Multi-Queue Fair Queueing

Mohammad Hedayati\textsuperscript{1}, Kai Shen\textsuperscript{2}, Michael L. Scott\textsuperscript{1}, Mike Marty\textsuperscript{2}

\textsuperscript{1} University of Rochester
\textsuperscript{2} Google
Conventional I/O Design

O(1M IOP/s): less than 1µs per IOP
What can be done in less than 1µs?
Multi-Queue I/O Design
Multi-Queue I/O Design

Pros:

- Better scalability
- Better throughput

Cons:

- Challenges in preserving system-wide properties
  - e.g., Fairness
Overview

- Motivation
- Multi-Queue Fair Queueing
- Scalable Implementation of MQFQ
- Evaluation
- Conclusion
Fair Queueing

● Supports proportional sharing (weights)
● Work-conserving
● Handling of under-utilizing tasks
● Provable fairness bounds

● Additionally, we need to support Parallel Dispatch
Multi-Queue Fair Queueing

- MQFQ builds on SFQ(D) [Jin et al.’04]
  - **Start tag**: roughly, the task’s accumulated resource usage at request dispatch
  - Orders requests based on their start tags for fairness
  - Allows up to D parallel dispatches

- **Challenges**:
  - Strict ordering hampers scalability
  - Tracking global statistics requires cross-CPU communication
Multi-Queue Fair Queueing

- SFQ(D)

Process 1  Process 2  ...  Process n

increasing start tags
Multi-Queue Fair Queueing

Process 1  Process 2  ...  Process n

Throttling Threshold

increasing start tags

T increasing start tags
Multi-Queue Fair Queueing

Process 1  Process 2  ...  Process n

increasing start tags
Multi-Queue Fair Queueing

Process 1  Process 2  ...  Process n

Interrupt

increasing start tags
Bounded Unfairness

- For throttling threshold $T$ and $D$-parallel dispatch:

$$\text{Difference in service received by any two flows, tasks, etc. is less than } (D+1)(2T+c)$$

(c is a function of maximum request length and flow weights)

- See paper for proof and assumptions
Multi-Queue Fair Queueing
Scalable Implementation

- Fairness is inherently global
- MQFQ needs to maintain:
  - Smallest start tag (i.e., slowest queue) -- *Mindicator* (see paper)
  - Parallelism utilization (i.e., # of in-flight requests) -- *Token-Tree*
  - Throttling meta-data (see paper)
Scalable Implementation

Process 1 → Process 2 → ... → Process n

Core State → Socket State → Global State
Example: Parallelism Utilization

- Token-Tree

LCA(CPU0,CPU2)

sum of internal nodes = unutilized parallelism
Overview

- Motivation
- Multi-Queue Fair Queueing
- Scalable Implementation of MQFQ
- Evaluation
- Conclusion
Evaluation

● Implemented as Linux IO-Scheduler
● Benchmarked over:
  ○ NVMe SSD (up to 0.5M IOP/s)
  ○ NVMe over RDMA (up to 4M IOP/s)
● Tested applications:
  ○ Flexible IO (FIO): benchmarking tool
  ○ Aerospike: key-value store
  ○ FlashX: graph processing
● Compared against Linux’s Budget Fair Queueing (BFQ)
MQFQ is Fair

FlashX vs. FIO on SSD

Aerospike vs. FIO on SSD

Slowdown rel. to run alone

proportional slowdown

Slowdown rel. to run alone

proportional slowdown
MQFQ is Scalable
Conclusion

- We discussed:
  - Scalability vs. fairness in multi-queue I/O

- We introduced:
  - Multi-Queue Fair Queueing (MQFQ)

- We presented:
  - Scalable implementation
    - Up to 3.1 $M$ IOP/s
    - All while guaranteeing fairness