The Need for Low Latency in Data Centers

- The call for µs-scale and ns-scale processing
- Emerging userspace networking runtimes
- **Thread-dispatch and interrupt handling** are culprits!

---

**Graph:**

- One-way RPC Latency (µs)
  - Conventional Linux: ~35 µs
  - Kernel-bypass UDP: ~5 µs
  - Kernel-bypass TCP: ~10 µs
  - Library RDMA: ~20 µs
  - Raw RDMA: ~15 µs

- Cumulative Latency (µs)
  - RDMA: ~20-30 µs
  - Two-sided: ~5 µs
  - Thread Dispatch: ~10 µs
  - Interrupts: ~15 µs
  - RPC: ~25 µs
  - TCP: ~30 µs

---

[Zhang et al., SOSP 2021]
[Barroso et al., CACM 2017]
Process Scheduling Involved Everywhere!

Network Applications

Userland
Kernel space

RPC | Memcached | Nginx | RocksDB

Network packet arrives: Trigger the network stack

Incoming packets

Egress packets

Process Scheduler

Response is ready: prepare network packet(s)

Application threads asking for CPU time to produce a response

Process scheduler is invoked at least 3 times during a single network communication

Understanding Microquanta Process Scheduling for Cloud Applications, CloudAM 2022
Conventional Linux Schedulers Falling Short

- Running RocksDB benchmark on a single machine under three process schedulers.

Non-skewed workloads can benefit from Realtime scheduling by minimizing the interference!

Microquanta holds a middle-ground but raises its own issues!

Non-preemptive realtime scheduling is unfit for skewed workloads due to HoL blocking!
Introducing Three Representative Schedulers

- Strictly higher priority
- Non-preemptive

- Strict Fairness
- Time-sharing

Latency sensitive (Higher Prio)
CPU intensive (Lower Prio)
Microquanta Scheduling

- Per-CPU FIFO queues
- Microsecond-scale scheduling between processes
- Tunable CPU allocation via Runtime and Period Parameters

Understanding Microquanta Process Scheduling for Cloud Applications, CloudAM 2022
Impact of Microquanta Parameter Setting on Application Performance

Strictly favoring Memcached threads

Finer period settings perform better
Microquanta and Fast Load-Balancing

- 500 benchmark threads pinned to core #1 -> Released on 10th second
- The schedulers start distributing threads

10-second convergence time for CFS!

Non uniform load distribution, hot zones still exist

Very fast convergence for Microquanta ~1s

Uniform load-distribution
Application Performance Comparison

Uncontested

Similar latency performance

Contested

Network application must be prioritized!

Load-balancing is crucial!

Microquanta cannot benefit the busy-polling threads of IO-kernel

Understanding Microquanta Process Scheduling for Cloud Applications, CloudAM 2022
The Future of Process Scheduling

• Linux process scheduling is challenged by skewed workloads
• Parameter-based scheduling faces tuning issues
• Design space of process schedulers
  • Schedulers that can learn and adapt to workload changes
  • Schedulers that are tied to applications logic
    • Kernel-bypass runtimes (Shinjuku, Caladan)
    • Userspace thread-management (Arachne)
    • In-application scheduling (Ghost, Peafowl)

Microquanta Kernel Repository: https://github.com/erfan111/linux_uquanta
Contact: erfan@cs.jhu.edu