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
Cloud-native computing

Cloud-native infrastructure:
- Monolithic design --- microservices
- Dense deployment
- Disposable and immutable system
- Various applications

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
Decoupling monitor from infrastructure!

Zero-overhead monitoring

Metric features
 ✓ Counters & reproducible calculation

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

How to decouple?

Register
RDMA read

Socket
TCP/IP
Collector

Host (PM/VM)
**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
Control Plane

Microservices

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

Universal interface

Disposable overhead

- System/persistent/tidal metrics
- Serverless functions

QP connection share

- Manage & share QP connection

Zero Agent

- Zero Copy

Host (PM/VM)

- Zero Controller

Processing Thread

- Persistence + Visualization

Collecting Thread

- Credit-FC + Hybrid I/O

Universal interface

Disposable overhead

QP connection share
Data Plane

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

Shared memory

Memory management

Shared memory
Efficient threading and I/O model
Avoid incast with many connections
Guaranteed QoS level via receiver-driven model
1. Background & Motivation
2. Core Design
3. Implementation & Evaluation
4. Conclusion
Implementation

✓ Zero framework
✓ Case studies

Evaluation Setup

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

```c
// 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));
```
Zero Overhead

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

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

✓ High throughput & low CPU utilization
✓ Avoid incast & PFC/ECN triggering
✓ Stable QoS
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
simamt@alibaba-inc.com
wang-zhe@sjtu.edu.cn
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?