SHANGHAI JIAO TONG UNIVERSITY

SJTU

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Zero Overhead Monitoring for Cloud-native Infrastructure using RDMA

Zhe Wang, Teng Ma, Linghe Kong, Zhenzao Wen, Jingxuan Li, Zhuo Song, Yang Lu, Yong Yang, Tao Ma, Guihai Chen, Wei Cao

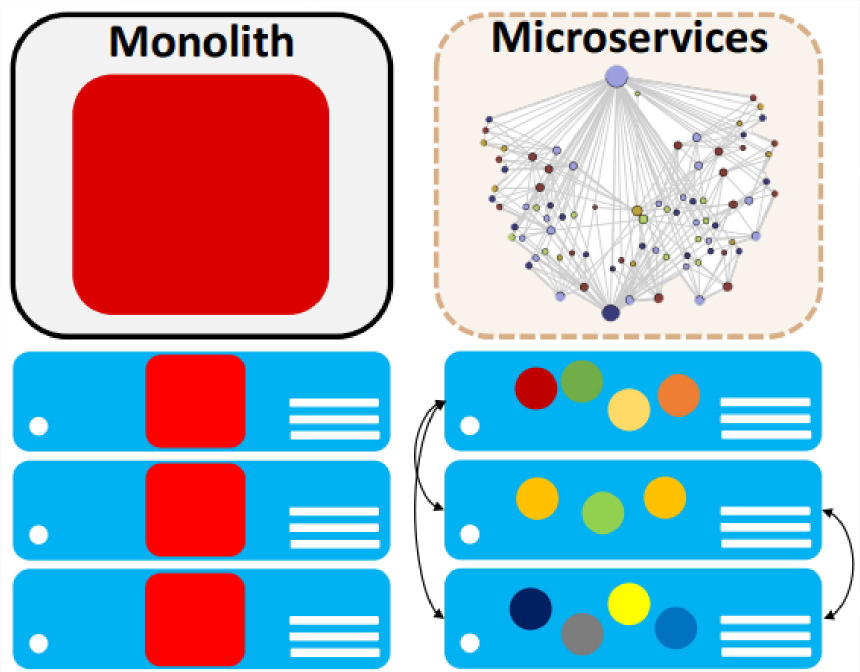
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Alibaba Cloud

饮 水 思 源 • 爱 国 荣 校

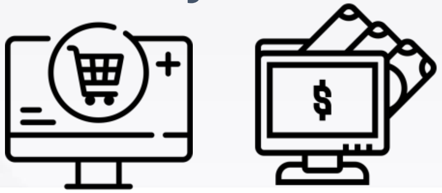
Contents

Contents

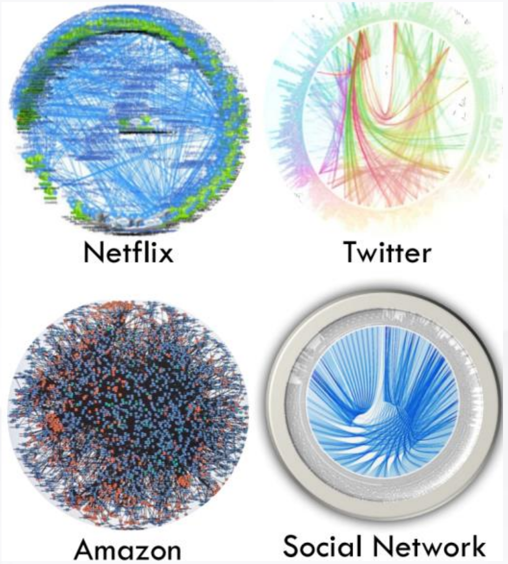


Cloud-native infrastructure:

- ✓ Monolithic design ---> microservices
- ✓ Dense deployment
- ✓ Disposable and immutable system
- ✓ Various applications

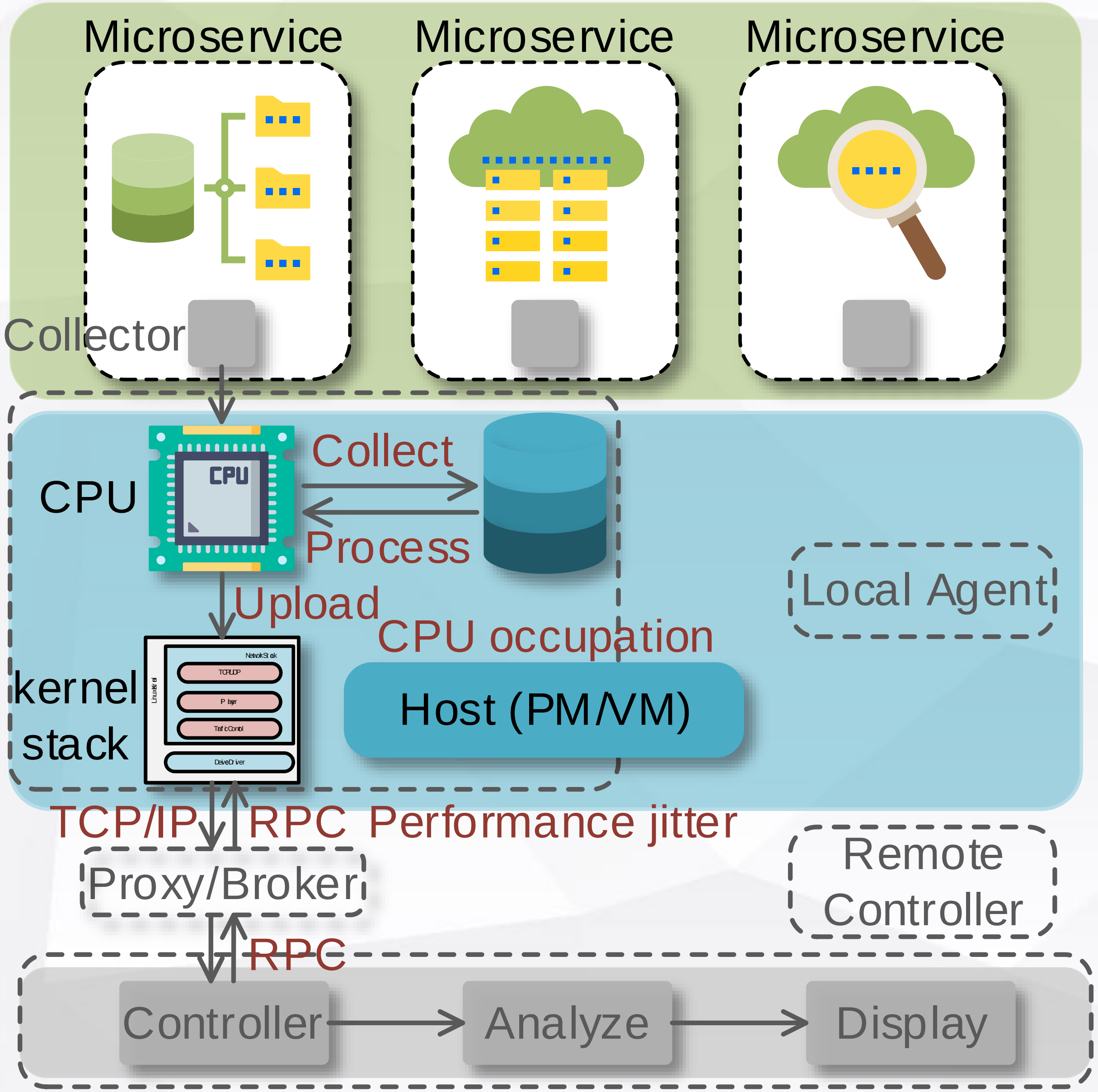


Cloud-native computing



Implications:

- ✓ Stricter QoS
- ✓ Highly resource constrained
- ✓ Massive metrics
- ✓ Rapid variations



Cloud-native monitoring

Monitor---service interference

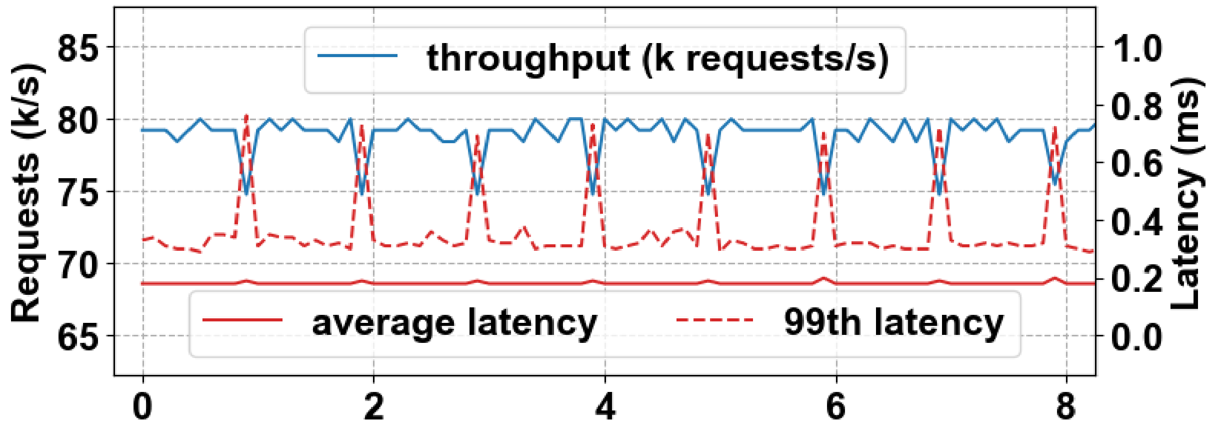
- ✓ High CPU utilization
- ✓ CPU bonding vs. default scheduling

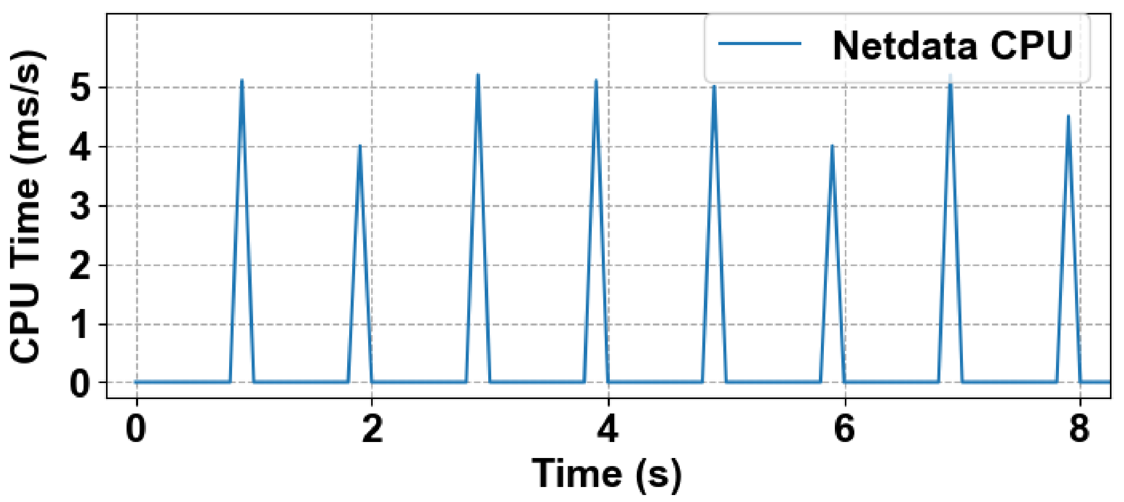
Service & monitor resource contentions

Service---monitor interference

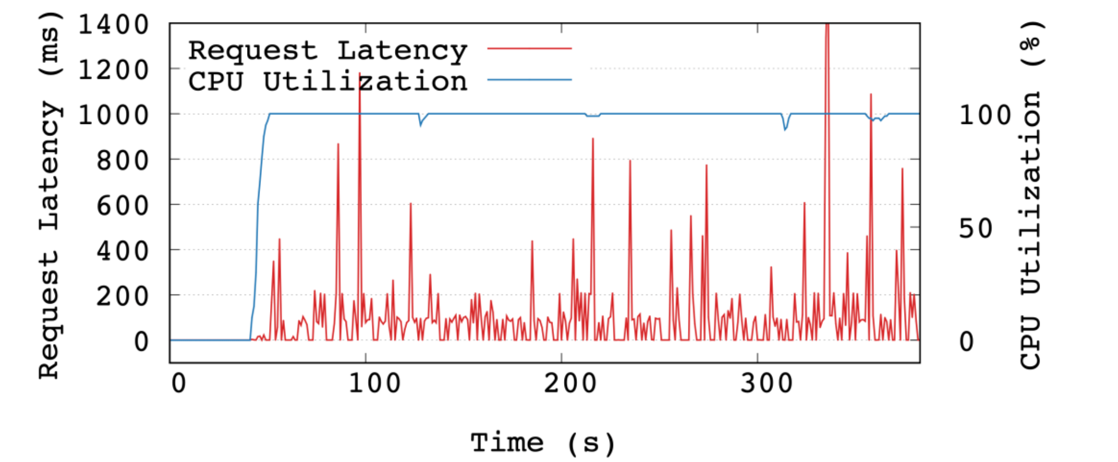
- ✓ CPU quota
- ✓ Interface reusing

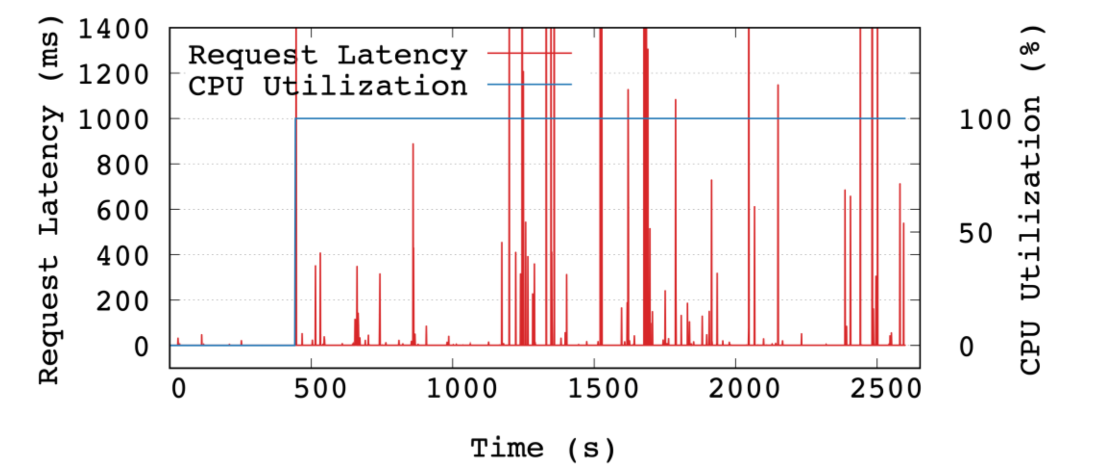
Service jitters

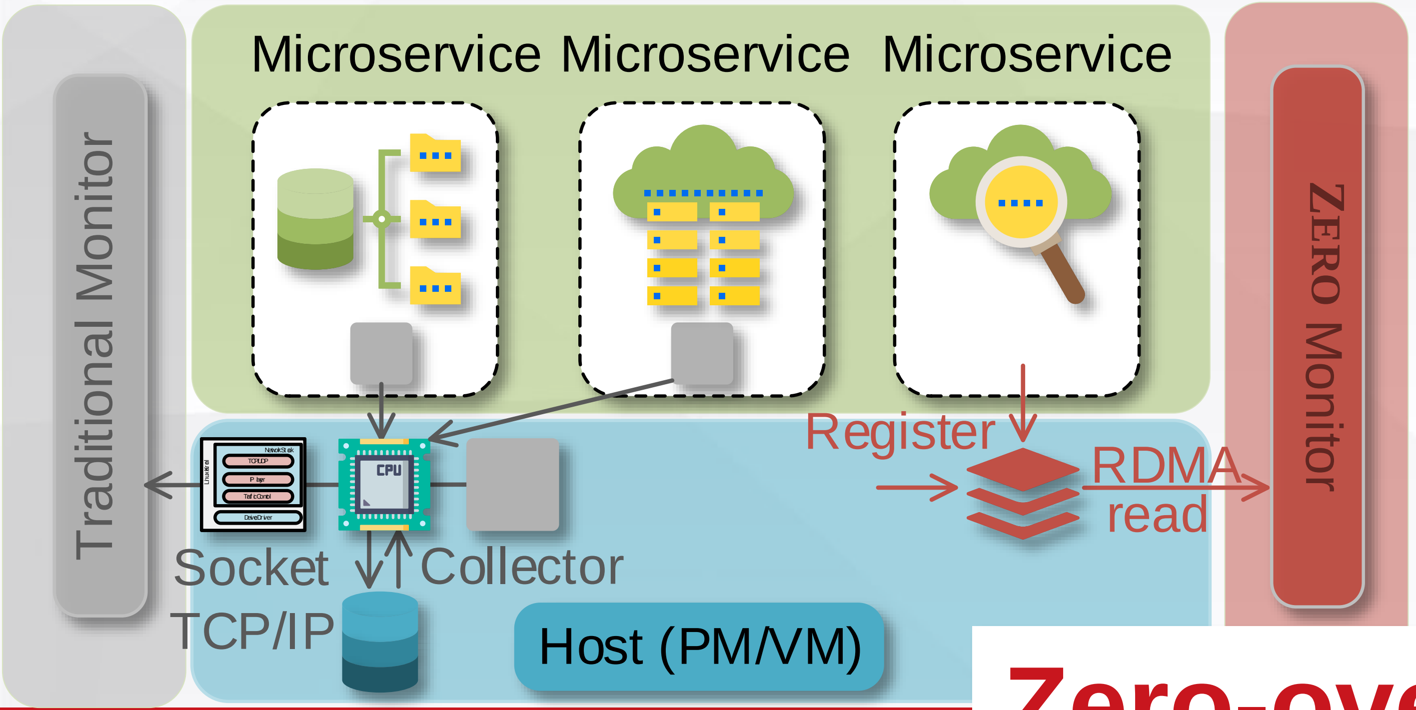




Monitor jitters







Decoupling monitor from infrastructure!

Zero-overhead monitoring

How to decouple?

Metric features

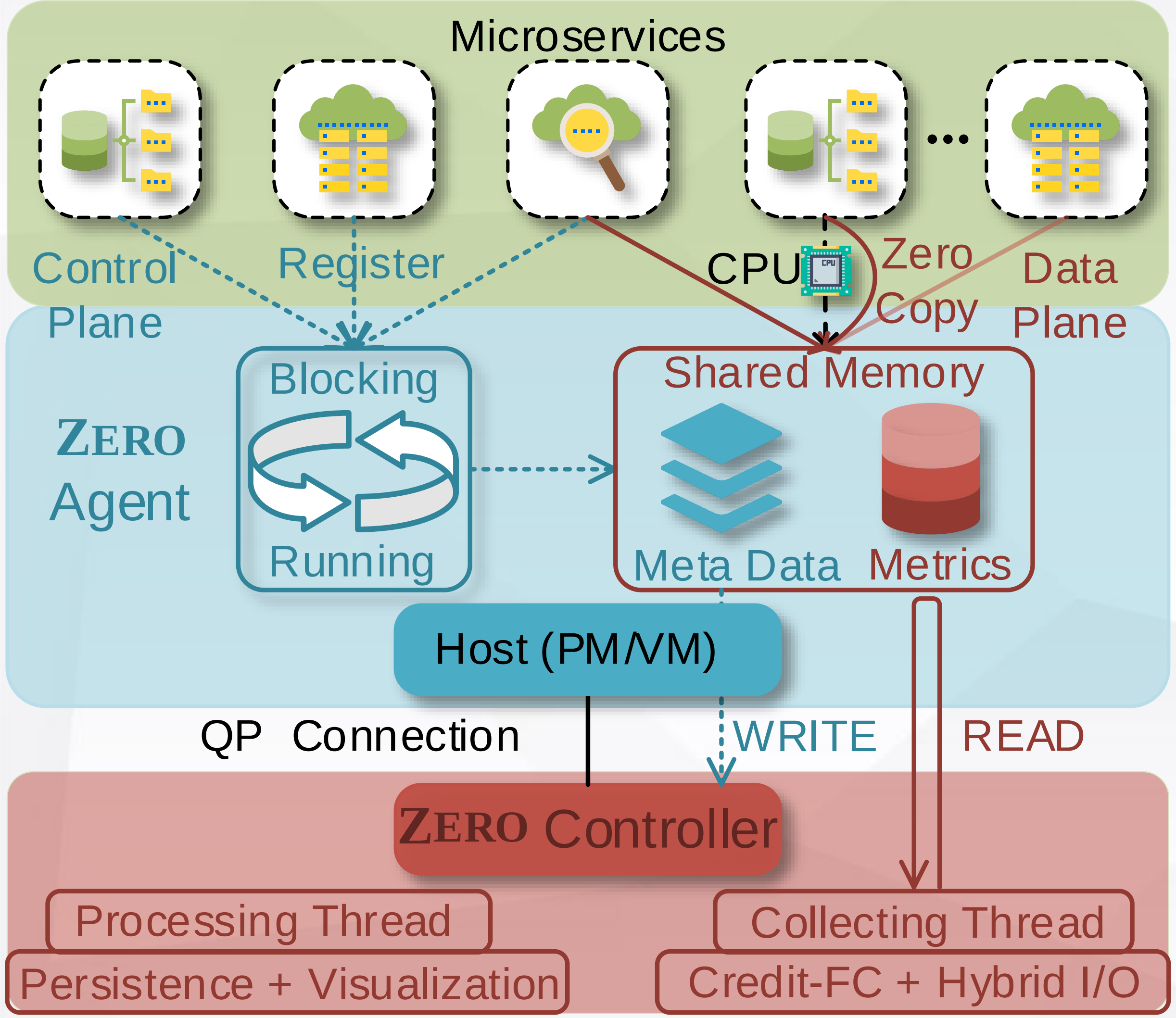
- ✓ Counters & reproducible calculation

RDMA support

- ✓ One-sided RDMA (CPU & kernel bypass)

Contents

Zero Overview



Challenge 1

- ✓ Offload collect overheads besides upload overheads

Challenge 2

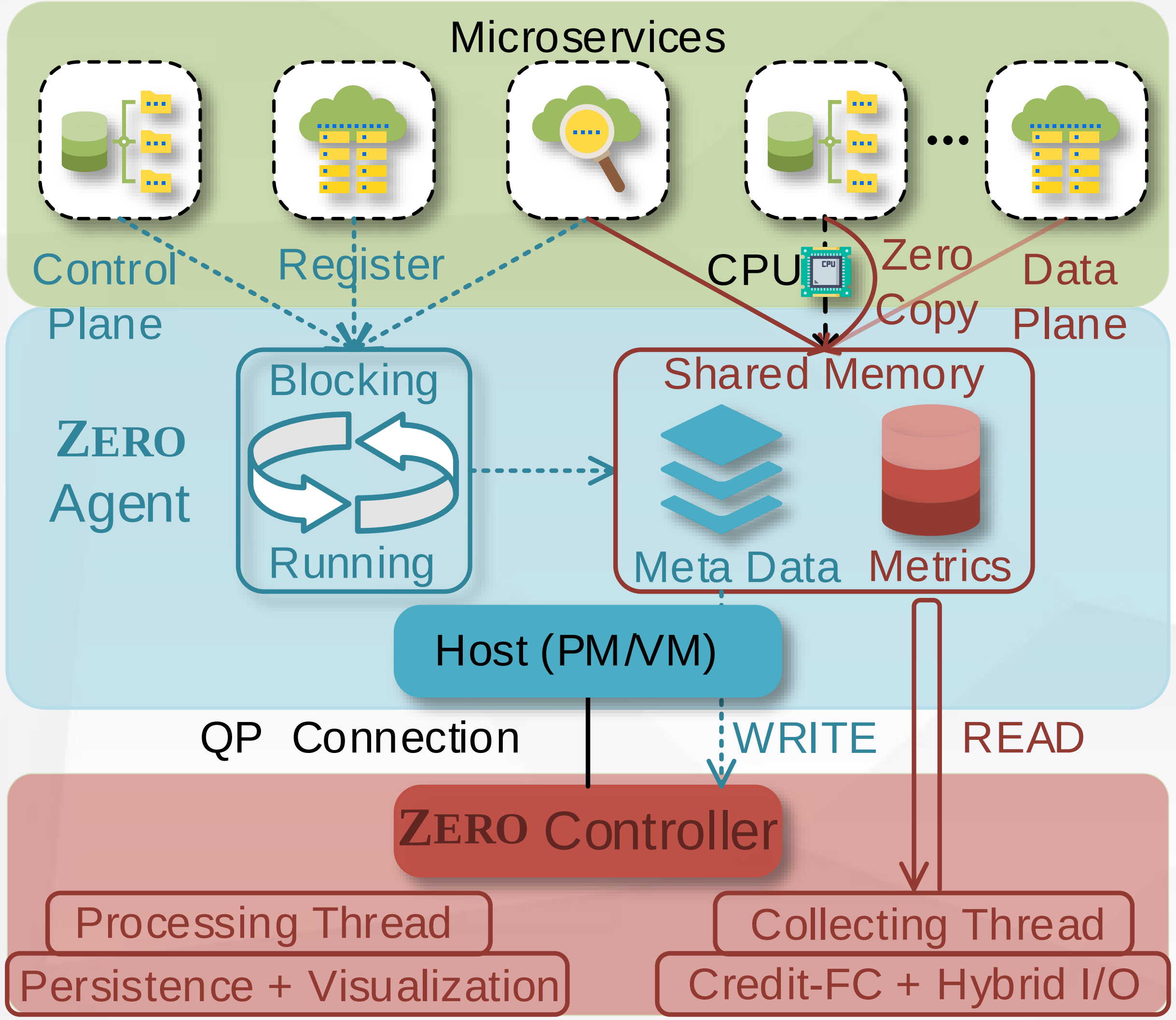
- ✓ Achieve high throughput while avoiding incast

Challenge 3

- ✓ Collect & process metrics from multiple connections

SJTU

Control Plane



Universal interface

- ✓ (De)register metrics
- ✓ Update metadata in control region

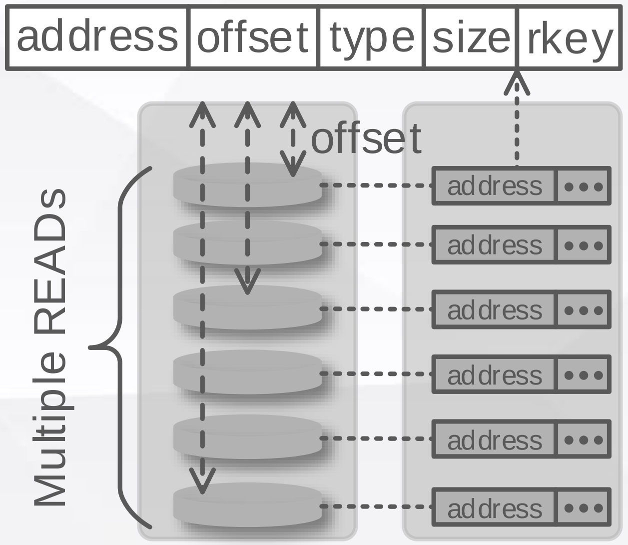
Disposable overhead

- ✓ System/persistent/tidal metrics
- ✓ Serverless functions

QP connection share

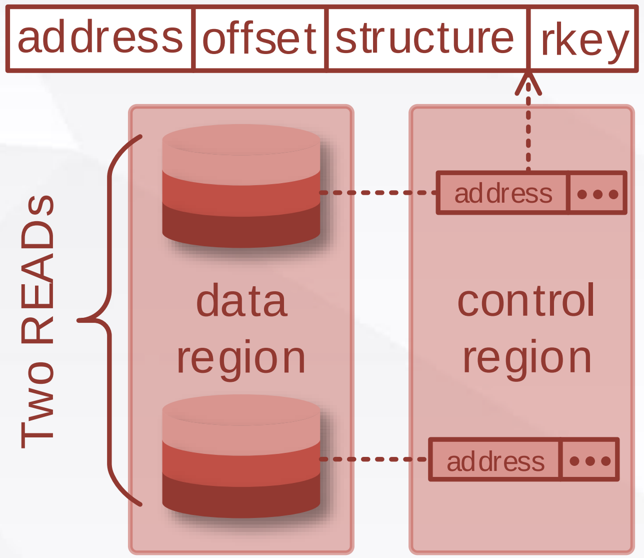
- ✓ Manage & share QP connection

SJTU

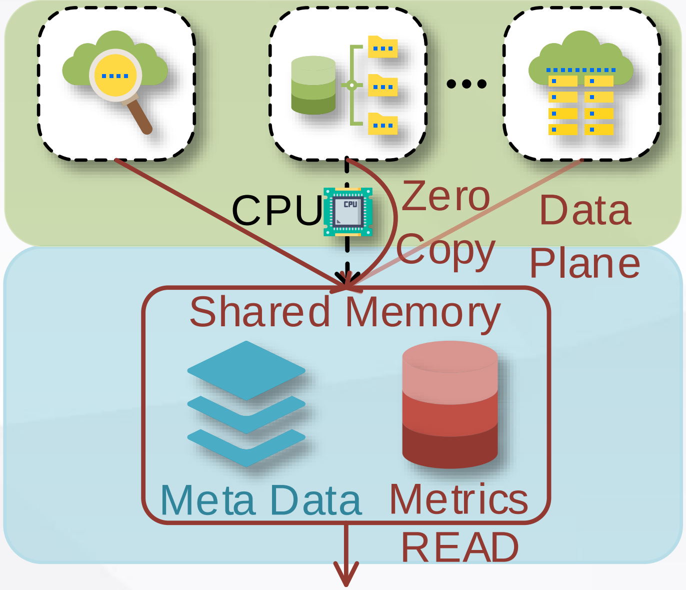


Shared memory

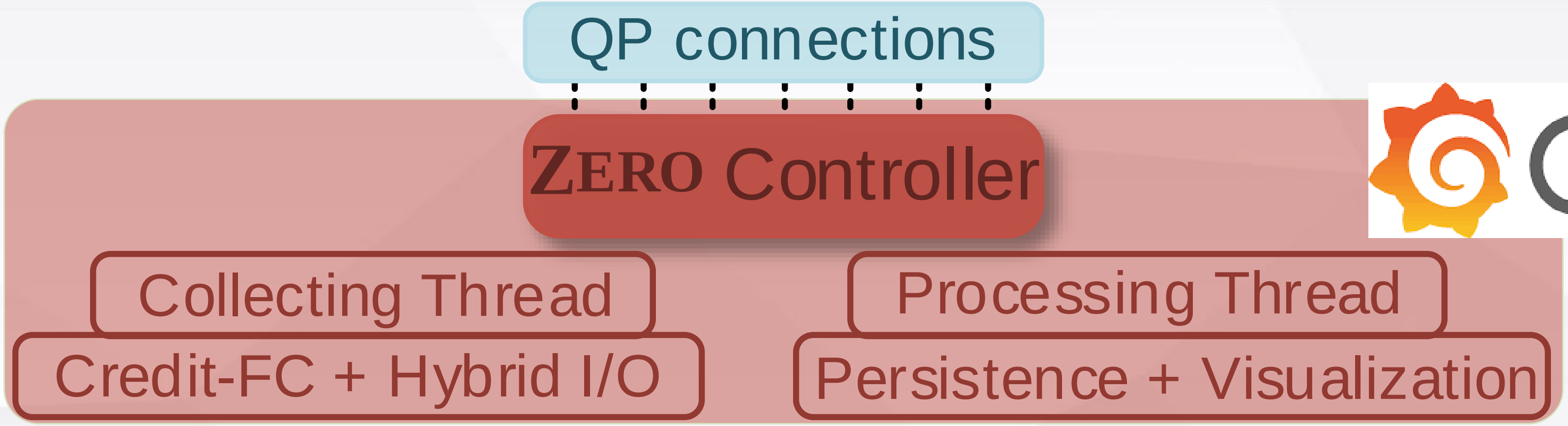
Memory management



Data Plane



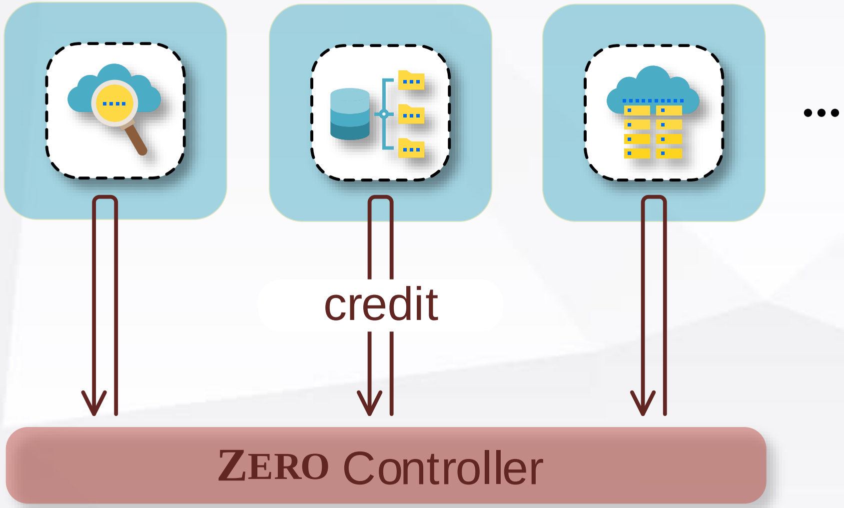
- ✓ Zero copy
- ✓ Zero CPU involvement
- ✓ Reduce MR entries and READs



Receiver-driven CC

Thread dispatch

Scale-out monitoring



- ✓ **Efficient threading and I/O model**
- ✓ **Avoid incast with many connections**
- ✓ **Guaranteed QoS level via receiver-driven model**

Contents

Implementation & Evaluation

Implementation

- ✓ Zero framework
- ✓ Case studies

Linux redis eBPF

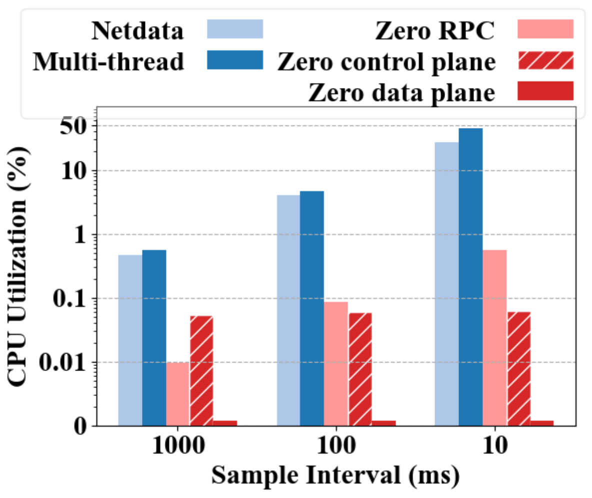
```
// type one, specifying attributes of variables
struct disk my_disk{
    .disk            = "sda",
    .hash    = 0x000f3456, ...
} __attribute__((section(".zero_init"));
//type two, using allocator
struct disk *my_disk = zero_malloc(sizeof(struct disk));
```

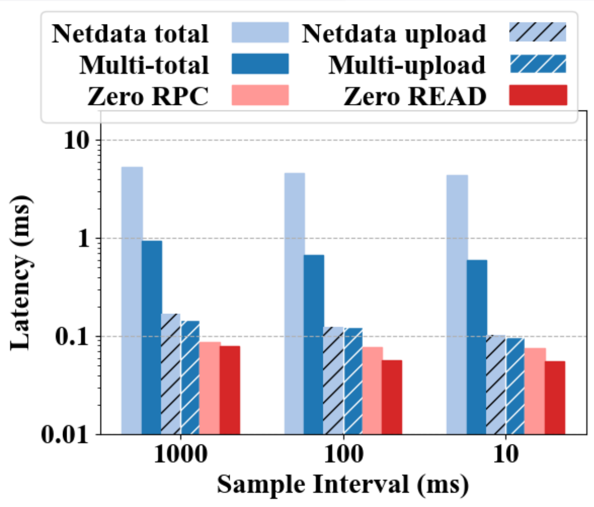
Evaluation Setup

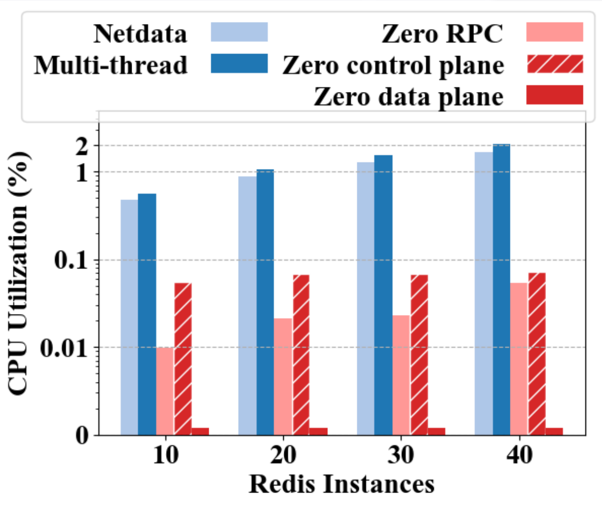
- ✓ Test Clusters

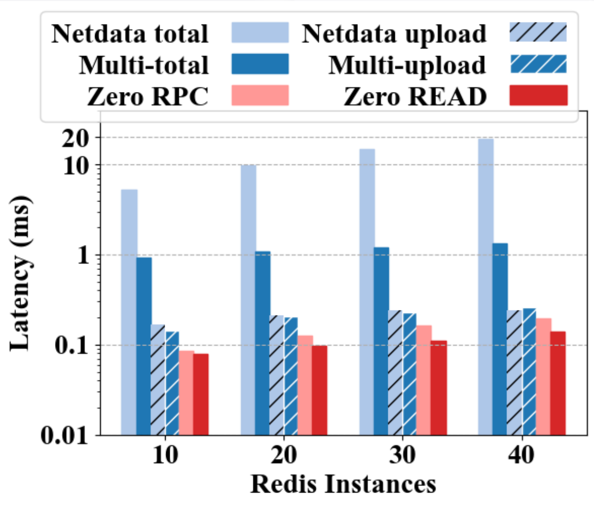
Name	Hosts	OS kernel	Intel Xeon CPU code	Mellanox NIC	Protocol	ECN	PFC
Cluster1	VMs	Linux 5.5	E5-2682 (64 cores)	2 × 25GbE ConnectX-4 Lx	RoCEv1/2	✗	✓
Cluster2	Containers	Linux 3.10	Platinum 8369B (64 cores)	200GbE ConnectX-6 Dx	RoCEv1/2	✓	✓

- ✓ Benchmarks： *CPU utilization (both sides), latency, throughput*
- ✓ Parameters： *Sampling interval (QoS), Instances (Metrics), Hosts (Connections)*
- ✓ Baselines： *Legacy tools, Netdata, Zero RPC (SEND/RECV)*





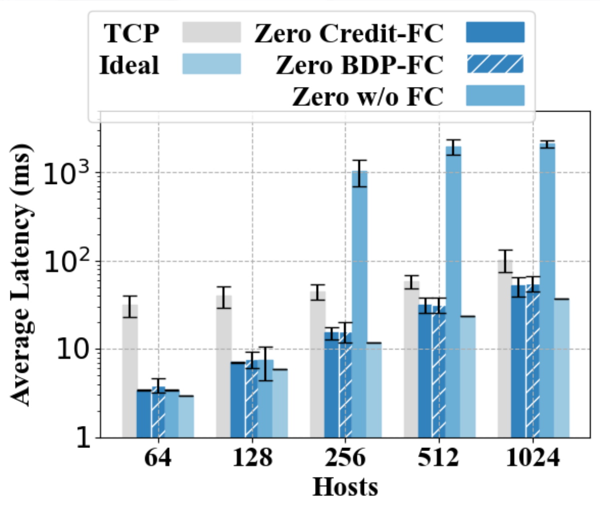


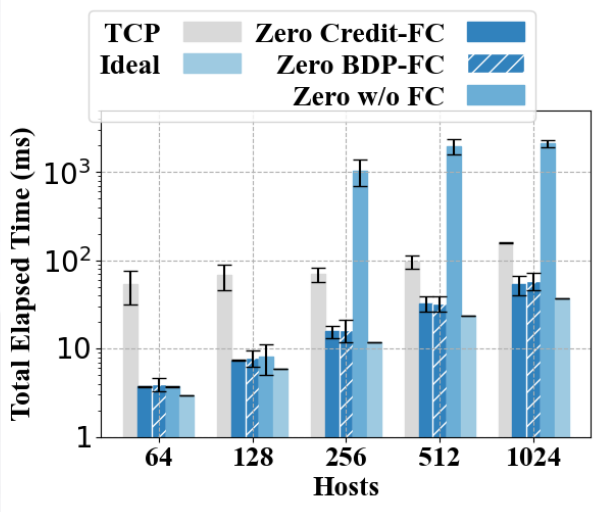


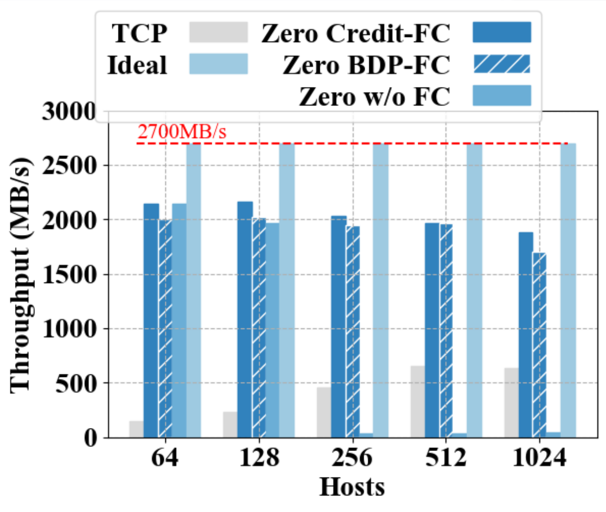
Zero Overhead

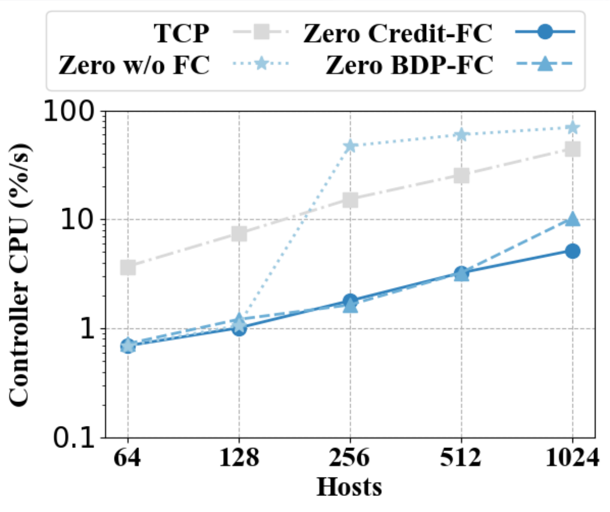
Metric	Monitor	Redis	Kernel	eBPF
Total Latency (ms)	Baseline	0.7 ~ 19.3	0.5 ~ 1.6	0.8 ~ 12.5
	ZERO RPC	0.08 ~ 0.18	0.14 ~ 0.36	0.10 ~ 1.02
	ZERO	0.05 ~ 0.14	0.07 ~ 0.23	0.08 ~ 0.87
Agent CPU Utilization (%)	Baseline	0.5 ~ 45	0.01 ~ 4	0.08 ~ 6
	ZERO RPC	0.01 ~ 0.55	0.08 ~ 0.9	0.05 ~ 0.68
	Control plane	0.05 ~ 0.07	0.8 ~ 1.5	0.04 ~ 0.05
	Data plane	0	0	0

- ✓ Disposable overhead at control plane
- ✓ Zero overhead at data plane
- ✓ Reduce latency by 1~2 order of magnitudes

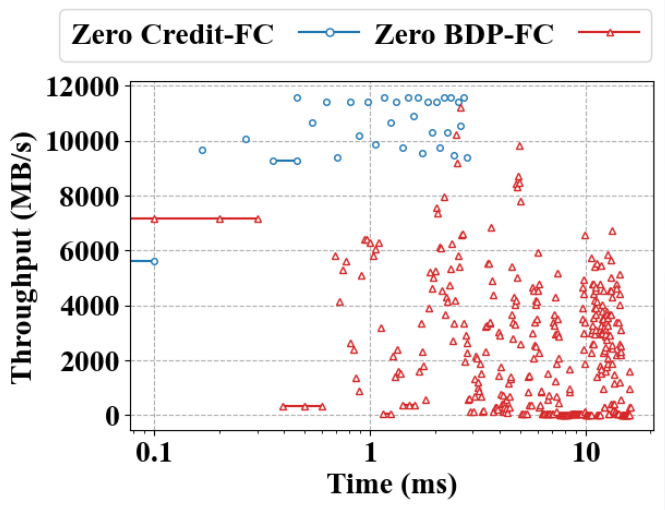








Zero Scalability



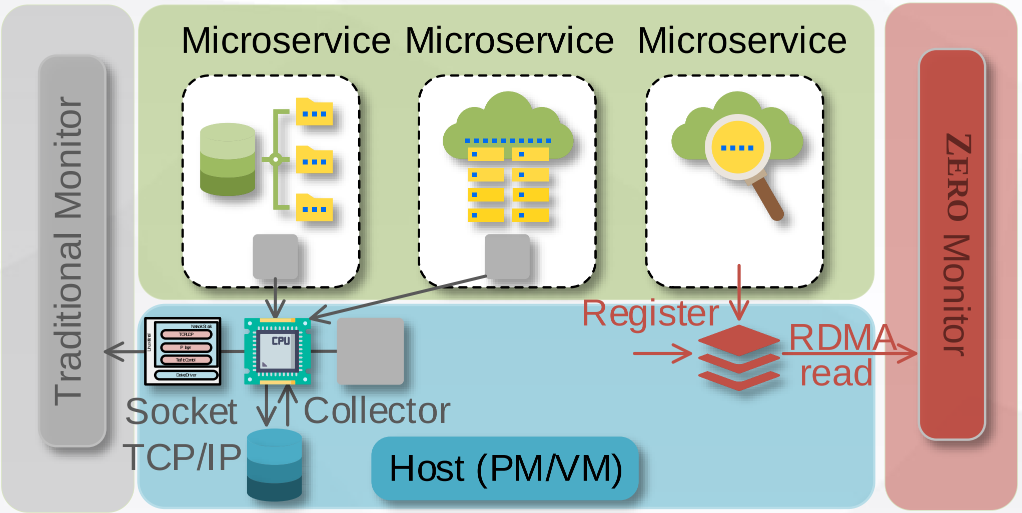
- ✓ High throughput & low CPU utilization
- ✓ Avoid incast & PFC/ECN triggering
- ✓ Stable QoS

Contents

Conclusion

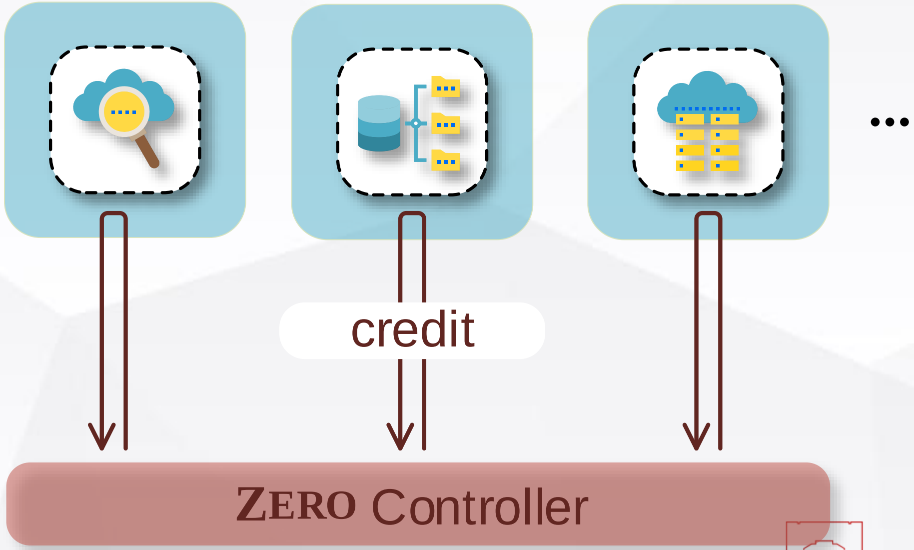
Zero-overhead monitoring

- ✓ One-sided RDMA (RDMA read)
- ✓ Novel control & data plane design



Large-scale distributed monitoring

- ✓ Receiver-driven CC
- ✓ Highly-efficient thread and I/O model



Thanks! Q&A

Contacts:

linghe.kong@sjtu.edu.cn

songzhuo.sz@alibaba-inc.com

sima.mt@alibaba-inc.com

wang-zhe@sjtu.edu.cn

Experience & Future works

Achieving high scalability and availability

- ✓ QP sharing & group switching, standby controller

Avoiding network interference

- × Physical isolation (high cost)，low priority (timeout)
- ✓ Control build-up queue (receiver-driven)

Receiver-driven CC

- × Equal bandwidth sharing, rely on ECN or INT
- ✓ Credit only (<BDP), pacing is required, general case?