Centralized Scheduling of Mixed Workloads at μs Granularity

Kostis Kaffes, Timothy Chong, Jack Humphries, Adam Belay, David Mazieres, Christos Kozyrakis
Problem Overview

The goal is to develop a system that allows the scheduling of mixed latency-critical workloads on a single server while satisfying service level objectives (SLO) and utilizing resources efficiently.
Problem Overview

The goal is to develop a system that allows the scheduling of mixed latency-critical workloads on a single server while satisfying service level objectives (SLO) and utilizing resources efficiently.

App 1 (e.g. key-value store)
SLO: 99% 200us@ 2000QPS

App 2 (e.g. search engine)
SLO: 99% 500us@ 500QPS
Problem Overview

The goal is to develop a system that allows the scheduling of mixed latency-critical workloads on a single server while satisfying service level objectives (SLO) and utilizing resources efficiently.

App 1a (e.g. KV put/get)
SLO: 99% 50us
10000QPS

App 1b (e.g. KV range query)
SLO: 99% 500us
500QPS
Why latency matters?
Why latency matters?

User-facing
Why latency matters?

User-facing

High Fan-Out
Why latency matters?
Why mixed workloads?

Tail latency is very susceptible to interference
Why mixed workloads?

Tail latency is very susceptible to interference

Workload Isolation
Why mixed workloads?

Tail latency is very susceptible to interference

Workload Isolation

Core Allocation

Core w/ App 1

Core w/ App 2
Why mixed workloads?

Tail latency is very susceptible to interference

Workload Isolation

Core Allocation

Core w/ App 1

Core w/ App 2
Why mixed workloads?

Tail latency is very susceptible to interference

Workload Isolation

(-) Cannot handle variability within apps
Why mixed workloads?

Tail latency is very susceptible to interference

Workload Isolation

(-) Cannot handle variability within apps
(-) Load changes cannot be sharp
and/or tail latency requirements must be relaxed due to reallocation
Why mixed workloads?

Tail latency is very susceptible to interference

Workload Isolation

(-) Cannot handle variability within apps
(-) Load changes cannot be sharp and/or tail latency requirements must be relaxed due to reallocation

Overprovisioning & Underutilization
Why mixed workloads?

Why mixed workloads?

Tail latency is very susceptible to interference

Workload Isolation

(-) Cannot handle variability within apps
(-) Load changes cannot be sharp and/or tail latency requirements must be relaxed due to reallocation
(-) Does not support a large number of different app types
Why mixed workloads?

Tail latency is very susceptible to interference

Workload Isolation

(-) Cannot handle variability within apps
(-) Load changes cannot be sharp and/or tail latency requirements must be relaxed due to reallocation
(-) Does not support a large number of different app types

Need to schedule mixed workloads on the same set of CPUs!
System Design Overview
System Design Overview

1. Centralized Queue
System Design Overview

1. Centralized Queue
2. Scheduling Policies
System Design Overview

1. Centralized Queue
2. Scheduling Policies
3. Preemption Mechanisms
System Design Overview

1. Centralized Queue
2. Scheduling Policies
3. Preemption Mechanisms

Previous Work
1. Linux
2. IX
3. ZygOS
Centralized Queue
Some Queuing Theory

A/S/n

A: Inter-arrival distribution

S: Service time distribution

n: Number of workers (CPUs in our case)
Some Queuing Theory

A/S/n

A: Inter-arrival distribution
S: Service time distribution
n: Number of workers (CPUs in our case)

M/M/1

M: Exponential

\( \lambda \)

\( \mu \)
Some Queuing Theory

A/S/n

A: Inter-arrival distribution

S: Service time distribution

n: Number of workers (CPUs in our case)

A/S/n

M/M/1

G/G/1

M: Exponential

G: Any Distribution
Local vs Centralized Queues

N x G/G/1
(local queues)
Local vs Centralized Queues

N x G/G/1
(local queues)

G/G/N
(centralized queue)
Local vs Centralized Queues

\[ N \times \text{G/G/1} \] (local queues)

\[ \text{G/G/N} \] (centralized queue)

Work conserving
Local vs Centralized Queues

N x G/G/1  
(local queues)

G/G/N  
(centralized queue)

(+): Work conserving  
(+): Better latency
Local vs Centralized Queues

N x G/G/1
(local queues)

G/G/N
(centralized queue)

(+): Work conserving
(+): Better latency
(+): No shared resources

N Queues

N CPUs

1 Queue

N CPUs

N CPUs
Local vs Centralized Queues

N x G/G/1
(local queues)

G/G/N
(centralized queue)

(+ Work conserving
(+ Better latency
(+ No shared resources
(+ Work stealing

N CPU

N Queues

1 Queue

N CPUs

N CPUs
Local vs Centralized Queues

- **Local Queues**
  - $N \times G/G/1$
  - (local queues)
  - (+) Central view of the system
  - (+) More informed scheduling decisions

- **Centralized Queue**
  - $G/G/N$
  - (centralized queue)
  - (+) Work conserving
  - (+) Better latency
  - (+) No shared resources
  - (+) Work stealing
  - (+) Central view of the system
  - (+) More informed scheduling decisions

N CPUs

1 Queue

N CPUs
Local vs Centralized Queues

1 Queue
N CPUs

N Queues
N CPUs

N x G/G/1
(local queues)

(+@) Work conserving
(+@) Better latency
(+@) No shared resources
(+@) Work stealing
(+@) Central view of the system
(+@) More informed scheduling decisions

G/G/N
(centralized queue)

1 Queue
N CPUs
Existing Systems

- **Traditional Linux:**
  - Provides flexibility as we can use either local \((N \times G/G/1)\) or centralized \((G/G/N)\) queue architecture
  - (-) Incurs high overhead for \(\mu s\)-level tasks

- **IX (OSDI 2014):**
  - Uses multi-queue NICs and RSS to scale the networking stack across multiple cores of the system
  - (+) Suitable for \(\mu s\)-level tasks
  - (-) Statically partitions connections adopting the \(N \times G/G/1\) architecture

- **ZygOS (SOSP 2017):**
  - Approximates \(G/G/N\) by implementing work stealing on top of IX
  - (+) Suitable for \(\mu s\)-level tasks
  - (+) Work conserving scheduler
  - (-) Not suitable for implementing efficient scheduling policies due to its distributed design
Efficient Centralized Queue - Our Approach

1 Queue

N CPUs

G/G/N
Efficient Centralized Queue - Our Approach

1. Locking

G/G/N
Efficient Centralized Queue - Our Approach

1. Locking
2. Delegation

G/G/N - 1
Efficient Centralized Queue - Our Approach

1. Locking
2. Delegation

Early Results
- < 400 ns latency
- ~ 10 MOPS

G/G/N - 1

Dispatcher CPU

N - 1 Worker CPUs
Scheduling Policies
Selecting a scheduling policy

- The optimal scheduling policy in terms of tail latency depends on the distribution of the service time of the requests of a workload, $A/S/n$
Selecting a scheduling policy

- The optimal scheduling policy in terms of tail latency depends on the distribution of the service time of the requests of a workload, $A/S/n$
- **Heavy-Tailed vs Light-Tailed**
Distribution Types

- Heavy-Tailed
  - Regularly-Varying
    - Pareto
  - Weibull
- Light-Tailed
  - Exponential
- Normal
Light-tailed workloads

- The best policy for light-tailed distributions is **First Come First Served**
- The tail latency is mostly determined by queuing time
Heavy-tailed workloads

- FCFS does not work for heavy-tailed distributions
- A long running request can monopolize a processor
- It has been mathematically proven that the best policy for heavy-tails is Processor Sharing. [Wierman and Zwart]
Heavy-tailed workloads

- FCFS does not work for heavy-tailed distributions
- A long running request can monopolize a processor
- It has been mathematically proven that the best policy for heavy-tails is Processor Sharing. [Wierman and Zwart]

Need for preemption
Existing Systems

- **Traditional Linux:**
  - Provides different scheduling options (CFS, RT, FCFS)
  - None of them is particularly good for mixing latency-critical workloads
  - Incurs high overhead for μs-level tasks

- **Current dataplane OS, e.g. IX (OSDI 2014):**
  - Only FCFS scheduling, no preemption
  - Suitable for homogeneous μs-level tasks
  - Cannot handle heavy-tailed and mixed workloads

- **ZygOS (SOSP 2017):**
  - Only FCFS scheduling, no preemption for scheduling
  - Suitable for homogeneous μs-level tasks
  - Cannot handle heavy-tailed and mixed workloads
Mixing workloads - Linux CFS

- Masstree: Key-value store -- light-tailed
- Xapian: Search Engine -- heavy-tailed
Mixing workloads - IX/ZygOS (FCFS)

- 64 byte UDP request - 64 byte reply
- Mixed 20/200us processing
- 50%/50% load
Mixing workloads - IX/ZygOS (FCFS)

- 64 byte UDP request - 64 byte reply
- Mixed 20/200us processing
- 50%/50% load
Mixing workloads - IX/ZygOS (FCFS)

- 64 byte UDP request - 64 byte reply
- Mixed 20/200us processing
- 50%/50% load
Mixing workloads - IX/ZygOS (FCFS)

- 64 byte UDP request - 64 byte reply
- Mixed 20/200us processing
- 50%/50% load

Opportunity for improvement!
Mixing workloads - Our approach

- Centralized per application queues
- Goal → Each app stays within a target latency
Mixing workloads - Our approach

1) Which app to select from?
Mixing workloads - Our approach

1) Which app to select from?
2) How to set the time slice?
Mixing workloads - Our approach

1) Which app to select from?
2) How to set the time slice?
3) Where to enqueue an unfinished request?
Scheduling Simulator

- System-aware, highly configurable scheduling simulator
- Written in Python to ensure modularity and easy extensibility
- Based on SimPy, a discrete event simulator (DES)
Scheduling Simulator

- System-aware, highly configurable scheduling simulator
- Written in Python to ensure modularity and easy extensibility
- Based on SimPy, a discrete event simulator (DES)

User-defined parameters:
- Host Configuration and Queue Types
- Number of host cores
- Scheduler (FCFS, PS, custom)
- Dequeuing cost
- Preemption cost
- Time Slice (for PS)
- Multiple applications; per app:
  - Request execution time distribution (Bernoulli, LogNormal, Exponential, Pareto, Normal)
  - Request inter-arrival time distribution (Exponential, LogNormal)
  - Load
  - Target tail latency (SLO)

Metrics:
- Per application and overall statistics
- Latency at any percentile
- Throughput (requests/sec)
- % of requests violating SLO
Scheduling Simulator

- System-aware, highly configurable scheduling simulator
- Written in Python to ensure modularity and easy extensibility
- Based on SimPy, a discrete event simulator (DES)

**User-defined parameters:**
- Host Configuration and Queue Types
- Number of host cores
- Scheduler (FCFS, PS, custom)
- Dequeuing cost
- Preemption cost
- Time Slice (for PS)
- Multiple applications; per app:
  - Request execution time distribution (Bernoulli, LogNormal, Exponential, Pareto, Normal)
  - Request inter-arrival time distribution (Exponential, LogNormal)
  - Load
  - Target tail latency (SLO)

**Metrics:**
- Per application and overall statistics
- Latency at any percentile
- Throughput (requests/sec)
- % of requests violating SLO

Let us know if you are interested in using it!
1) Which queue to select from?

App 1

App 2

Dispatcher

Worker CPUs
1) Which queue to select from?

**Idea 1:** The queue with the largest backlog (remaining running time) normalized by the target latency of the requests of the queue.
1) Which queue to select from?

Idea 1: The queue with the largest backlog (remaining running time) normalized by the target latency of the requests of the queue
(-) Assumes knowledge of the exact execution time of all requests in the system
(-) Does not take aging into account
1) Which queue to select from?

Idea 2: Check the first packet of each queue and select the one with the highest normalized waiting + remaining execution time.
1) Which queue to select from?

Idea 2: Check the first packet of each queue and select the one with the highest normalized waiting + remaining execution time

(-) Still assumes knowledge of the exact execution time of all requests in the system
1) Which queue to select from?

Idea 3: Check only the normalized waiting time of first packet of each queue.
App 1

Dispatcher

App 2

Worker CPUs
1) Which queue to select from? -- Evaluation

- Exponential + Lognormal workloads
- Same means
- Individual load = Cumulative load / 2.0
1) Which queue to select from? -- Evaluation

- Exponential + Lognormal workloads
- Same means
- Individual load = Cumulative load / 2.0
1) Which queue to select from? -- Evaluation

- Exponential + Lognormal workloads
- Same means
- Individual load = Cumulative load / 2.0
1) Which queue to select from? -- Evaluation

- Exponential + Lognormal workloads
- Same means
- Individual load = Cumulative load / 2.0
1) Which queue to select from? -- Evaluation

- Exponential + Lognormal workloads
- Same means
- Individual load = Cumulative load / 2.0
1) Which queue to select from? -- Conclusion

- Exponential + Lognormal workloads
- Same means
- Individual load = Cumulative load / 2.0

Knowing only the age of requests is enough to make near optimal scheduling decisions!
2) How to set the time slice?

Idea: Set the time slice as some fixed percentage of the SLO of the shorter app
2) How to set the time slice? -- Evaluation

- Exponential + Lognormal workloads
- Different means
- Individual load = Cumulative load / 2.0
2) How to set the time slice? -- Evaluation

- Exponential + Lognormal workloads
- Different means
- Individual load = Cumulative load / 2.0

Exponential Workload ($\mu=2$)

Lognormal Workload ($\mu=20$, $\sigma=200$)
2) How to set the time slice? -- Evaluation

- Exponential + Lognormal workloads
- Different means
- Individual load = Cumulative load / 2.0
2) How to set the time slice? -- Evaluation

- Exponential + Lognormal workloads
- Different means
- Individual load = Cumulative load / 2.0

**Exponential Workload (μ=2)**

- No Preemption
- Time Slice = 2 SLO
- Time Slice = SLO

**Lognormal Workload (μ=20, σ=200)**

- No Preemption
- Time Slice = 2 SLO
- Time Slice = SLO
2) How to set the time slice? -- Evaluation

- Exponential + Lognormal workloads
- Different means
- Individual load = Cumulative load / 2.0

Exponential Workload ($\mu=2$)

Lognormal Workload ($\mu=20$, $\sigma=200$)
2) How to set the time slice? -- Evaluation

- Exponential + Lognormal workloads
- Different means
- Individual load = Cumulative load / 2.0
2) How to set the time slice? -- Evaluation

- Exponential + Lognormal workloads
- Different means
- Individual load = Cumulative load / 2.0
2) How to set the time slice? -- Conclusion

- Exponential + Lognormal workloads
- Different means
- Individual load = Cumulative load / 2.0

- Preemption is necessary
- Setting the time slice equal up to ~50% of the SLO of the shortest app is sufficient to achieve good performance
3) Where to enqueue an unfinished request?

App 1

App 2

Idea: Approximate the optimal policy for each app type

Dispatcher

Worker CPUs
3) Where to enqueue an unfinished request?

**Idea:** Approximate the optimal policy for each app type

- For **light-tailed** apps enqueue to the **front** of the queue to approximate **FCFS**
3) Where to enqueue an unfinished request?

**Idea:** Approximate the optimal policy for each app type
- For **light-tailed** apps enqueue to the **front** of the queue to approximate **FCFS**
- For **heavy-tailed** apps enqueue to the **back** of the queue to approximate **PS**
Preemption Mechanisms
## Interrupt Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Median Latency</th>
<th>99% Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signals</td>
<td>3-4us</td>
<td>10us</td>
</tr>
<tr>
<td>Local Timer Interrupts</td>
<td>~640ns</td>
<td>&lt;1us</td>
</tr>
<tr>
<td>“Posted” Interrupts</td>
<td>~1us</td>
<td>?</td>
</tr>
</tbody>
</table>
Preemption Cost

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load

Exponential ($\mu = 2$) -- Preemption Cost

Lognormal ($\mu = 20$, $\sigma = 200$) -- Preemption Cost
**Preemption Cost**

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load

![Graph of Preemption Cost](image)

- **Exponential ($\mu=2$) -- Preemption Cost**
  - 99% Latency vs Cumulative Load
  - No Preemption
  - 0.5 Time Slice

- **Lognormal ($\mu=20$, $\sigma=200$) -- Preemption Cost**
  - 99% Latency vs Cumulative Load
  - No Preemption
  - 0.5 Time Slice
Preemption Cost

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load
Preemption Cost

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load
Preemption Cost

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load

**Exponential ($\mu=2$) -- Preemption Cost**

**Lognormal ($\mu=20$, $\sigma=200$) -- Preemption Cost**
Preemption Cost

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load
Preemption Cost

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load
Preemption Cost

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load
Preemption Cost

- Exponential ($\mu = 2$), Lognormal ($\mu = 20$)
- 8 CPUs
- Time Slice = 50% SLO
- 50%/50% load

Preemption cost of up to 10% of the time slice is acceptable
Which mechanism to use?

<table>
<thead>
<tr>
<th>Example App</th>
<th>SLO</th>
<th>Time Slice</th>
<th>Preemption Cost</th>
<th>Preemption Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silo</td>
<td>200us</td>
<td>100us</td>
<td>10us</td>
<td>Any</td>
</tr>
<tr>
<td>Memcached</td>
<td>100us</td>
<td>50us</td>
<td>5us</td>
<td>Posted/Local Interrupts</td>
</tr>
<tr>
<td>RAMCloud</td>
<td>20us (?)</td>
<td>10us</td>
<td>1us</td>
<td>Local Interrupts</td>
</tr>
</tbody>
</table>
Full Picture

App 1

App 2

Dispatcher

Worker CPUs
1) Delegate central queue accesses to dispatcher
1) Delegate central queue accesses to dispatcher
2) **Make latency-aware scheduling decisions at the dispatcher**
1) Delegate central queue accesses to dispatcher
2) Make latency-aware scheduling decisions at the dispatcher
3) **Preempt running tasks**
1) Delegate central queue accesses to dispatcher
2) Make latency-aware scheduling decisions at the dispatcher
3) Preempt running tasks

Doing everything at μs scale!
Status

Building on top of IX
→ Scheduling Policies
→ Centralized Queue
→ Preemption
Open Questions
Open Questions

- What is the unit of scheduling?
Open Questions

- What is the unit of scheduling?
  - Needs to be preemptible → Lightweight Context, fiber
Open Questions

● What is the unit of scheduling?
  ○ Needs to be preemptible → Lightweight Context, fiber

● When/where is the network processing taking place?
Open Questions

- What is the unit of scheduling?
  - Needs to be preemptible → Lightweight Context, fiber
- When/where is the network processing taking place?
  - Need to avoid HoL blocking within TCP connections
  - Need to have a notion of measuring the age of each request
Open Questions

- What is the unit of scheduling?
  - Needs to be preemptible → Lightweight Context, fiber

- When/where is the network processing taking place?
  - Need to avoid HoL blocking within TCP connections
  - Need to have a notion of measuring the age of each request

Separation of network packet processing and application-level request execution
Open Questions

- What is the unit of scheduling?
  - Needs to be preemptible → Lightweight Context, fiber

- When/where is the network processing taking place?
  - **Option 1:** Dispatcher Core: (+) Locality (-) Scalability Issues (-) Might delay scheduling

Open Questions

- What is the unit of scheduling?
  - Needs to be preemptible → Lightweight Context, fiber

- When/where is the network processing taking place?
  - **Option 2:** Add separate queue with networking work
Open Questions

- What is the unit of scheduling?
  - Needs to be preemptible → Lightweight Context, fiber

- When/where is the network processing taking place?
  - **Option 2:** Add separate queue with networking work
Open Questions

- What is the unit of scheduling?
  - Needs to be preemptible \(\rightarrow\) Lightweight Context, fiber

- When/where is the network processing taking place?
  - **Option 2:** Add separate queue with networking work
Open Questions

- What is the unit of scheduling?
  - Needs to be preemptible \(\rightarrow\) Lightweight Context, fiber
- When/where is the network processing taking place?
  - Dispatcher Core vs Separate Queue
- Can we scale to large app/request types counts?
Future directions

● Learn application distribution on the fly
● Use machine learning to determine all the “arbitrary”-set parameters, e.g. time slice (maybe even for scheduling)
Thank you!

Questions?