# Catena: A Scheduling System for Microsecond-Level Microservice DAGs

†Ryan Kosta (rkosta@ucsd.edu) Amy Ousterhout (aousterhout@ucsd.edu) Yiying Zhang (yiying@ucsd.edu)
UC San Diego †Student

Modern web services such as Twitter and Netflix are composed of many small tasks, known as microservices. This approach enables each microservice to scale independently and allows for parallel development of components. A given end-to-end service is represented as a DAG, where each microservice is a vertex, and calls between microservices are edges.

The service time of microservices is much shorter than that of traditional services, within the magnitude of microseconds. At this timescale, preemption overhead is too costly relative to task length. As a result, tasks suffer dispersion-based head-of-line blocking, where a long task blocks many shorter tasks from running. Tasks are so short, hardware factors such as cache misses cause runtime unpredictability. Since most microservices call other microservices, a burst in a service causes fluctuations in demand for subsequent services in the DAG.

Due to the frequent fluctuations in service demand, it is common to allocate resources based on the peak demand, resulting in resource wastage during periods of lower demand. Alternatively, resources can be allocated for average demand, causing overloads during bursts. However, a single overloaded microservice often degrades the performance of the entire application. As DAGs of interdependent microservices become longer and more intricate, the probability of any service in the DAG experiencing overload increases. This poses the research problem of how to minimize overload across an entire application without over-allocating resources.

Overload control techniques [1] regulate the ingress rate of incoming requests. This ensures a service's execution rate is not degraded due to an excess of requests, and regulates throughput allowing for tight resource allocations. However, these systems are incompatible with fine-grained resource reallocation, leading to lower utilization in multi-tenant scenarios.

Scheduling techniques [2] quickly adjust resource allocations to respond to load variance. They collocate multiple services and quickly adjust resource allocations to respond to load demands of the different services, avoiding failure of requests. However, since they frequently adjust core allocations, their processing rate heavily fluctuates, intensifying load spikes for later microservices. In long DAGs, this causes cascading failure.

We propose *Catena*, a new RPC system that allows colocation of latency-critical microservices whilst achieving low tail latency and high server utilization. We achieve this by creating a credit system that supports microsecond scheduling, and using knowledge of microservice dependencies to resolve resource contention when it occurs. Our credit system allocates credits per service request, controlling the request ingress rate based on the number of outstanding credits. We allocate credits using a dynamic credit multiplier which is multiplied with the number of cores allocated to a service to decide how many outstanding credits are distributed.

Catena's RPC system is built into the network layer of Shenango [2] to explicitly co-optimize with the existing scheduling system and utilize kernel-bypass networking. Catena uses a tiered approach to achieve both high throughput and low latency when vertical autoscaling compute resources.

In the first tier, we measure queuing delay and adjust the credit multiplier via Additive Increase Multiplicative Decrease if delay is not within target latency range, similar to Breakwater [1]. Since the request queue can handle a small excess of requests, minor credit mispredictions are acceptable. Therefore, we use more aggressive parameters to achieve faster convergence.

In the second tier, we measure CPU utilization and adjust core allocations via Additive Increase Additive Decrease if CPU utilization is not within the target range. We use less aggressive parameters to avoid core thrashing and starvation between services.

Allocating cores in proportion to CPU utilization achieves a much more stable request throughput then explicit delay-based core allocation, quelling burstiness that could lead to cascading failure. At the same time, the credit system quickly reacts to latency violations to ensure low tail latency.

While microsecond scheduling quells small bursts in resource demands on a given server, if co-located services burst concurrently then resource contention occurs. Traditionally, horizontal autoscaling is utilized, but deploying a new instance would take much longer then a cascading service failure would take to develop.

Catena proposes using knowledge of the DAG to solve resource contention. Since CPU allocation is dependent on latency targets, if a service cannot allocate enough CPUs, it can raise its latency target and request that less resource-constrained dependent services in the DAG(s) proportionally decrease their latency target so the end-to-end service latency remains unchanged.

## References

[1] I. Cho, A. Saeed, J. Fried, S. J. Park, M. Alizadeh, and A. Belay. Overload control for µs-scale RPCs with breakwater. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, Nov. 2020.

[2] A. Ousterhout, J. Fried, J. Behrens, A. Belay, and H. Balakrishnan. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019.

# Catena: A Scheduling System for Microsecond-Level Microservice DAGs

UC San Diego

Ryan Kosta (student),
Amy Ousterhout, Yiying Zhang

WukLab

## Modern Microservices

**Bursty Demand and Performance:**

- Demand varies heavily on the μs scale
- HOL blocking causes bursts in task completion
- Large load spikes can lead to:
  - Adverse traffic effects
  - Cascading failure

## State of the Art

**Traditional μs Scheduling:**

- Constantly adjusts core allocations causing processing rate fluctuations and traffic bursts
- With a longer DAG, the burst effect multiplies

**Traditional Overload Control:**

- Unable to respond to quick changes in resource allocations

## Goals

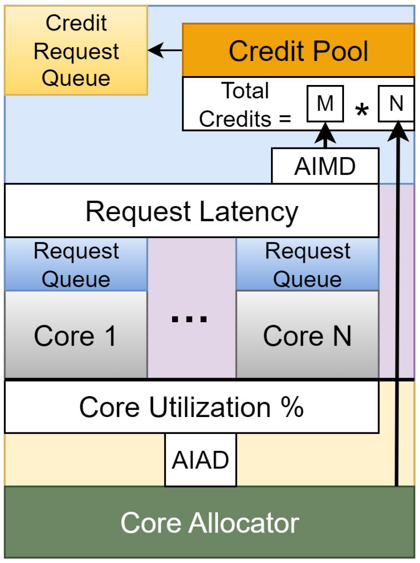
- Respond to **μs-level load variance**
- **Tightly pack many** latency-critical μs microservices
- Handle **resource contention** of microservices

## Idea: DAG-Aware Distributed Autoscaling

- Use DAG properties to dynamically adjust allocations
- Handle **μbursts** (10s μs) of resource contention
  - **Without violating SLO** guarantees
  - Too fine-grained for horizontal autoscaling

## Idea: Throughput-Aware Vertical Autoscaling

- **Multi-Tiered Optimization Approach**
- Assign credits based on multiplier of cores allocated
- **Optimize for Tail Latency**: Credit Multiplier
  - AIMD algo [1] based on measured queue delay
  - More Aggressive: Allows for faster convergence
- **Optimize for Resource Utilization**: Core allocation
  - AIAD algorithm based on measured CPU util
  - Less aggressive: avoids CPU starvation/thrashing

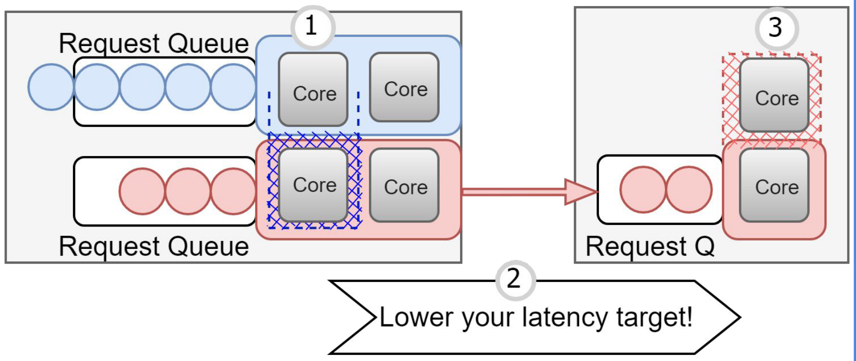


## Prototype

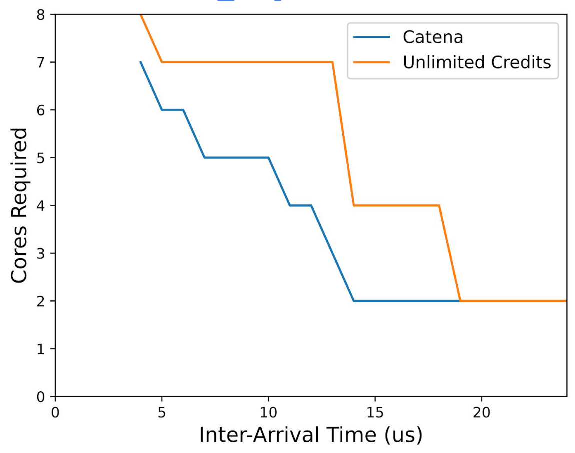
- RPC protocol built into Shenango's[2] kernel bypass networking layer
- **Throughput-Aware Vertical Autoscaling** via credit control system

## Example: DAG-Aware

1. Blue has queue buildup, **steals** red's core
2. Red is now under resource constraints, so it
   a. Increases its latency target (reducing resource requirements)
   b. **Asks** service in red DAG to decrease its latency target proportionally
3. Other service in Red DAG
   a. Decreases its latency target (increasing resource requirements)
   b. Allocates a new core



## Example: Throughput-Aware



- **Arrival Rate:** Poisson centered around mean
- **Processing Time:** Bimodal (90% 5μs, 10% 55μs)
- **Microservices:** 3 in linear DAG
- **Cores:** 2 statically allocated cores + burst cores

[1] I. Cho, A. Saeed, J. Fried, S. J. Park, M. Alizadeh, and A. Belay. Overload control for μs-scale RPCs with breakwater. In OSDI 20, Nov. 2020.

[2] A. Ousterhout, et al. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In NSDI 2019.