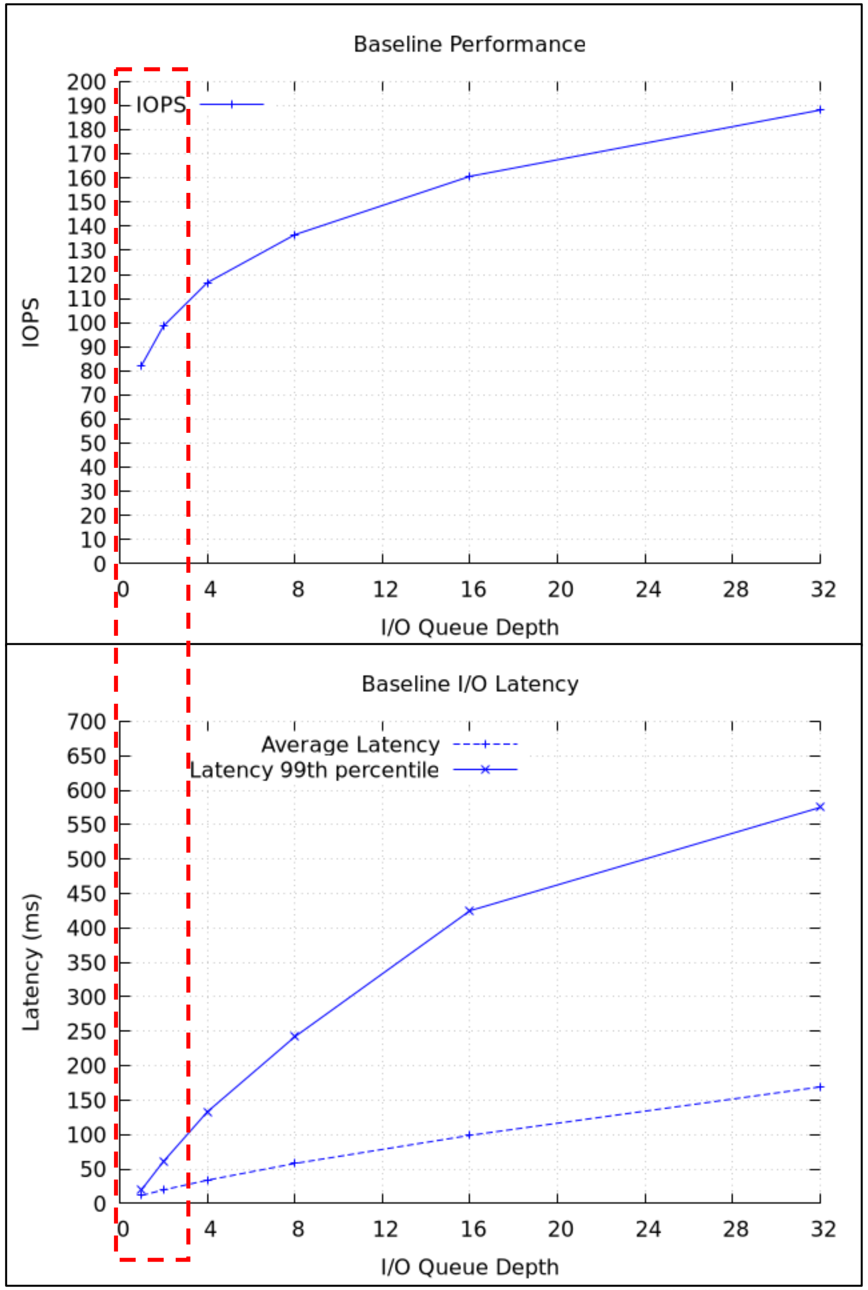
- I/O latency control is an important aspect of many storage applications
  - For implementing different user service levels and guaranteeing service quality
  - For the overall system performance
    - In particular for RAID and erasure coded systems where user I/Os are split over multiple drives
      - The drive with the lowest access time slows down the entire I/O execution time
- Several methods exist for controlling I/O latency at the device level
  - Control based on queue depth limits
    - Trade-off performance for tight control over I/O latency
  - ATA NCQ Priority Feature
    - Give hints to the drive for partial (best-effort) control over I/O latency with higher performance
- The new command duration limits feature provides a more advanced interface allowing users to precisely control I/O latency
  - Provides more detailed hints to the drive to enable an efficient and precise (on time) command execution

Queue Depth Based Control

Sacrificing performance for tight control over I/O latency

Queue Depth Based Control

- Storage applications can control tail latency by using the drive at a low queue depth
  - E.g. maintaining a latency 99th percentile under 100ms implies using the disk at QD=3 at most
    - Maximum performance drops to 110 IOPS
    - 40% decrease from QD=32 maximum IOPS
- Simple and efficient method
  - Widely used in the field
  - But increasing a system overall performance requires more drives
    - Can significantly increase the system cost



ATA NCQ Priority Feature

Higher performance with partial (best-effort) control over I/O latency

ATA NCQ Priority Feature

- First introduced with ACS-2 (2011)
  - No SCSI equivalent !
- Defines a high priority level for NCQ FPDMA read and write commands, in addition to the normal/no priority level
  - Allows the user to indicate to the drive the commands that must be executed ”quickly”
  - Best effort execution
- The standard is vague about what “quickly” means
  - Vendors can implement various command execution scheduling policies with different characteristics
    - This results in drive behavior differences between drive vendors and drive models

ATA NCQ Priority in Linux

- Supported in Linux since kernel 4.10
- Relies on Linux I/O priority API
  - Initially defined for kernel block I/O schedulers
  - Three priority classes are defined
    - Real-time, best effort and idle
  - NCQ high priority level is set for commands serving I/Os using the real-time priority class
- I/O priorities can be assigned directly by the user
  - Per user, per process group and per process
    - *ioprio_set()* system call and cgroups
  - Per asynchronous I/O (libaio and io_uring)
    - *aio_reqprio* field of *struct aiocb*
  - In-kernel I/O path propagates the I/O priority to the block IO scheduler and to the device driver

include/linux/ioprio.h

```
...
#define IOPRIO_PRIO_VALUE(class, data) \
                    (((class) << IOPRIO_CLASS_SHIFT) | data)
...
enum {
     IOPRIO_CLASS_NONE,
     IOPRIO_CLASS_RT,
     IOPRIO_CLASS_BE,
     IOPRIO_CLASS_IDLE,
};

/*
 * 8 best effort priority levels are supported
 */
#define IOPRIO_BE_NR   (8)
...
```

Application

Per process

```
ioprio_set(IOPRIO_WHO_PROCESS, 0,
                   IOPRIO_PRIO_VALUE(class, level));
...
read()
```

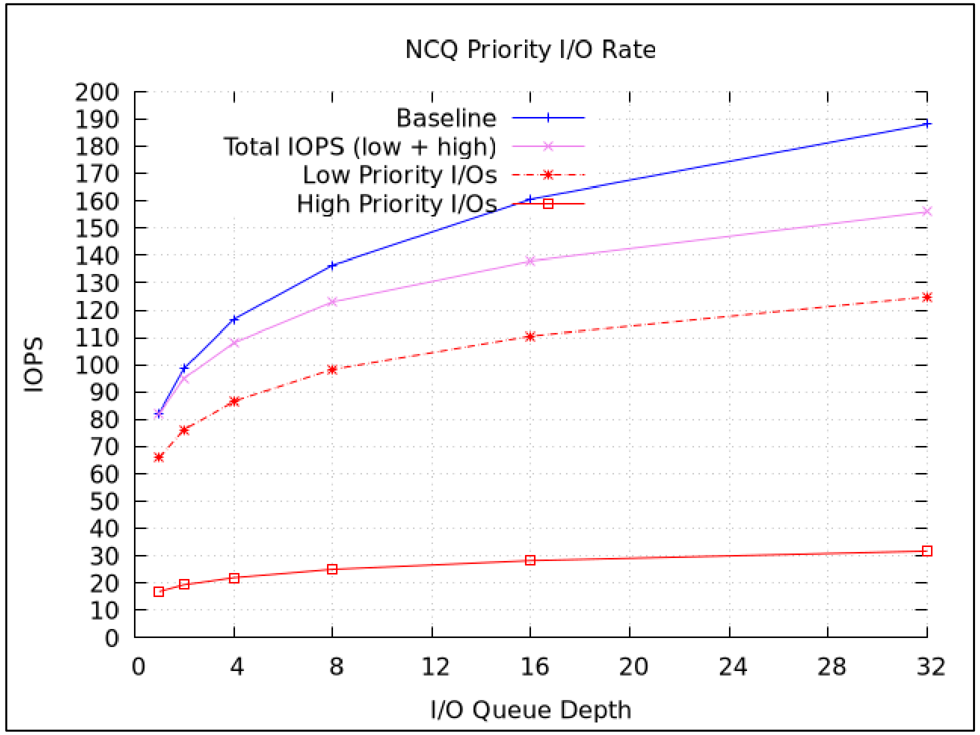
Per asynchronous I/O

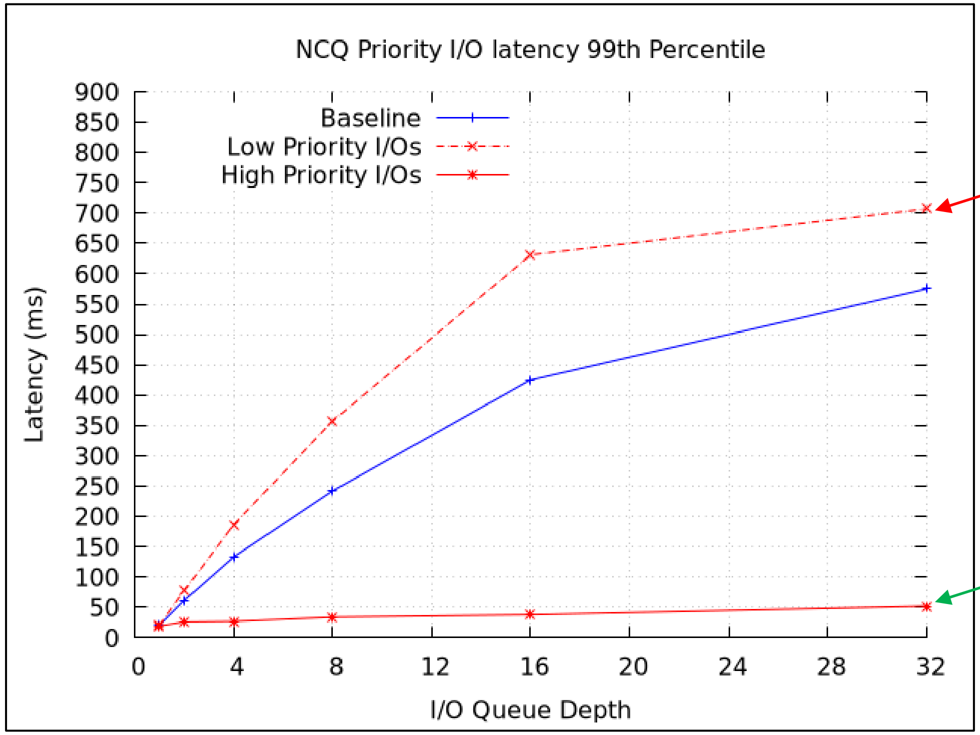
```
...
aio->aio_reqprio = IOPRIO_PRIO_VALUE(class, level);
aio->u.c.flags |= IOCB_FLAG_IOPRIO;
io_submit();
```


STORAGE DEVELOPER CONFERENCE SDC 21

ATA NCQ Priority Use Example

- With 20% of high-priority I/Os, a low (50ms) tail latency is maintained for all queue depths
  - 18% decrease of the maximum IOPS (156) from the baseline maximum performance (188)
  - Tail latency (99th percentile) of low priority I/Os significantly increases from the baseline





Higher tail latency for low priority commands

128KB Random Read, 20% high priority I/Os

Fast execution of high-priority commands

Command Duration Limits

Higher performance with fine control over I/O latency

Command Duration Limits (CDL) Feature

- New SCSI and ATA command feature currently in draft state
  - T10 SPC-6 (SCSI) and T13 ACS-5 (ATA)
    - A simpler version of the feature is defined in T10 SPC-5
- CDL defines Duration Limit Descriptors (DLD)
  - 7 DLDs for read commands and 7 DLDs for write commands
    - ACS-5: log page 18h
    - SPC-6: mode page 0Ah, subpages 07h and 08h
  - 3 bits for read and write commands to indicate to the disk the duration limit descriptor (DLD) to apply to the command
    - For backward compatibility, descriptor 0 means “no limit”
- The user can change a drive DLDs to adjust command latencies for the target workload
  - MODE SELECT for SCSI and WRITE LOG [DMA] EXT for ATA
  - This allows defining a similar behavior for different drive models from different vendors
    - Mitigate latency characteristics variations between drives

Command Duration Limit Descriptors

- A DLD defines 3 duration limits (timeouts) and a policy for each limit
  - A limit policy defines how a command should be handled if the limit is exceeded during the command processing
- Defined limits
  - **Command duration guideline:** Maximum command execution time target
    - The target command maximum latency
  - **Maximum inactive time:** Maximum command queuing time
    - Limits the time a command waits for execution
  - **Maximum active time:** Maximum media access time
    - Limits media access retries to bad sectors
  - At least one limit must be non-zero for the DLD to be valid

Command Duration Limit Descriptors

- Defined limit policies
  - **Best-effort**: the device tries to complete the command at the earliest possible time consistent with the limit value
    - No timeout errors
  - **Continue-limited**: if the limit is exceeded, continue execution of the command using the next valid descriptor
  - **Continue-no-limit**: if the limit is exceeded, continue execution of the command without any limit
  - **Complete**: if the limit is exceeded, complete the command with GOOD STATUS/ DATA CURRENTLY UNAVAILABLE
    - Fast fail the command while avoiding queue aborts with ATA NCQ
  - **Abort**: If the limit is exceeded, abort the command with ABORTED COMMAND/COMMAND TIMEOUT DURING PROCESSING or COMMAND TIMEOUT DURING PROCESSING DUE TO ERROR RECOVERY
    - Fast fail the command

Linux Integration

- Reuse Linux I/O priority API
  - Same per-context and per asynchronous I/O controls
- Introduce the new IOPRIO_CLASS_DL
  - The priority level value directly indicates the descriptor to apply to commands
    - Read and write commands DLD bits
- Modify the SCSI disk driver (sd) to set DLD bits based on the I/O request priority
  - Libata also modified to set the DLD bits in FPDMA READ/WRITE commands
  - Feature support discovery
- The drive descriptors are advertised to the user through *sysfs*
  - User applications can automatically choose the best descriptor for an I/O

include/linux/ioprio.h

```
...
#define IOPRIO_PRIO_VALUE(class, data) \
                (((class) << IOPRIO_CLASS_SHIFT) | data)
...
enum {
    IOPRIO_CLASS_NONE,
    IOPRIO_CLASS_RT,
    IOPRIO_CLASS_BE,
    IOPRIO_CLASS_IDLE,
    IOPRIO_CLASS_DL,
};

/*
 * 8 best effort priority levels are supported
 */
#define IOPRIO_BE_NR        8

/*
 * The Duration limits class allows 8 levels: level 0 for
 * "no limit" and levels 1 to 7, each corresponding to a
 * read or write limit descriptor.
 */
#define IOPRIO_DL_NR        8
...
```


Prototype Implementation

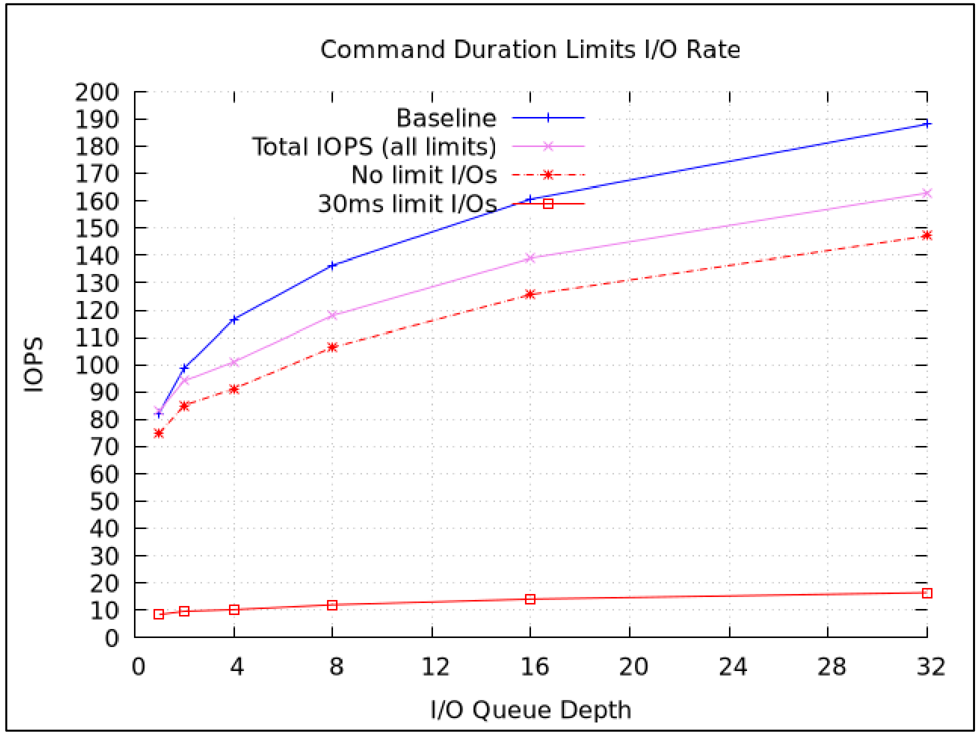
- Based on kernel 5.14 stable
- Latest fio version modified to allow specifying the new duration limit priority class (IOPRIO_CLASS_DL)
  - Per job priority definition
    - *--prio_class=4* options
    - *--prio=X* specifies that DLD X must be used
  - Per asynchronous I/O priority definition
    - *libaio* and *io_uring* I/O engines
    - *--cmdprio_percentage=P, --cmdprio_class=4* and *--cmdprio=X* options

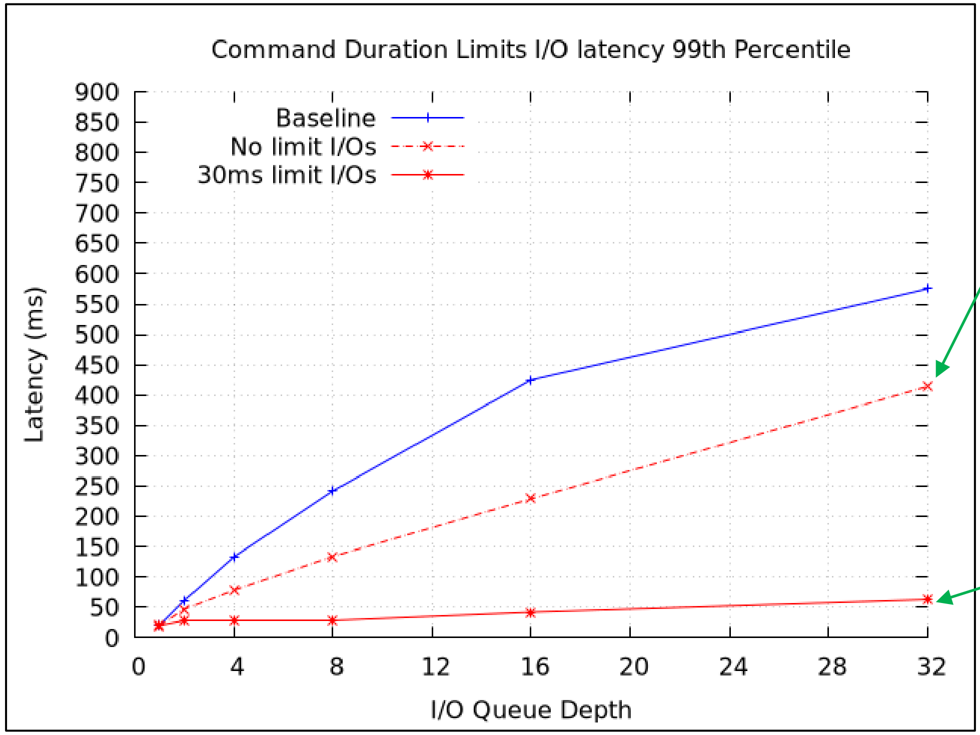
***sysfs* duration limits attributes**

```
# tree /sys/block/sdX/device/duration_limits/
/sys/block/sdk/device/duration_limits/
├── enable
├── read
│   ├── 1
│   │   ├── duration_guideline
│   │   ├── duration_guideline_policy
│   │   ├── max_active_time
│   │   ├── max_active_time_policy
│   │   ├── max_inactive_time
│   │   └── max_inactive_time_policy
│   ├── 2
...
│   └── 7
│       ├── duration_guideline
│       ├── duration_guideline_policy
│       ├── max_active_time
│       ├── max_active_time_policy
│       ├── max_inactive_time
│       └── max_inactive_time_policy
└── write
    ├── 1
    │   ├── duration_guideline
...
    └── 7
        ├── duration_guideline
        ├── duration_guideline_policy
        ├── max_active_time
        ├── max_active_time_policy
        ├── max_inactive_time
        └── max_inactive_time_policy
```


Command Duration Limits: NCQ Priority Like Use

- With 20% of commands using a short 30ms duration guideline and the best-effort policy, NCQ priority results can be replicated
  - A low 50ms latency is maintained for all queue depths
  - Better tail latency (99th percentile) for the “no limits” I/Os compared to NCQ priority “low priority I/Os”





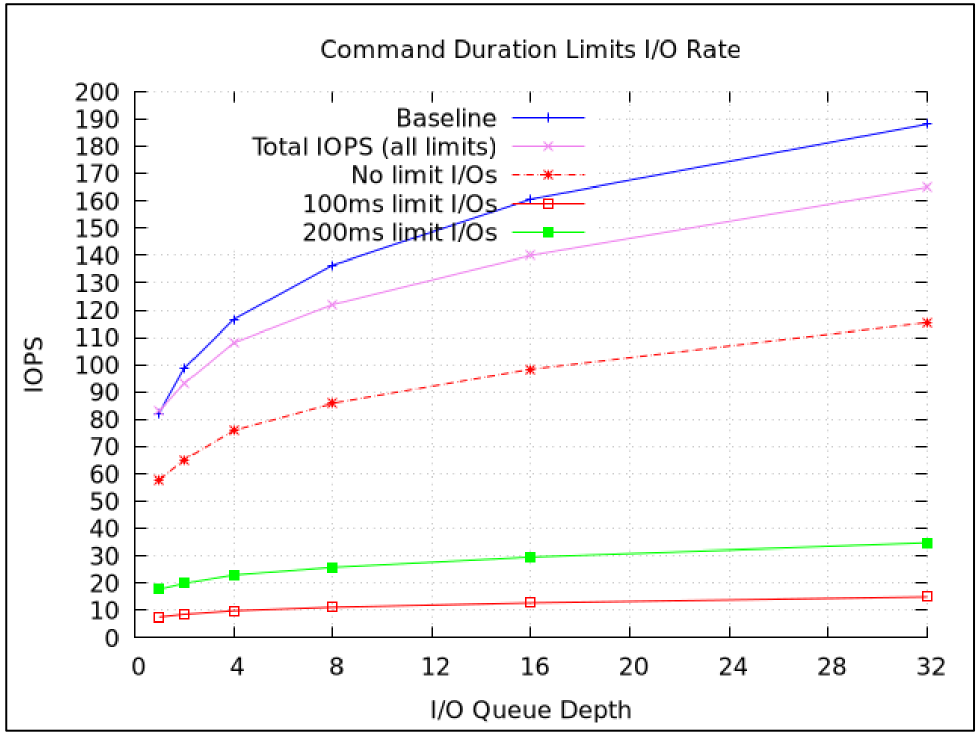
Better tail latency for no-limit commands

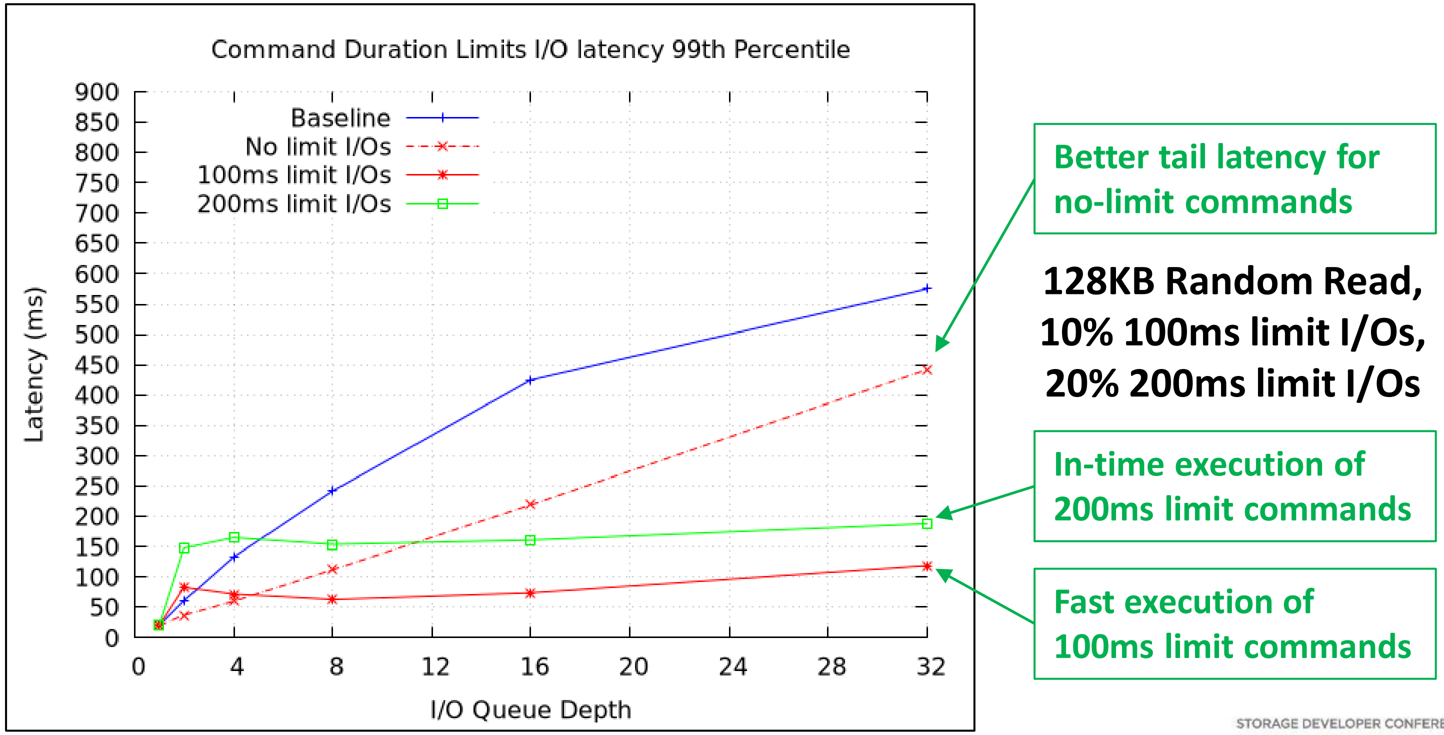
128KB Random Read, 20% 30ms limit I/Os

Fast execution of 30ms limit commands

Command Duration Limits: I/O latency Fine Control

- Users can combine different limits within the same workload
  - To achieve different service levels on the same device (e.g. 10% of short 100ms limit I/Os and 20% of longer 200ms limit I/Os
  - Small increase of tail latency for 100ms limit commands at queue depth 32





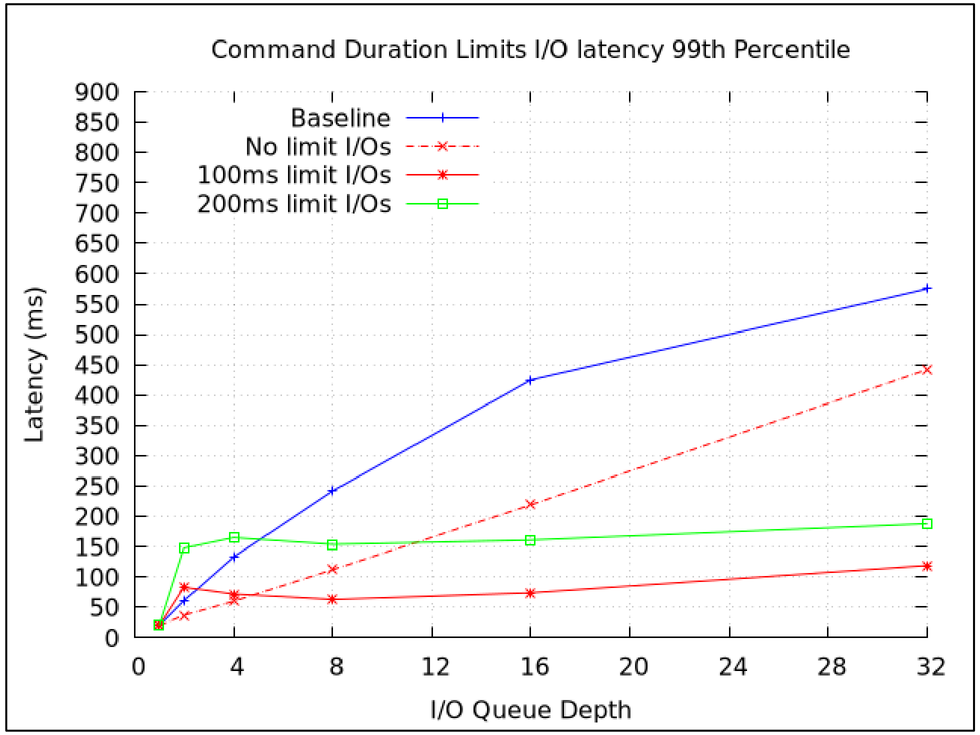
128KB Random Read, 10% 100ms limit I/Os, 20% 200ms limit I/Os

 |

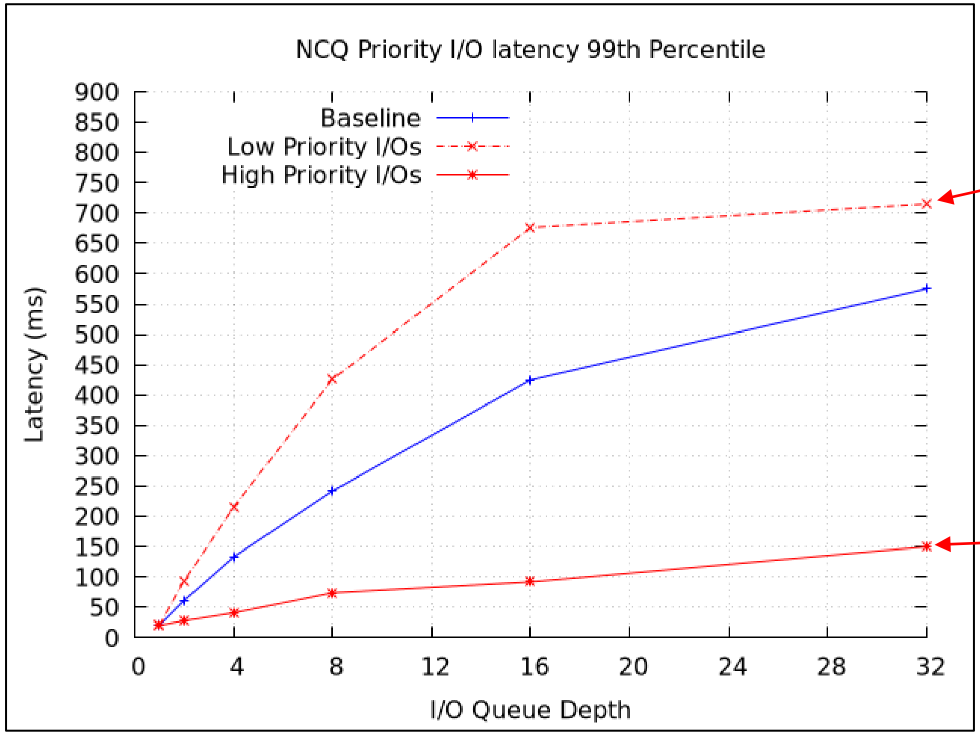
Command Duration Limits vs NCQ Priority

- Unlike CDL, fine control over I/O latencies is not possible with NCQ priority
  - The drive trades off performance in order to maintain a very low tail latency for all I/Os
    - This results in a lower maximum IOPS (159 vs 165) and higher tail latencies for low priority I/Os

Command Duration Limits



NCQ Priority



Higher tail latency for low priority commands

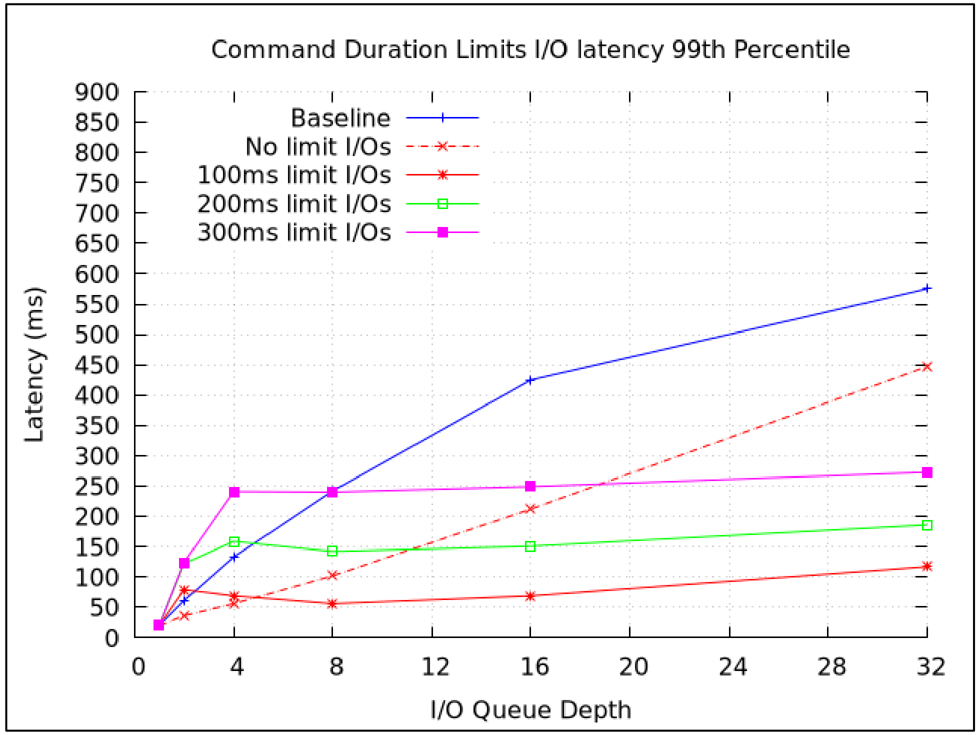
128KB Random Read, 30% high-priority I/Os

Higher latency at high queue depth

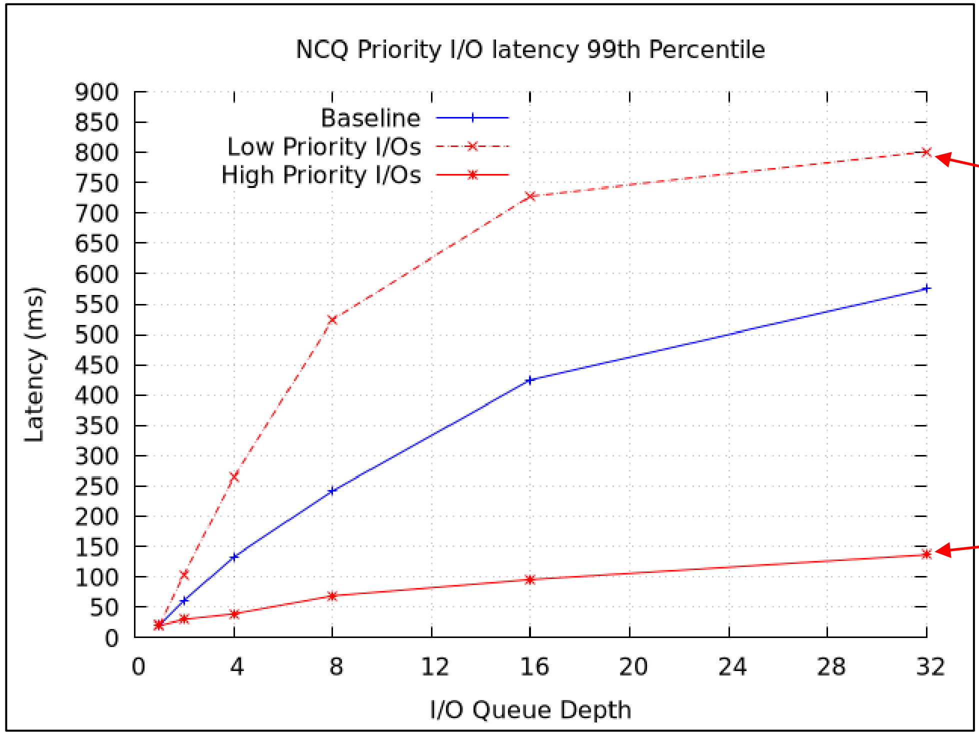
Command Duration Limits vs NCQ Priority

- CDL maintains good performance and tail latencies even for complex workloads
  - E.g. 4 service levels: 10% 100ms, 10% 200ms and 20% 300ms and 60% no-limit
  - With 40% of high priority I/Os, NCQ priority overall performance degrades further (158 vs 165)

Command Duration Limits: maximum IOPS 165



NCQ Priority: maximum IOPS 158



Higher tail latency for low priority commands

128KB Random Read, 40% high-priority I/Os

Higher latency at high queue depth

 |

STORAGE DEVELOPER CONFERENCE SDC 21

Conclusion

Concluding Remarks

- Command duration limits is more flexible than the ATA NCQ Priority feature
  - CDL with short duration limits can efficiently replace ATA NCQ priority
    - Avoid behavior variations between different drive vendors and drive models
  - The user can precisely control command latencies using multiple limit descriptors
- Re-using Linux I/O priority API simplifies application migration from NCQ Priority
  - Same API, only different priority values
- Deeper integration of CDL in Linux can further improve results
  - I/O schedulers and cgroups (latency controller)
- Upstream submission of this work dependent on the completion of the specifications
  - T10 (SPC and SAT) and T13 (ACS) work is on-going

Please take a moment to rate this session.

Your feedback is important to us.

STORAGE DEVELOPER CONFERENCE SDC 21