MittOS
Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface

Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi
Millisecond Matters!

Amazon: “every 100ms of latency costs 1% in sales”

Tabb Group: “broker could lose as much as $4 million in revenues per millisecond if its electronic trading platform was only 5ms behind the competition”

Google: “extra 500ms in search page generation time dropped traffic by 20%”
Millisecond Tail Latency

Parallel task #1

Parallel task #2

IO contention

Completion 30ms

10ms

30ms
Current Tail-Tolerant Mechanisms

1. Speculation
   - Most popular

2. Cloning
   - Introduces 2x workload

3. Snitching
   - Does not work when burstiness fluctuates in ms-level
My disk is busy!

Try elsewhere

MittOS @ SOSP’17

Must Wait!

Wait

Completion

Backup

20ms

No Wait?

Disk Queue

My disk is busy! Try elsewhere

Failover (no-wait)

Fast reject

Completion

10ms + network-hop
**Use-Case**

I want < 20ms latency

OS can see "everything" and tell app when it is busy

Latency = 10ms + network-hop

1. SLO = 20ms
2. ret = read(., SLO)
3. Disk Queue
4. Reject fast
5. if (ret == Reject) // failover

App

OS
MittOS

ret = read(.., < 20ms)

if (ret == Reject)
  // failover

App

OS

Disk Queue

Reject

• MittOS **Principles**
  • SLO-aware interface
  • Reject fast
  • *Transparent of busyness*
    • **PC** era: is best effort (cannot reject IOs)
    • **DC** era: Less-busy replicas available
Challenge

ret = read(., < 20ms)

App

OS

Should I reject this IO?

What is the OS queue policy?
FIFO, elevator, CFQ, etc.

What is the device type?

Prediction depends on queue policy and device type
**Contribution**  
+50 LOC

MittOS-powered MongoDB

Fast Reject Interface

**MittOS Latency Prediction**

Disk
Open-Channel SSD
OS Cache

**MittOS principle:** Support fast rejecting SLO-aware interface

vs. **state of the art:**  
hedged requests, cloning, application timeout, etc.

**Cut tail:**  
50% latency reduction above 75 percentile
Outline

- Introduction

- Design
  - Challenges
  - Solutions

- Evaluation

- Conclusion
Prediction

ret = read(…, < 20ms)

How to predict latency before submitting to the device?

Latency < SLO → Accept
Latency > SLO → Reject

How many IOs in front?
How long?
Challenge #1: Modeling Queue Policy

FIFO

- SLO < 20ms
- > 20ms Reject
- Outstanding IOs:
  - 50ms
  - 40ms
  - 30ms
  - 20ms
  - 10ms

Elevator + CFQ

- Reject / Accept depends on queue policy
- < 20ms Accept
- > 20ms Reject

Priority

High Priority

App

OS
Challenge #2: Device Type

Reject / Accept depends on device type

- FIFO
  - Single spindle
  - Reject
  - > 20ms

- Parallel channels & chips
  - Reject
  - < 20ms
  - Accept

Disk

SSD
Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed

**Elevator**

**IO Offset**

- 200
- 700
- 600
- 250

**OS prediction incorrect!**

Too many! **Reject!**

**SSTF**

Re-sort, thus fast, **Accept!**

**Scheduling algorithm**

**End of queue!**

**Reject?**

**Remap to fast chip, Accept!**
Outline

- Introduction

- Design
  - Challenges
  - Solutions

- Evaluation

- Conclusion
Reject/Latency Prediction

Reject? = \( \int (SLO, \text{ queue policy}, \text{ device type}) \)

Get from source-code. e.g. CFQ, noop

Simple type

Profiling is enough

MittCFQ

Complicated type

White-box knowledge required

MittSSD
MittCFQ

Open-sourced

Reverse engineer
Which tree/queue each IO belongs to?
How many IOs in front?

Contains user group management
Contains 3 service trees
Contains 7 different IO priorities
Contains ~4500 LOC

Disk Scheduling?
Seek latency?
(depends on seek distance)
Transfer time?
(depending on IO size)
For each interval in [100MB, 200MB, ..., 1GB] do:
for (startOffset = 0; startOffset < maxOffset; startOffset += interval) {
    for (endOffset = 0; endOffset < maxOffset; endOffset += interval) {
        for (size = 0; size < maxSize; size += sizeInterval) {
            start_ts = gettimeofday();
            seek (startOffset);
            read (endOffset, size);
            end_ts = gettimeofday();
            latency = start_ts - end_ts;
            print (endOffset - endOffset, size, latency);
        }
    }
}

IO Size
Seek Distance
Latency

Linear Regression

2 disk models
11-hour profiling

Accurate prediction

+ concurrent IO profiling

SSTF scheduling

Infer

Random seek
Random read

Collect latency

Random seek
Random read

Collect latency

MittCFQ Profiling

1 million entries
(30MB memory overhead)
For 1TB drive

scikit-learn
MittSSD

Which channel/chip?
Fast? Busy?

FTL invisible
to OS!

Pages

OS

Too complex
to model!

SSD

Fast?

Busy?

FTL

Write lat. variability

1ms
2ms
1ms
1ms
2ms
1ms
2ms

Lower MLC bits

Upper MLC bits

Invisible
dynamic GC
MittSSD

*Software-defined flash*

LightNVM

**FTL**

at host

OS knows where IOs are mapped

OS can see #outstanding IOs to every chip/channel

Accurate prediction

Open-Channel SSD

GC

OS knows page-level latencies

OS can track every single IO

OS can capture all GCs

Pages

<table>
<thead>
<tr>
<th>Pages</th>
<th>Latencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1ms</td>
<td></td>
</tr>
<tr>
<td>2ms</td>
<td></td>
</tr>
<tr>
<td>1ms</td>
<td></td>
</tr>
<tr>
<td>1ms</td>
<td></td>
</tr>
</tbody>
</table>
Reject/Latency Prediction

Reject? = \( f(\text{SLO, queue policy, device type}) \)

Reverse engineering based on source code

Profiling is enough

MittCFQ ~1800 LOC

LightNVM + Open-Channel SSD

White-box knowledge required

MittSSD ~1400 LOC
Other Solved Challenges

- Prediction overhead optimizations
  - Avoids going through every IO in the queue
  - Reduces overhead from O(n) to roughly O(1)
  - Shows < 5μs overhead for MittCFQ prediction
  - < 300ns for MittSSD prediction
- MittCache
- Prediction for OS Cache

Please refer to the paper!
Outline

- Introduction
- Design
- Evaluation
  - Tail reduction
  - Latency prediction accuracy
- Conclusion
MittCFQ-powered MongoDB

Remote YCSB client #1

Physical node #1

get()

Can failover to 3 replicas

Node #2

Client #2

get()

Node #20

Client #20

... get()

Metric:

CDF of all `get()` requests latencies (total 6 million data points)

Noisy neighbors based on EC2 data
**Baseline**

CDF of YCSB get() Latencies on 20-node MongoDB

- p99
- p95
- p90

- 90th percentile

**Next slides:** use 13ms deadline SLO for Hedged & MittCFQ

- Better!
- 13ms at p95
- > 40ms above p98
Clone

CDF of YCSB get() Latencies on 20-node MongoDB

- Tail reduction
- Worse performance < p95
Hedged Requests

Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUIZ ANDRÉ BARROSO

The Tail at Scale


(a) Sends first request

(b) Waits for 13ms timeout

(c) Sends secondary request

(d) Picks faster response
CDF of YCSB get() Latencies on 20-node MongoDB

- **Wait**
- **Secondary**
- **Little extra workload**
- **Tail reduction**

Latency (ms)
MittCFQ

CDF of YCSB get() Latencies on 20-node MongoDB

- Wait
- Failover
- Cut ms tail!
- No-wait wins!
- Fast reject
- Secondary

MittCFQ
Hedged
Tail amplified at Scale

Parallel

get() #1

get() #2

…

get() #5

Tail amplified & more improvement space

Scale Factor: 5

Up to 2x speedup above p75

Latency (ms)
Accuracy Evaluation

MittCFQ

Disk

MittSSD

Open-Channel SSD

5 real-world block-level traces

DAPPS

DTRS

EXCH

TPCC

LMBE

Metrics:

• **False positive**: IO rejected, but deadline is met
• **False negative**: Deadline violated, but IO is not rejected
Accuracy Evaluation

Only <1% inaccuracy!

Among incorrect cases:

MittCFQ: < 3ms diff
MittSSD: < 1ms diff
MittSSD

(a) Scale Factor: 1
(b) % Latency Reduction of MittSSD vs. Hedged

MittCache

(a) Scale Factor: 1
(b) % Latency Reduction of MittCache vs. Hedged

MongoDB + Filebench + Hadoop

(a) Latency CDF of MongoDB

MittCFQ
Hedged
Base

MongoDB + Filebench + Hadoop

(a) Latency CDF of MongoDB

MittCFQ
Hedged
Base

All in one

SLO 13ms
SLO 20ms
SLO 5ms

MongoDB + Filebench + Hadoop

(a) Latency CDF of MongoDB

MittCFQ
Hedged
Base

All in one

SLO 13ms
SLO 20ms
SLO 5ms

MongoDB + Filebench + Hadoop

(a) Latency CDF of MongoDB

MittCFQ
Hedged
Base

All in one

SLO 13ms
SLO 20ms
SLO 5ms

MongoDB + Filebench + Hadoop

(a) Latency CDF of MongoDB

MittCFQ
Hedged
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak

(a) Latency CDF of Riak

MittCFQ
Base

Riak
Conclusion

Do X

No wait!

Reject!

Try other students!

I'm busy!
Conclusion

MittOS-powered apps

MongoDB

riak

Fast Reject (No-wait) Interface

MittOS Latency Predictions

SSD

http://ucare.cs.uchicago.edu

Thank you! Questions?

http://ucare.cs.uchicago.edu

https://ceres.uchicago.edu