# Concord: An Efficient Runtime for Microsecond-Scale Datacenter Applications


EPFL


Rishabh Iyer


Musa Unal


Marios Kogias

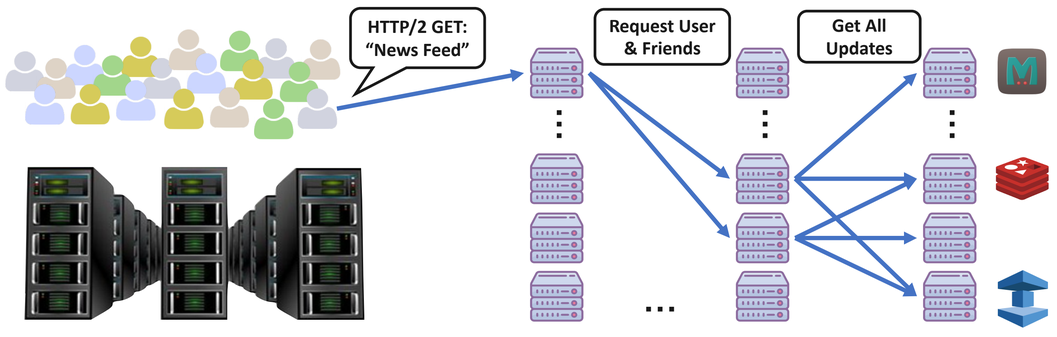

George Candea

Imperial College London


## Problem: Datacenter OSes that optimize application tail-latency sacrifice max throughput and generality

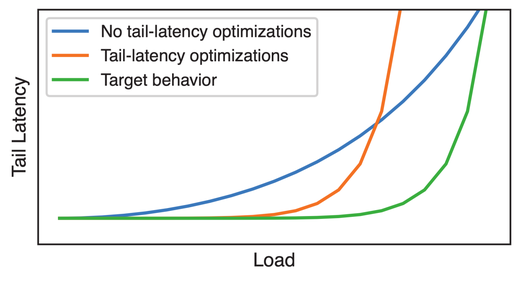
### Online services place stringent demands on the tail-latency of individual nodes with µs-scale service times

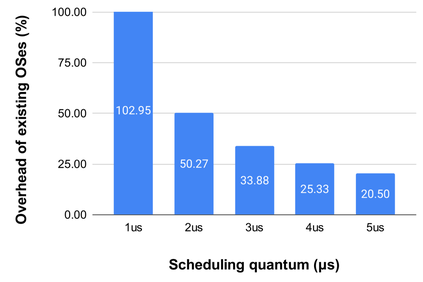
- E2E latency impacts revenue for online services
  - Amazon loses $1M for every 100ms increase in latency
- Datacenter services have large RPC fan-outs
  - E2E latency determined by slowest individual response




### Scheduling policies that optimize tail-latency incur significant system throughput overheads at µs-scale

- Three key sources of overhead:
  - Hardware interrupts for precise preemption
  - Cache coherence stalls due to a physical single queue
  - Dedicated dispatcher thread that doesn't contribute to goodput
- OSes sacrifice generality or deployability to recover throughput



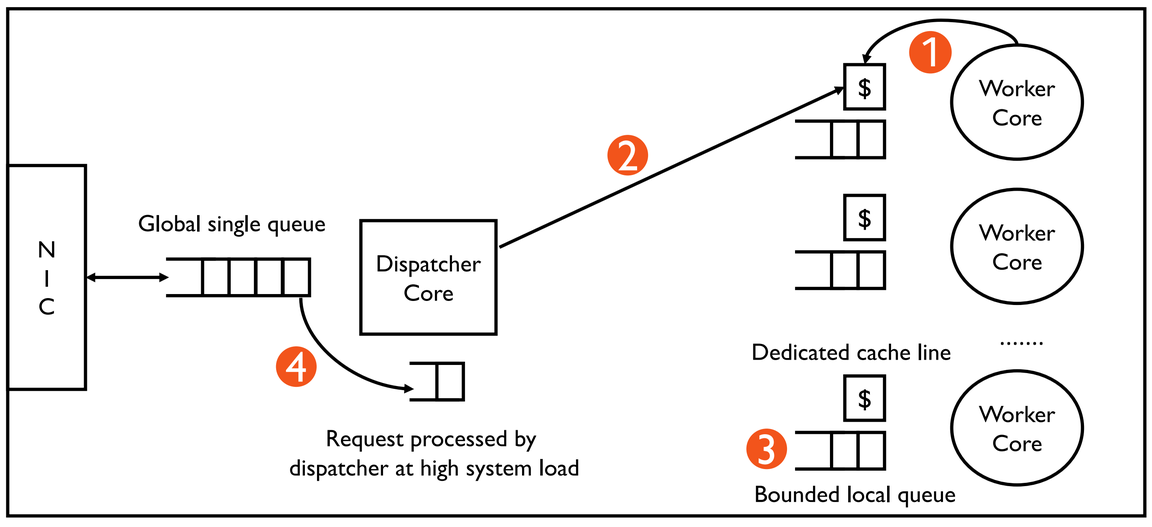




## Solution: Concord, an efficient, general scheduling runtime immediately deployable on the public cloud

**Key insight: *Approximate* theoretically optimal scheduling policies to reduce system overheads**

Concord co-designs applications and the runtime

- Compiler-enforced cooperation eliminates interrupts
  1. Instrumentation to periodically poll dedicated cache line
  2. Dispatcher initiates preemption by writing to cache line
- Join Bounded Shortest Queue (JBSQ) scheduling
  3. Bounded core-local queues, eliminates coherence stalls
- Work stealing dispatcher contributes to goodput
  4. Dispatcher begins processing requests if all workers are busy

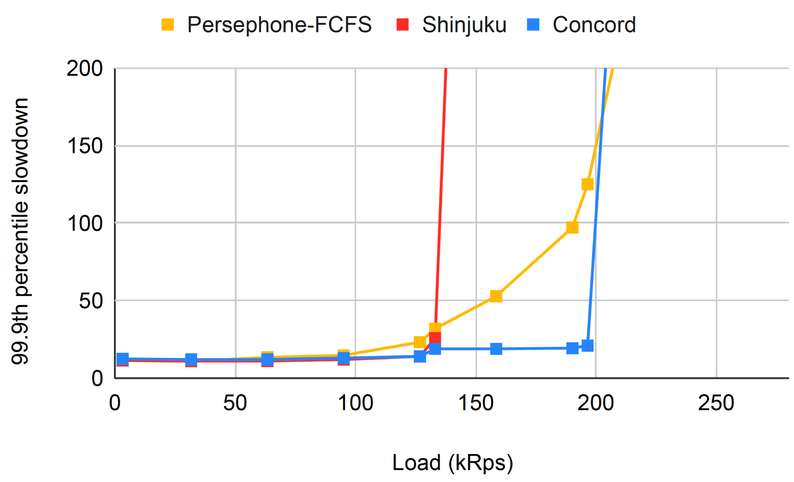



Unlike state-of-the-art datacenter OSes, Concord does not rely on application-level assumptions or non-standard use of hardware

## Evaluation: Concord improves application throughput by 18-83% for a given tail-latency SLO

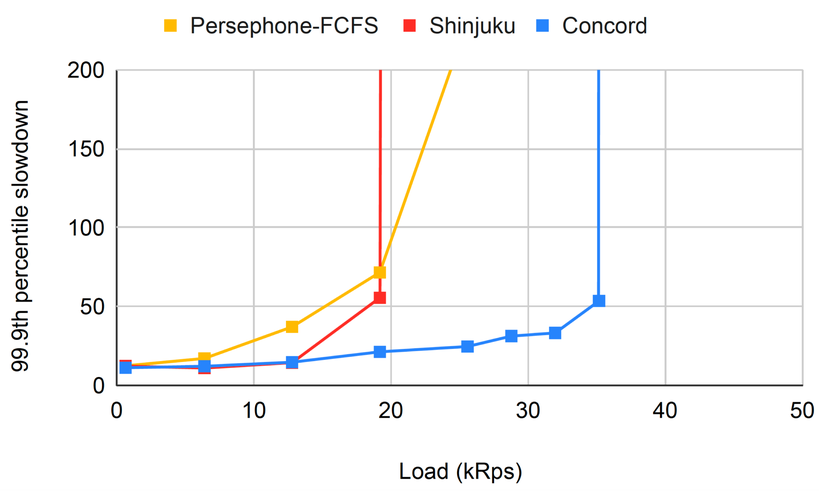
### Microbenchmarks

- Program that spins for duration specified by request
  - Can evaluate multiple service time distributions
- Measure throughput sustained for a target slowdown
  - Ratio of the total sojourn time to the service time.




### Google's LevelDB

- Workload: 50% GETs, 50% SCANs.
  - GETs take 600ns, SCANs take 600µs
- Key-value store is populated with 1500 unique keys




Concord supports 18-83% greater throughput than state-of-the-art datacenter OSes (Shinjuku [NSDI'19], Persephone [SOSP'21])

## Approximating---not implementing canonically---theoretically optimal scheduling significantly improves system throughput at µs-scale at negligible tail-latency costs