Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads





Amy Ousterhout

Joshua Fried

Jonathan Behrens

Adam Belay

Hari Balakrishnan

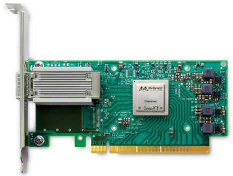
CSAIL

Trend #1: Faster Networks

2008
Latency: ~100 µs
Throughput: 1 Gbits/s

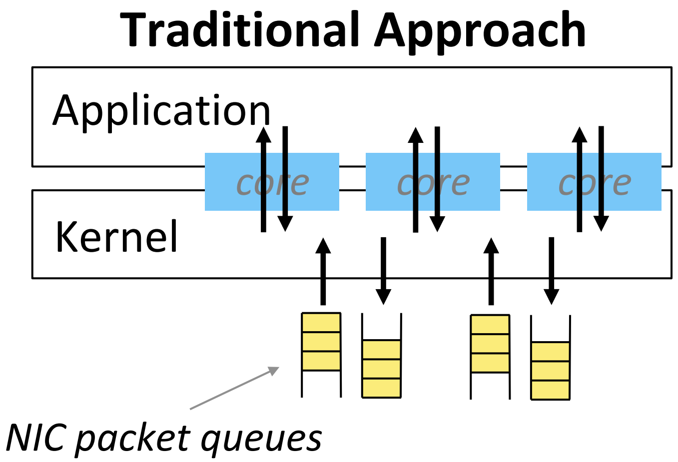
20x
100x

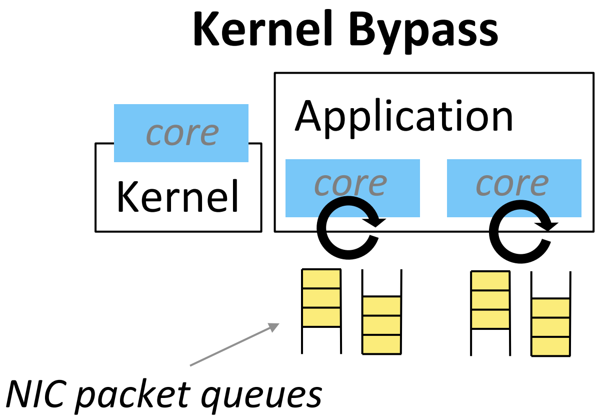
2018
Latency: ~5 µs
Throughput: 100 Gbits/s



- But today's operating systems add significant overheads to I/O

The Rise of Kernel Bypass





- Dedicate busy-spinning cores
- Applications directly poll NIC queues
- Enables higher throughput and lower latency

Trend #2: Slowing of Moore’s Law

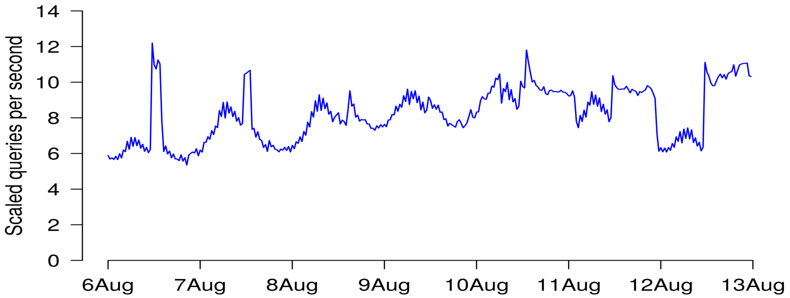
increased demand for servers

increased demand for energy

- CPUs only utilized 10-66% today
- CPU efficiency becomes increasingly important

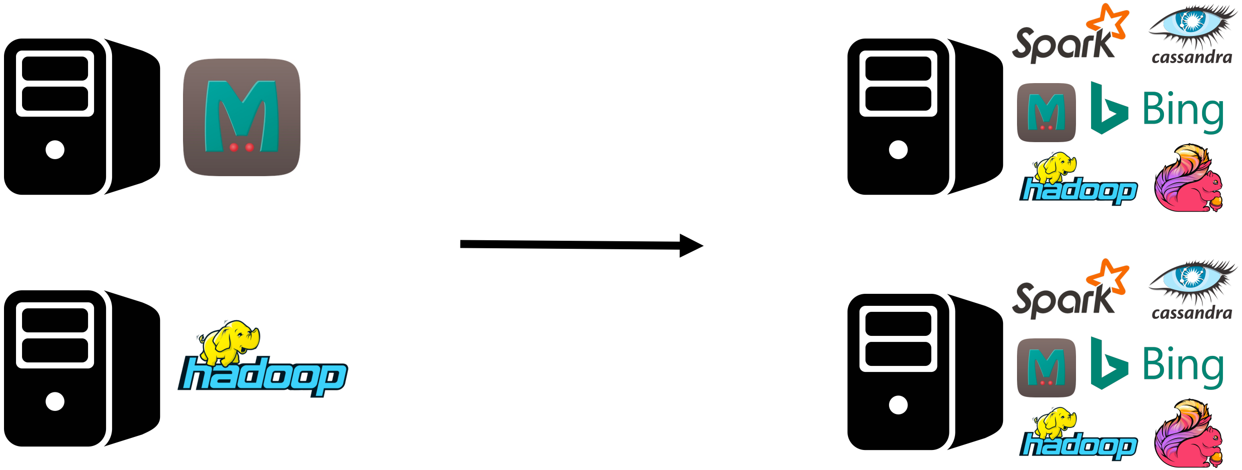
Load Variation Makes Efficiency Challenging

- Load variation for datacenter workloads
  - Days: diurnal cycles
  - Microseconds: packet bursts, thread bursts
- Peak load requires significantly more cores than average load



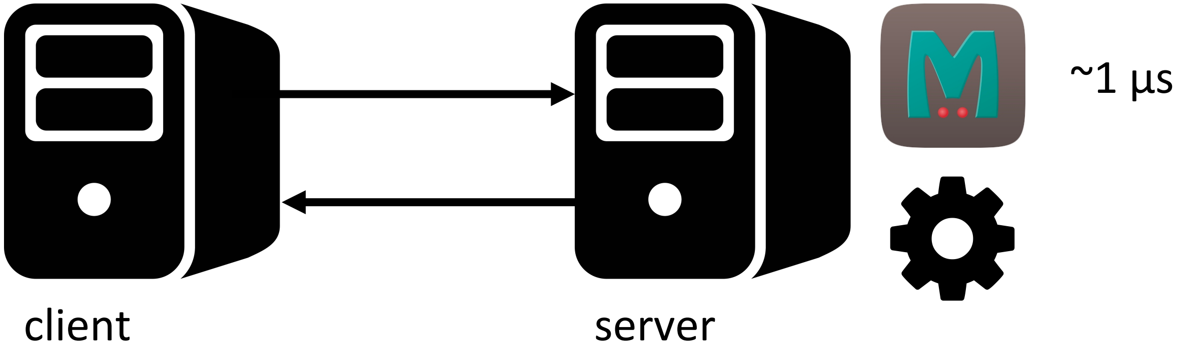
The Need for Multiplexing

- Two types of applications: latency-sensitive and batch-processing
- Pack both on the same server
  - Bing does this on over 90,000 servers

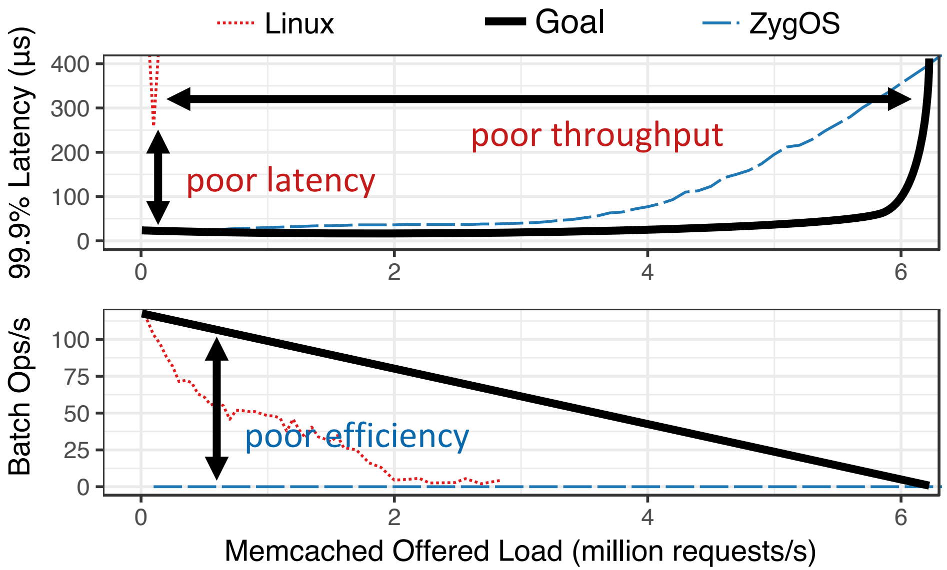


Multiplexing with Existing Approaches

- Example: Memcached + batch processing application



Multiplexing with Existing Approaches



No existing approach provides **high network performance** and **high CPU efficiency**

Goal

- Reconcile the tradeoff between high CPU efficiency and network performance
- Reallocate cores across applications at **microsecond** granularity
  - Coarser granularities insufficient for microsecond-scale tasks and microsecond-scale bursts

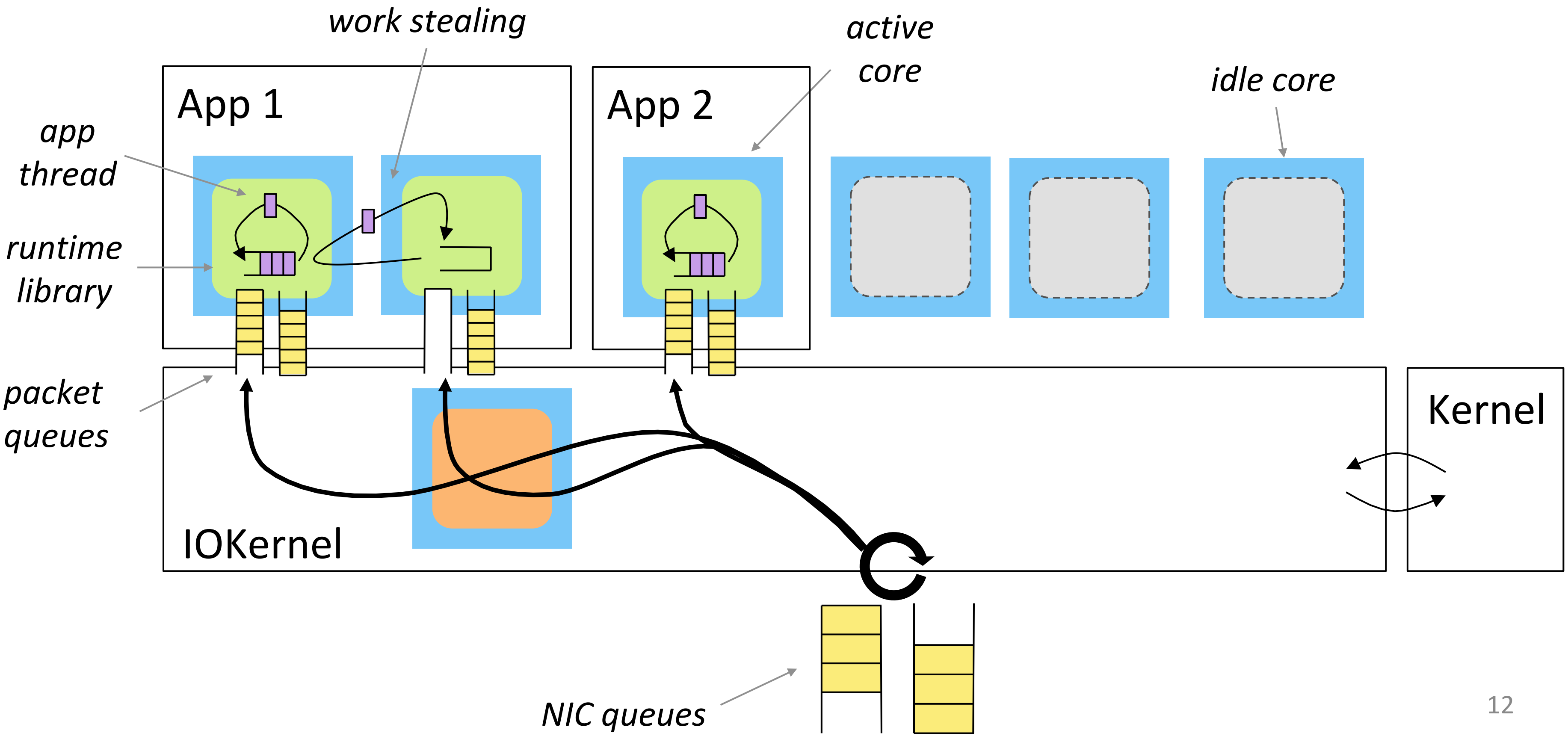
Challenges of Fast Reallocations

- How many cores does an application need?
  - Application-level metrics are too slow
  - Multiple sources of load: packets and threads
- Overhead of reallocation
  - Reconfiguring hardware is too slow
- Existing systems don’t address these challenges

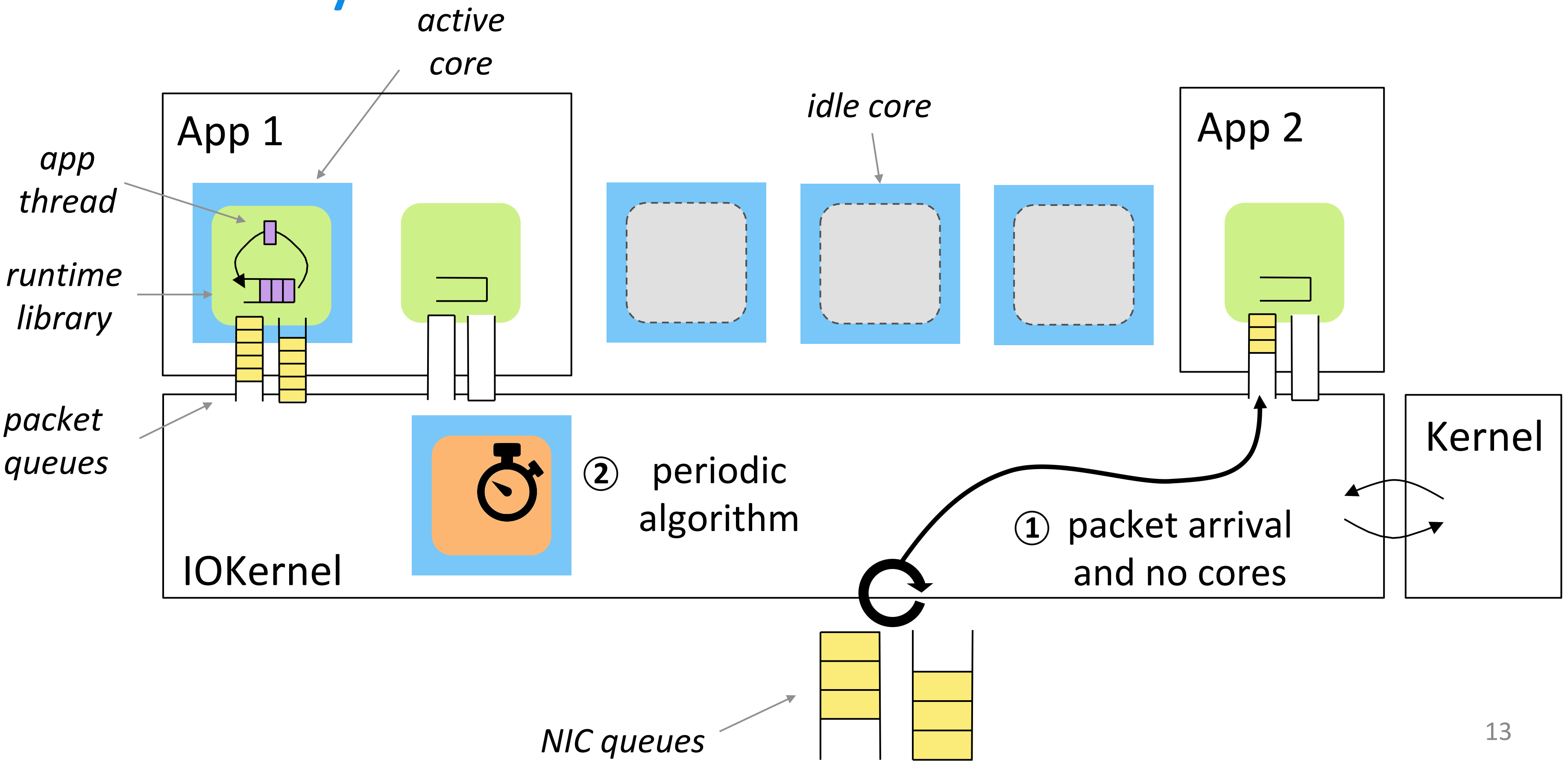
Shenango's Contributions

- Efficient **algorithm** for determining when an application needs more cores
  - Based on thread and packet queueing delays
- **IOKernel**: steers packets in software and allocates cores
  - Core reallocations take ~5 µs
- Cache-aware core selection algorithm
- Load balancing of packet protocol (e.g., TCP) handling

Shenango's Design

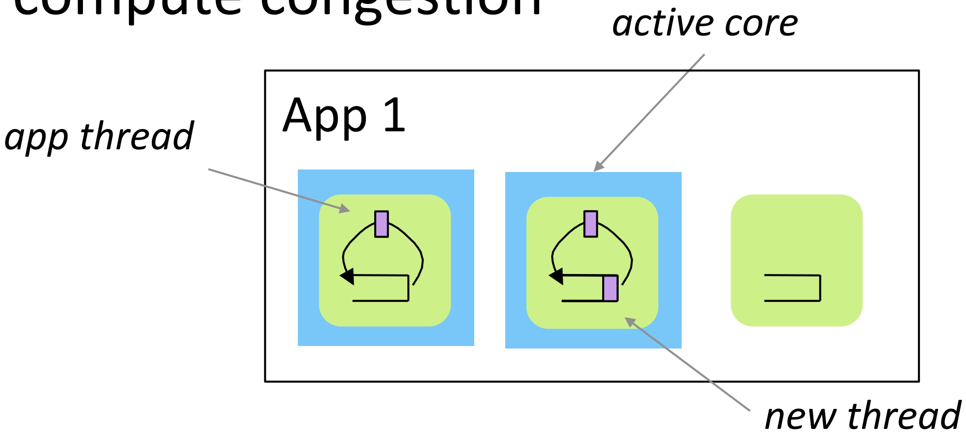


How Many Cores Should the IOKernel Allocate?



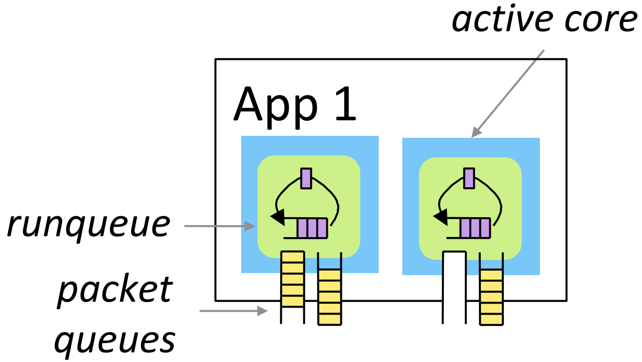
Compute Congestion

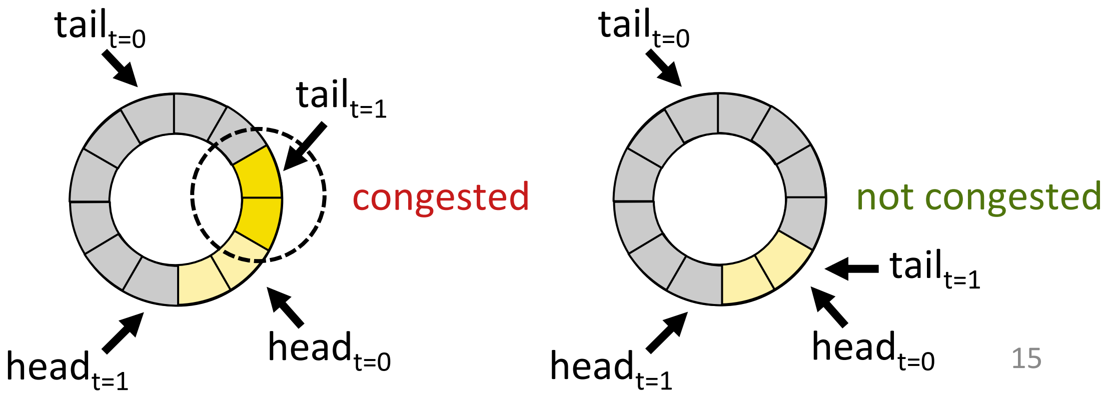
- Compute congestion: when granting an application an additional core would allow it to complete its work more quickly
- Goal: grant each application as few cores as possible while avoiding compute congestion



Congestion Detection Algorithm

- Queued threads or packets indicate congestion
- Any packets or threads queued since the last run (5 µs ago)?
  – Grant one more core
- Ring buffers enable an efficient check
  – $head_{t=n-1} > tail_{t=n}$ implies congestion





Implementation

- IOKernel
  - Uses DPDK 18.11
- Runtime
  - UDP and TCP
  - C++ and Rust bindings
- 13,000 lines of code total

Evaluation Questions

- How well does Shenango reconcile the tradeoff between CPU efficiency and network performance?
- How does Shenango respond to sudden bursts in load?
- How do Shenango's individual mechanisms contribute to its overall performance?

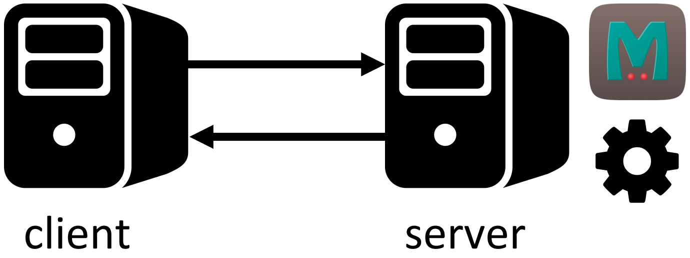
Experimental Setup

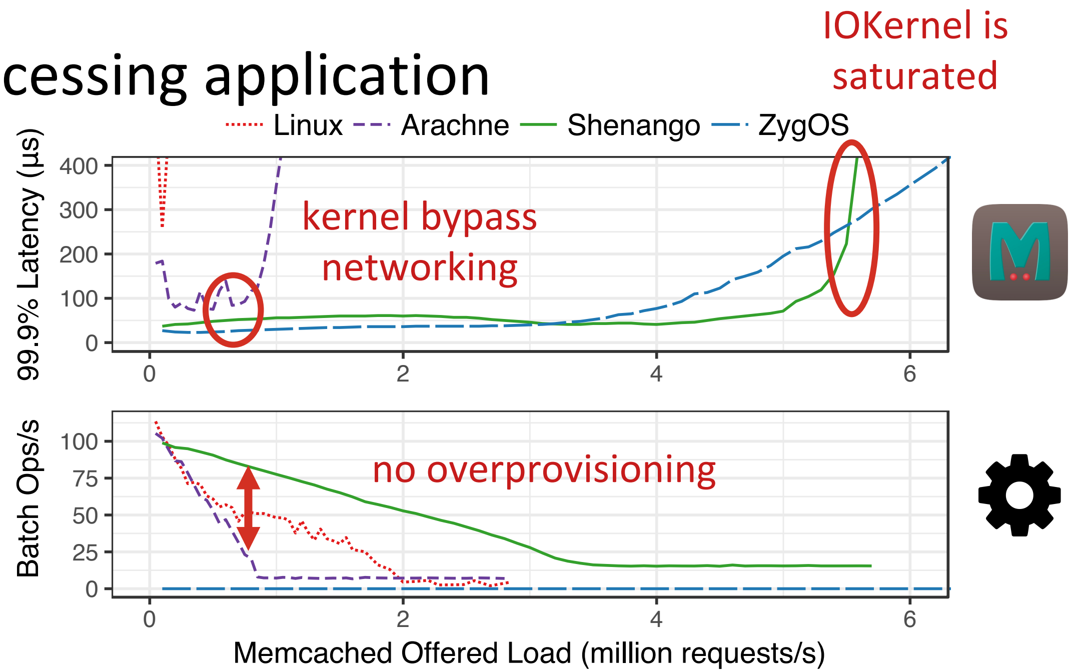
- 1 server + 6 clients, 10 Gbits/s NICs
- Clients run our open-loop load generator built on Shenango
  - Requests follow Poisson arrivals, use TCP

System	Kernel Bypass Networking	Lightweight Threading	Balancing Interval
Linux	✗	✗	4000 µs
ZygOS (SOSP '17)	✓	✗	N/A
Arachne (OSDI '18)	✗	✓	50000 µs
Shenango	✓	✓	5 µs

CPU Efficiency and Network Performance with Memcached

- Memcached + batch processing application

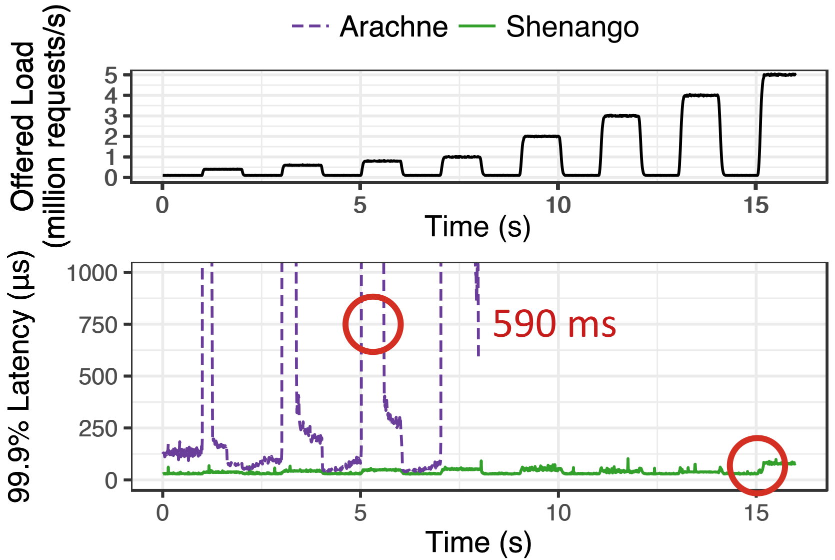




- Shenango matches ZygOS’s tail latency with high CPU efficiency

Shenango is Resilient to Bursts in Load

- TCP requests with 1 µs synthetic work + batch processing application
- Increase or decrease the load every 1 s



reallocates cores 10,000x as often

Conclusion

- Shenango reconciles the tradeoff between low tail latency and high CPU efficiency
- Reallocates cores at microsecond granularity
  - Efficient **congestion detection algorithm**
  - **IOKernel**: allocates cores and steers packets in software

https://github.com/shenango