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%**”
Millissecond Tail Latency
Millissecond Tail Latency

Parallel task #1

Parallel task #2
Millisecond Tail Latency

Parallel task #1

Parallel task #2

10ms...
Millisecond Tail Latency

Parallel task #1

Parallel task #2

IO contention
Millisecond Tail Latency

Parallel task #1

10ms

Parallel task #2

30ms

IO contention
Millisecond Tail Latency

Parallel task #1

Parallel task #2

10ms

Completion 30ms
Current Tail-Tolerant Mechanisms

1. Speculation
Current Tail-Tolerant Mechanisms

1. Speculation

Wait
Current Tail-Tolerant Mechanisms

1. Speculation

Straggler!

Wait
Current Tail-Tolerant Mechanisms

1. Speculation

Straggler!

Wait

Backup
Current Tail-Tolerant Mechanisms

1. Speculation

Straggler!

Wait

Completion 20ms

Backup
Current Tail-Tolerant Mechanisms

1. Speculation

Straggler!

Wait

Backup

Completion 20ms
Current Tail-Tolerant Mechanisms

1. Speculation

2. Cloning
   - Introduces 2x workload

Wait

Completion
20ms

Backup

Straggler!
Current Tail-Tolerant Mechanisms

1. Speculation

2. Cloning
   - Introduces 2x workload

3. Snitching
   - Does not work when burstiness fluctuates in ms-level
Current Tail-Tolerant Mechanisms

1. Speculation
   - Most popular
   - Straggler!
   - Wait
   - Completion 20ms
   - Backup

2. Cloning
   - Introduces 2x workload

3. Snitching
   - Does not work when burstiness fluctuates in ms-level
Must Wait!

Backup

Wait

Completion

20ms
Must Wait!

Completion

Wait

Backup

No Wait?

Completion

20ms
Must Wait!

Completion 20ms

Backup

Wait

No Wait?

App → OS
**Must Wait!**

<table>
<thead>
<tr>
<th>Wait</th>
<th>Completion 20ms</th>
</tr>
</thead>
</table>

**No Wait?**

<table>
<thead>
<tr>
<th>App</th>
<th>OS</th>
</tr>
</thead>
</table>

Disk Queue
Must Wait!

Wait

Completion

20ms

No Wait?

App → OS

Disk Queue
My disk is busy!

Must Wait!

No Wait?

Completion
20ms
Must Wait!

Wait

Completion

20ms

Backup

No Wait?

My disk is busy! Try elsewhere

Disk Queue

App

OS
My disk is busy!
Try elsewhere

Must Wait!

No Wait?

Wait
Completion
Backup

20ms

App
OS

Disk Queue

Fast reject
Must Wait!

Backup

Completion 20ms

Wait

My disk is busy!
Try elsewhere

No Wait?

Disk Queue

App

Fast reject

Backup

Complete

os

ok
My disk is busy!
Try elsewhere

Must Wait!

Wait
Completion 20ms
Backup

No Wait?

App → OS

Fast reject

Failover \textbf{(no-wait)}

Disk Queue
**Must Wait!**

My disk is busy!  
Wait  
Completion 20ms  
Backup

**No Wait?**

My disk is busy!  
Try elsewhere  
Disk Queue  
App  
OS  
Fast reject  
Failover (no-wait)  
Completion 10ms + network-hop
My disk is busy!
Try elsewhere

Must Wait!

Wait
Completion 20ms
Backup

No Wait?

App
Fast reject
Failover (no-wait)
Completion 10ms + network-hop

Disk Queue
Use-Case

Latency = 10ms + network-hop

Fast Reject (no-wait)

App | OS
Use-Case

Latency = 10ms + network-hop

App

I want < 20ms latency

OS

Fast Reject (no-wait)
Use-Case

I want < 20ms latency

1. SLO = 20ms

Latency = 10ms + network-hop

Fast Reject (no-wait)
Use-Case

I want < 20ms latency

1. SLO = 20ms
2. ret = read(., SLO)

OS

Fast Reject
(no-wait)

Latency = 10ms + network-hop

App
Use-Case

I want < 20ms latency

1. SLO = 20ms
2. ret = read(.., SLO)
3. Fast Reject (no-wait)

Latency = 10ms + network-hop
Use-Case

I want < 20ms latency

1. SLO = 20ms
2. ret = read(., SLO)
3. Fast Reject (no-wait)

Latency = 10ms + network-hop
Use-Case

I want < 20ms latency

1. SLO = 20ms
2. ret = read(., SLO)

OS can see “everything” and tell app when it is busy

Latency = 10ms + network-hop

Fast Reject (no-wait)
Use-Case

I want <20ms latency

1. SLO = 20ms
2. \texttt{ret = read(., \texttt{SLO})}
3. \texttt{Disk Queue}
4. Reject fast

OS can see "everything" and tell app when \textbf{it is busy}

Latency = 10ms + network-hop
I want < 20ms latency

OS can see “everything” and tell app when **it is busy**

1. SLO = 20ms
2. `ret = read(..., SLO)`
3. `if (ret == Reject)`
   // failover

Latency = 10ms + network-hop

**Use-Case**
ret = read(., < 20ms)

if (ret == Reject)
    // failover

MittOS Principles
MittOS

- **MittOS Principles**
  - SLO-aware interface
  - Reject fast

```
ret = read(..., < 20ms)
if (ret == Reject)
  // failover
```
MittOS

ret = read(., < 20ms)
if ( ret == Reject)
   // failover

MittOS Principles
- SLO-aware interface
- Reject fast
- Transparent of busyness
MittOS

ret = read(., < 20ms)

if (ret == Reject)
// failover

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

ret = read(., < 20ms)

if (ret == Reject)
    // failover

MittOS Principles

• SLO-aware interface
• Reject fast
• Transparent of busyness
  • PC era: is best effort (cannot reject I/Os)
  • DC era: Less-busy replicas available
Challenge

\[
\text{ret} = \text{read}(\ldots, < 20\text{ms})
\]

App

？

OS
Challenge

ret = read(., < 20ms)

Should I reject this IO?
Challenge

\[
\text{ret} = \text{read}(\ldots, < 20\text{ms})
\]

App

OS

Should I reject this IO?

What is the OS queue policy?
Challenge

ret = read(.., < 20ms)

What is the OS queue policy?

FIFO, elevator, CFQ, etc.

Should I reject this IO?
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?
Challenge

ret = read(.., < 20ms)

App

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

OS

What is the device type?

Should I reject this IO?
Challenge

ret = read(.., < 20ms)

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

What is the device type?

Prediction depends on queue policy and device type

Should I reject this IO?
Contribution

MittOS principle: Support fast rejecting SLO-aware interface
Contribution

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

MittOS Latency Prediction

MittOS principle: Support fast rejecting SLO-aware interface
Contribution

MittOS Latency Prediction

MittOS principle: Support fast rejecting SLO-aware interface
Contribution

MittOS-powered

MittOS Latency Prediction

Disk
Open-Channel SSD
OS Cache

MittOS principle: Support fast rejecting SLO-aware interface
Contribution +50 LOC

MittOS-powered

MittOS Latency Prediction

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

MongoDB

Fast Reject Interface

Disk
Open-Channel SSD
OS Cache

MittOS Latency Prediction

MittOS principle: Support fast rejecting SLO-aware interface

Contribution

+50 LOC

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

MittOS powered

riak

Latency Prediction

+50 LOC
Contribution +50 LOC

MittOS-powered

MongoDB

Fast Reject Interface

MittOS Latency Prediction

Disk

Open-Channel

SSD

OS Cache

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

Cut tail:
50% latency reduction above 75 percentile

MittOS principle: Support fast rejecting SLO-aware interface
Outline

- Introduction
- Design
  - Challenges
  - Solutions
- Evaluation
- Conclusion
Prediction

$$\text{ret} = \text{read}(\ldots, < 20\text{ms})$$
Prediction

ret = read(., < 20ms)

How to predict latency before submitting to the device?
Prediction

ret = read(\ldots, < 20ms)

How to predict latency before submitting to the device?
Prediction

```
ret = read(., < 20ms)
```

How to predict latency before submitting to the device?
Prediction

ret = read(., < 20ms)

How to predict latency before submitting to the device?

How many IOs in front?
Prediction

ret = read(., < 20ms)

How to predict latency before submitting to the device?

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

SLO < 20ms

App

OS
Challenge #1: Modeling Queue Policy

SLO < 20ms

FIFO

Outstanding I/Os

App

OS
Challenge #1: Modeling Queue Policy

FIFO

- SLO < 20ms
- > 20ms: Reject

Outstanding IOs:
- 50ms
- 40ms
- 30ms
- 20ms
- 10ms
Challenge #1: Modeling Queue Policy

<table>
<thead>
<tr>
<th>FIFO</th>
<th>Elevator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- SLO < 20ms
- > 20ms
  - Reject
  - 50ms
  - 40ms
  - 30ms
  - 20ms
  - 10ms

- < 20ms

- Accept

- App
  - OS
Challenge #1: Modeling Queue Policy

FIFO

- Outstanding IOs:
  - 10ms
  - 20ms
  - 30ms
  - 40ms
  - 50ms

- SLO < 20ms
- > 20ms Reject

Elevator

- < 20ms Accept

Elevator + CFQ

Low Priority
- > 20ms Reject

High Priority
Challenge #1: Modeling Queue Policy

FIFO

- SLO < 20ms
- > 20ms
  - Reject

Outstanding IOs
- → 50ms
- → 40ms
- → 30ms
- → 20ms
- → 10ms

Elevator

Reject / Accept depends on queue policy

< 20ms
- Accept

> 20ms
- Reject

Elevator + CFQ

App

OS

Priority

High Priority

> 20ms
- Reject

Low Priority
Challenge #2: Device Type
Challenge #2: Device Type

- **FIFO**
  - Single spindle
  - > 20ms
  - Reject
- Disk

Single spindle
Challenge #2: Device Type

- **FIFO**
  - Single spindle
  - Parallel channels & chips
  - > 20ms
  - Reject

- **Disk**

- **SSD**
Challenge #2: Device Type

FIFO

Single spindle

Disk

> 20ms

Reject

SSD

Parallel channels & chips

< 20ms

Accept
Challenge #2: Device Type

Reject / Accept depends on device type

FIFO

Single spindle

Disk

> 20ms

Reject

Parallel channels & chips

< 20ms

Accept

FIFO

SSD

Single spindle

Parallel channels & chips

< 20ms

Accept
Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed
Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed

Elevator

OS

Too many!
Reject!

IO Offset

200
700
600
250
Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed

**Elevator**

**IO Offset**

- 200
- 700
- 600
- 250

Too many! **Reject!**

**SSTF**

Scheduling algorithm
Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed

**Elevator**

OS

**SSTF**

Scheduling algorithm

**IO Offset**

- 200
- 700
- 600
- 250

Too many! **Reject!**

Re-sort, thus fast, **Accept!**
Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed

**IO Offset**

Elevator

<table>
<thead>
<tr>
<th>Offset</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>700</td>
</tr>
<tr>
<td></td>
<td>600</td>
</tr>
<tr>
<td></td>
<td>250</td>
</tr>
</tbody>
</table>

OS

Too many! Reject!

SSTF

Re-sort, thus fast, Accept!

<table>
<thead>
<tr>
<th>Offset</th>
<th>700</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>600</td>
</tr>
<tr>
<td></td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>250</td>
</tr>
</tbody>
</table>
## Challenge #2: Device Type

**Idiosyncrasies** of devices are mostly unrevealed

### Scheduling algorithm

<table>
<thead>
<tr>
<th>IO Offset</th>
<th>(200)</th>
<th>(700)</th>
<th>(600)</th>
<th>(250)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elevator</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSTF</td>
<td>SSTF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Too many! **Reject**!

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

*End of queue!* **Reject**?

---

**OS**
Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed

<table>
<thead>
<tr>
<th>IO Offset</th>
<th>Elevator</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td></td>
</tr>
<tr>
<td>700</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
</tr>
</tbody>
</table>

OS:

- Too many! Reject!

SSTF:

- Re-sort, thus fast, Accept!

End of queue! Reject?

Remap to fast chip, Accept!
Challenge #2: Device Type

Idiosyncrasies of devices are mostly unrevealed

**IO Offset**

- Elevator
  - 200
  - 700
  - 600
  - 250

- SSTF
  - 700
  - 600
  - 200
  - 250

OS prediction incorrect!

End of queue! Reject?

Too many! Reject!

Re-sort, thus fast, Accept!

Remap to fast chip, Accept!

Scheduling algorithm

OS
Outline

- Introduction

- Design
  - Challenges
  - Solutions

- Evaluation

- Conclusion
Reject/Latency Prediction

Reject? $= \int ( $
Reject/Latency Prediction

\[ \text{Reject?} = f( \text{SLO}, ) \]
Reject/Latency Prediction

\[ \text{Reject?} = f( \text{SLO}, \text{queue policy} ) \]
Reject/Latency Prediction

\[ \text{Reject?} = f( \text{SLO, queue policy, device type} ) \]
Reject/Latency Prediction

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

Get from source-code.
e.g. CFQ, noop
Reject/Latency Prediction

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

- Get from source-code. e.g. CFQ, noop
- Simple type
- Profiling is enough
Reject/Latency Prediction

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

- Get from source-code. e.g. CFQ, noop
- Simple type: Profiling is enough
- Complicated type: White-box knowledge required

White-box knowledge required
Reject/Latency Prediction

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

MittOS @ SOSP’17

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

Simple type
  Profiling is enough

MittCFQ

Complicated type
  White-box knowledge required

MittSSD
MittCFQ
MittCFQ

Contains user group management

CFQ  OS
MittCFQ

Contains user group management
Contains 3 service trees
MittCFQ

- Contains user group management
- Contains 3 service trees
- Contains 7 different IO priorities
MittCFQ

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

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

Open-sourced
MittCFQ

Open-sourced

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

Open-sourced

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

Black box
MittCFQ

Open-sourced

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

Disk Scheduling?

Black box
MittCFQ

Open-sourced

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

Disk Scheduling?

Seek latency?
(depends on seek distance)

Black box
MittCFQ

Open-sourced

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

Disk Scheduling?

Seek latency?
(depending 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);
        }
    }
}
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);
      }
   }
}

Random seek
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);
        }
    }
}

Random seek
Random read
For each interval in \([100\text{MB}, 200\text{MB}, \ldots, 1\text{GB}]\) do:

for (startOffset = 0; startOffset < maxOffset; startOffset += interval) {
  for (endOffset = 0; endOffset < maxOffset; endOffset += interval) {
    for (size = 0; size < maxSize; size += sizeInterval) {
      startTs = gettimeofday();
      seek (startOffset);
      read (endOffset, size);
      endTs = gettimeofday();
      latency = startTs - endTs;
      print (endOffset - endOffset, size, latency);
    }
  }
}

\textbf{Random seek}
\textbf{Random read}
\textbf{Collect latency}
For each interval in \([100\text{MB}, 200\text{MB}, \ldots, 1\text{GB}]\) do:

\[
\text{for (startOffset = 0; startOffset < maxOffset; startOffset += interval) { } for (endOffset = 0; endOffset < maxOffset; endOffset += interval) { } for (size = 0; size < maxSize; size += sizeInterval) { }
\]

\[
\text{start}_ts = \text{gettimeofday(); }
\text{seek(startOffset); }
\text{read(endOffset, size); }
\text{end}_ts = \text{gettimeofday(); }
\text{latency = start}_ts - \text{end}_ts; }
\text{print (endOffset - endOffset, size, latency); }
\]
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, size, latency);
        }
    }
}

Random seek
Random read
Collect latency

2 disk models
11-hour profiling

Linear Regression

MittCFQ Profiling
For each interval in \([100MB, 200MB, \ldots, 1GB]\) do:
for \((\text{startOffset} = 0; \text{startOffset} < \text{maxOffset}; \text{startOffset} += \text{interval})\) {
    for \((\text{endOffset} = 0; \text{endOffset} < \text{maxOffset}; \text{endOffset} += \text{interval})\) {
        for \((\text{size} = 0; \text{size} < \text{maxSize}; \text{size} += \text{sizeInterval})\) {
            \text{start\_ts} = \text{gettimeofday}();
            \text{seek} (\text{startOffset});
            \text{read} (\text{endOffset}, \text{size});
            \text{end\_ts} = \text{gettimeofday}();
            \text{latency} = \text{start\_ts} - \text{end\_ts};
            \text{print} (\text{endOffset} - \text{endOffset}, \text{size}, \text{latency});
        }
    }
}

**MittCFQ Profiling**

**Random seek**

**Random read**

**Collect latency**

**Linear Regression**

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

2 disk models
11-hour profiling

**IO Size**

**Latency**

**Seek Distance**
For each interval in $[100\text{MB}, 200\text{MB}, ..., 1\text{GB}]$ 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);
        }
    }
}

**Random seek**

**Random read**

**Collect latency**

**MittCFQ Profiling**

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

Linear Regression

2 disk models
11-hour profiling

+ concurrent IO profiling

Infer

scikit-learn
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);
    }
  }
}

**Random seek**
**Random read**

**Collect latency**

**2 disk models**
**11-hour profiling**

+ concurrent IO profiling

**Infer**

**SSTF scheduling**

**MittCFQ Profiling**

**MittOS @ SOSP'17**
For each interval in \([100\, \text{MB}, 200\, \text{MB}, ..., 1\, \text{GB}]\) do:

for (\text{startOffset} = 0; \text{startOffset} < \text{maxOffset}; \text{startOffset} += \text{interval}) {
    for (\text{endOffset} = 0; \text{endOffset} < \text{maxOffset}; \text{endOffset} += \text{interval}) {
        for (\text{size} = 0; \text{size} < \text{maxSize}; \text{size} += \text{sizeInterval}){
            \text{start} \_\text{ts} = \text{gettimeofday}();
            \text{seek} (\text{startOffset});
            \text{read} (\text{endOffset}, \text{size});
            \text{end} \_\text{ts} = \text{gettimeofday}();
            \text{latency} = \text{start} \_\text{ts} - \text{end} \_\text{ts};
            \text{print} (\text{endOffset} - \text{endOffset}, \text{size}, \text{latency});
        }
    }
}

\text{Random seek}
\text{Random read}

\text{Collect latency}

\text{IO Size}
\text{Latency}
\text{Seek Distance}

2 disk models
11-hour profiling

\text{Linear Regression}
+ concurrent IO profiling

\text{Accurate prediction}

\text{SSTF}
\text{scheduling}

\text{Infer}

\text{scikit-learn}

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

Which channel/chip?
Fast? Busy?

Fast?

Busy?
MittSSD

OS

FTL invisible to OS!

SSD

Fast?

Busy?
MittSSD

FTL invisible to OS!

Invisible dynamic GC
MittSSD

OS

FTL

FTL invisible to OS!

Invisible dynamic GC

Write lat. variability

1ms
1ms
1ms

Lower MLC bits
MittSSD

FTL invisible to OS!

Invisible dynamic GC

Write lat. variability

FTL

OS

SSD

Pages

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

Lower MLC bits

Upper MLC bits
MittSSD

Too complex to model!

FTL invisible to OS!

Invisible dynamic GC

Write lat. variability

Pages

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

Lower MLC bits

Upper MLC bits

Fast?

Busy?
MittSSD

Software-defined flash

SSD

OS
MittSSD

Software-defined flash

LightNVM

FTL at host

Open-Channel SSD

OS
MittSSD

Software-defined flash

LightNVM

Open-Channel SSD

FTL at host

OS knows where IOs are mapped

OS
MittSSD

Software-defined flash

LightNVM

FTL at host

OS knows where IOs are mapped

Open-Channel SSD

OS can track every single IO
MittSSD

Software-defined flash

LightNVM

OS knows where IOs are mapped

FTL at host

OS can track every single IO

GC

OS can capture all GCs

Open-Channel SSD
MittSSD

Software-defined flash

LightNVM

OS knows where IOs are mapped

FTL at host

OS knows page-level latencies

Open-Channel SSD

OS can track every single IO

OS can capture all GCs

Pages

1ms
2ms
1ms
1ms
MittSSD

Software-defined flash

LightNVM

FTL at host

OS knows where IOs are mapped

OS can see #outstanding IOs to every chip/channel

Open-Channel SSD

GC

OS can track every single IO

OS can capture all GCs

OS knows page-level latencies

Pages

1ms

2ms

1ms

1ms
MittSSD

Software-defined flash

LightNVM

OS knows where
IOs are mapped

FTL
at host

GC

OS can see
#outstanding IOs to
every chip/channel

Accurate
prediction

OS knows
page-level
latencies

OS can track
every single IO

OS can capture
all GCs

Open-Channel
SSD

1ms
2ms
1ms
1ms

Pages
Reject/Latency Prediction

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

- **Simple type**
  - Profiling is enough
- **Complicated type**
  - White-box knowledge required

Reverse engineering based on source code
Reject/Latency Prediction

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

Reverse engineering based on source code

Simple type
Profiling is enough

Complicated type
White-box knowledge required

MittCFQ ~ 1800 LOC
Reject/Latency Prediction

Reject? = f(SLO, queue policy, device type)

Reverse engineering based on source code

Simple type
Profiling is enough

MittCFQ ~1800 LOC

Complicated type
White-box knowledge required

MittSSD ~1400 LOC

LightNVM + Open-Channel SSD
Other Solved Challenges
Other Solved Challenges

• Prediction overhead optimizations
Other Solved Challenges

• Prediction overhead optimizations
  • Avoids going through every IO in the queue
Other Solved Challenges

• Prediction overhead optimizations
  • Avoids going through every IO in the queue
  • Reduces overhead from $O(n)$ to roughly $O(1)$
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
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
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
Outline

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

Physical node #1  Node #2  ...  Node #20
MittCFQ-powered MongoDB

Remote YCSB client #1

get()

Physical node #1

Client #2

get()

Node #2

...

Client #20

get()

Node #20
MittCFQ-powered MongoDB

Remote YCSB client #1

Physical node #1

get()

get()

Node #2

Node #20

Client #2

Client #20

…

get()

get()

…

Noisy neighbors based on EC2 data
MittCFQ-powered MongoDB

- Remote YCSB client #1
- Can failover 3 replicas
- Noisy neighbors based on EC2 data
MittCFQ-powered MongoDB

Remote YCSB client #1
Physical node #1

Can failover 3 replicas

Node #2

Client #2

... get()

Node #20

Client #20

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

MongoDB on EC2 (Baseline)
Baseline

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

90th percentile

Latency (ms)
Baseline

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

CDF (percentile)

p99
p95
p90

90th percentile

Latency (ms)

MongoDB on EC2 (Baseline)
Baseline

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

90\textsuperscript{th} percentile

Latency (ms)

MongoDB on EC2 (Baseline)

Better!
Baseline

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

> 40ms above p98
Baseline

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

- p99 > 40ms above p98
- p95 13ms at p95
- 90th percentile
Baseline

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

- p99
- p95
- p90

90th percentile

Latency (ms)

MongoDB on EC2 (Baseline)

13ms at p95

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

> 40ms above p98
Clone

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

- p99
- p95
- p90

Latency (ms)

Clone
"Baseline"
Clone

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

Tail reduction

Clone  "Baseline"
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

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
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

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
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

- p99
- p95
- p90

Latency (ms)

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

Latency (ms)

Little extra workload

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

- Tail reduction
- Little extra workload

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

- p99: Wait
- p95: Secondary
- p90: Tail reduction
- Little extra workload

Latency (ms)
MittCFQ

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

Latency (ms)
MittCFQ

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

Cut ms tail!
MittCFQ

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

- p99 Wait
- p95 Secondary
- p90 Cut ms tail!

MittCFQ
Hedged
MittCFQ

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

- **Wait**
- **Failover**
- **Secondary**
- **Cut ms tail!**

Fast reject
MittCFQ

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

- **Wait**
- **Failover**
- **Fast reject**
- **Secondary**
- **Cut ms tail!**

**No-wait wins!**
Tail amplified at Scale
Tail amplified at Scale

Parallel

get() #1

get() #2

get() #5

...
Tail amplified at Scale

Parallel

get() #1

get() #2

...
Tail amplified at Scale

Parallel
get() #1
get() #2
get() #3
get() #4
get() #5

Tail amplified & more improvement space
Tail amplified at Scale

Parallel
get() #1
get() #2
...
get() #S

Tail amplified & more improvement space

Scale Factor: 5

Latency (ms)

MittCFQ
Hedged
Base
Tail amplified at Scale

Parallel
get() #1
get() #2
get() #5

Tail amplified & more improvement space

Scale Factor: 5

MittCFQ
Hedged
Base

Up to 2x speedup above p75

Latency (ms)

p50
p75
p90
p95
Accuracy Evaluation
Accuracy Evaluation

MittCFQ

Disk

MittSSD

Open-Channel SSD

MittSSD
Accuracy Evaluation

MittCFQ

Disk

MittSSD

Open-Channel SSD

5 real-world block-level traces

DAPPS
DTRS
TPCC
EXCH
LMBE
Accuracy Evaluation

MittCFQ

Disk

MittSSD

Open-Channel SSD

Metrics:

- False positive: IO rejected, but deadline is met

5 real-world block-level traces

DAPPS
DTRS
TPCC
EXCH
LMBE
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

(b) MittSSD (a) MittCFQ

Accuracy Evaluation
Accuracy Evaluation

False positive

False negative

(b) MittSSD (a) MittCFQ

inaccuracy (%)

DAPPS  DTRS  EXCH  LMBE  TPCC

Accuracy Evaluation
Accuracy Evaluation

(b) MittSSD (a) MittCFQ inaccuracy inaccuracy (\%) (\%)

False positive

False negative

DAPPS  DTRS  EXCH  LMBE  TPCC

0.8  0.6  0.4  0.2

1  0.8  0.6  0.4  0.2
Accuracy Evaluation

(b) MittSSD (a) MittCFQ inaccuracy inaccuracy (%) (b) MittSSD (a) MittCFQ inaccuracy inaccuracy (%)
Accuracy Evaluation

Only <1% inaccuracy!

False positive

False negative

(b) MittSSD (a) MittCFQ inaccuracy (%)
Accuracy Evaluation

Only <1% inaccuracy!

Among incorrect cases:

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

(b) MittSSD (a) MittCFQ
inaccuracy (\%)

False positive

False negative

DAPPS  DTRS  EXCH  LMBE  TPCC
MittSSD

(a) Scale Factor: 1

(b) % Latency Reduction of MittSSD vs. Hedged

MittCache

(a) Scale Factor: 1

(b) % Latency Reduction of MittCache vs. Hedged
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
MittSSD

(a) Scale Factor: 1

(b) % Latency Reduction of MittSSD vs. Hedged

Latency (ms)

MittSSD
Hedged
Base

MittCache

(a) Scale Factor: 1

(b) % Latency Reduction of MittCache vs. Hedged

Latency (ms)

MittCache
Hedged
Base

MongoDB + Filebench + Hadoop

(a) Latency CDF of MongoDB

Latency (ms)

MittCFQ
Hedged
Base

All in one

SLO
13ms

SLO
20ms

SLO
5ms

MittCache

MittCFQ
MittSSD
**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

**All in one**

SLO 13ms  SLO 20ms  SLO 5ms

**Riak**

(a) Latency CDF of Riak

MittCache

MittCFQ

MittSSD
Conclusion

Do X
Conclusion

Do X

I’m busy!
Conclusion

Do X

Reject!

I’m busy!
Conclusion

Do X

Reject!

I’m busy!

Try other students!
**Conclusion**

- **Do X**
  - Reject!
  - I’m busy!

- **No wait!**
  - Try other students!
Conclusion

Fast Reject (No-wait) Interface

MittOS
Conclusion

MittOS

Latency Predictions

Fast Reject (No-wait) Interface
Conclusion

MittOS-powered apps

Fast Reject (No-wait) Interface

MittOS Latency Predictions

MongoDB riak

SSD
Conclusion

MittOS-powered apps

MongoDB

Fast Reject (No-wait) Interface

MittOS Latency Predictions

Cuts ms tail!

State of art

Latency CDF
Conclusion

MittOS-powered apps

MongoDB

riak

Fast Reject (No-wait) Interface

Latency Predictions

MittOS

CDF

Latency

Cuts ms tail!

Thank you! Questions?

http://ucare.cs.uchicago.edu

https://ceres.uchicago.edu