No Tradeoff

Low Latency + High Efficiency

Christos Kozyrakis

http://mast.stanford.edu
Latency-critical Applications

A growing class of online workloads

Search, social networking, software-as-service (SaaS), …

Occupy 1,000s of servers in datacenters
Example: Web-search

Web Servers

Front end

Back end

Metric: queries per sec (QPS) at a 99th percentile latency threshold

Massive distributed dataset, multiple tiers, high fan-out
Characteristics

High throughput + low latency (100μsec – 10msec)

Focus on tail latency (e.g., 99th percentile SLO)

Distributed and (often) stateful

Multi-tier with high fan-out

Diurnal patterns but load is never 0
Trends

“Death Star” Architecture Diagrams

Apps as collection of micro-services
Loosely-coupled services each with bounded context
Even lower tail latency requirements
From 100s to just a few μsec

[Adrian Cockcroft’14]
Conventional Wisdom for LC Apps
#1 LC apps cannot be power managed
LC Apps & Power Management

**Power (DVFS off)**

Example diurnal patterns for search

Search load

**Tail latency (DVFS on)**

Latency for search with cpufreq

[Lo et al’14]
#2 LC apps must use dedicated servers
Achieved QoS w/ two low-latency workloads

- **Green**: Both meet QoS
- **Yellow**: 1ms RPC fails
- **Red**: Memcached fails
- **Black**: Both fail to meet QoS

**QoS requirement**

\[95^{\text{th}} < 5 \times \text{low-load } 95^{\text{th}}\]

[Leverich’14]
#3 LC apps must use local Flash
Local Vs Remote NVMe Access

![Graph showing Local Flash, iSCSI (1 core), and libaio+libevent (1 core) performance with p95 read latency (us) vs IOPS (Thousands). The graph illustrates a 2x latency and a 75% throughput drop.](image)
Sharing NVMe Flash

Flash performance varies a lot with read/write ratio

The curve you paid for

The curve you get with sharing
#4 LC apps must bypass the kernel for I/O

cannot use TCP

cannot use Ethernet

must use specialized HW
LC Apps & Linux/TCP/Ethernet

64-byte TCP echo

- HW Limit
- Linux

4.8x Gap

8.8x Gap

[Belay et al’15]
Conventional Wisdom for LC Apps
Low Latency + High Efficiency

Understand the problem

Understand the opportunity
Understanding the Latency Problem

Latency-critical apps as queuing systems

Tail latency affected by

Service time, queuing delay, scheduling time, load imbalance
Understanding the Latency Problem

![Graph](image)

Latency (usecs)

Memcached QPS (% of peak)

Provisioned QPS

95th-%

[Provisioned QPS](#)

[Leverich'15](#)
Understanding the Opportunity

**Iso-latency**: maintain constant tail latency at all loads

Enables power management, HW sharing, adaptive batching, …

Key point: use tail latency as control signal

[Lo et al’14]
Low Latency + High Efficiency

Understand the problem

Understand the opportunity

End-to-end & latency-aware management
Pegasus: Iso-latency + Fine-grain DVFS

Measures latency slack end-to-end
Sets **uniform** latency goal across all servers

RAPL as knob for power
Power is set by workload specific policy

[Lo et al’14]
Pegasus: Iso-latency + Fine-grain DVFS

Achieves dynamic energy proportionality

[Lo et al’14]
**Heracles: Iso-latency + QoS Isolation**

**Goal:** meet SLO while using idle resources for batch workloads

End-to-end latency measurements control isolation mechanisms
Heracles: Iso-latency + QoS Isolation

Latency readings

Can BE grow?

Internal feedback loops

[Lo et al'15]
Heracles: Iso-latency + QoS Isolation

[Lo et al'15]
Iso-latency + QoS Isolation

LC apps + best-effort tasks on the same servers

HW & SW isolation mechanisms (cores, caches, network, power)

Iso-latency based control for resource allocation

[Lo et al'15]
Low Latency + High Efficiency

Understand the problem

Understand the opportunity

End-to-end & latency-aware management

Optimize for modern multi-core hardware

Specialize the SW
System SW for Low Latency + High BW
System SW for Low Latency + High BW
System SW for Low Latency + High BW

- Control plane
- App 1
  - RX
  - TX

- App 2
  - RX
  - TX

- OS kernel
  - Dune

- Guest
- Ring 0

- Host
- Ring 0

- Ring 3
System SW for Low Latency + High BW

[Belay et al'14]
Run-to-Completion

Improves data cache locality
Removes scheduling unpredictably
Adaptive Batching

Enabled by ISO-latency
Improves instruction cache locality and prefetching
IX: Low Latency + High Bandwidth

Latency (µs)

SLA

USR: Throughput (RPS x 10^6)

IX: Tail lower than Linux avg

2x Less Tail Latency

5x More RPS

+ TCP
+ commodity HW
+ protection
+ 1M connections

[Belay et al’14]
Low Latency + High Bandwidth + High Efficiency

![Graphs showing MC load, MC latency, Power, and BE Performance over time](image)
ReFlex: IX + QoS Scheduling
ReFlex: IX + QoS Scheduling

Ring 3
Event Conditions

Guest Ring 0

NVMe
TCP/IP
CQ
RX

ReFlex Server
libIX
3
Batched Syscalls

Scheduler

TCP/IP

TX
SQ

[Klimovic et al’17]
ReFlex: IX + QoS Scheduling

ReFlex Server

libIX

Batched Syscalls

Scheduler

Guest Ring 0

Ring 3

Event Conditions

NVMe

TCP/IP

CQ

RX

TX

SQ

[5]

[6]

[7]

[8]

[NVMe]

[TCP/IP]

Klimovic et al’17
ReFlex: Local $\approx$ Remote Latency

- *Local-1T*
- *ReFlex-1T*
- *Libaio-1T*

**Linux:**
75K IOPS/core

**ReFlex:**
850K IOPS/core
ReFlex: Local $\approx$ Remote Latency

Unloaded latency:
- Local Flash: 78 $\mu$s
- ReFlex: 99 $\mu$s
- Libaio: 121 $\mu$s
ReFlex: Local $\approx$ Remote Latency

ReFlex saturates Flash device
ReFlex: Private $\approx$ Shared QoS

Tenants A & B: latency-critical; Tenant C + D: best effort
Without scheduler: latency and bandwidth QoS for A/B are violated
Scheduler rate limits best-effort tenants to enforce SLOs
ReFlex: Private ≈ Shared QoS

Tenants A & B: latency-critical; Tenant C + D: best effort
Without scheduler: latency and bandwidth QoS for A/B are violated
Scheduler rate limits best-effort tenants to enforce SLOs
ReFlex: Private ≈ Shared QoS

Tenants A & B: latency-critical; Tenant C + D: best effort
Without scheduler: latency and bandwidth QoS for A/B are violated
Scheduler rate limits best-effort tenants to enforce SLOs
Looking Forward

Programming models for low tail latency
   Especially for multi-tier applications
Variability within and across applications
   Short vs long requests
Cluster-level issues
   Provisioning, scheduling, load-balancing, throttling, …
μsecond-scale tail latency
   Can we still get efficiency at ultra low latency?
Hardware for latency-critical applications
   Big vs little cores, isolation mechanisms, accelerators
Conclusions

Conventional wisdom is wrong

  Low latency is compatible with high efficiency
  Efficiency: energy proportionality, resource usage, throughput

Encouraging initial results

  Room for improvements throughout the stack

[Source:ThinkStock]