字节跳动容器化场景下的性能优化实践

江帆

字节跳动 工程师

极客邦科技
会议推荐2019

ArchSummit 深圳
全球架构师峰会
大会：7月12-13日
培训：7月14-15日

ArchSummit 北京
全球架构师峰会
大会：12月6-7日
培训：12月8-9日

5月　6月　7月　10月　11月　12月

QCon 北京
全球软件开发大会
大会：5月6-8日
培训：5月9-10日

QCon 广州
全球软件开发大会
培训：5月25-26日
大会：5月27-28日

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14-15日

GMTC 北京
全球大前端技术大会
大会：6月20-21日
培训：6月22-23日

QCon 上海
全球软件开发大会
大会：10月17-19日
培训：10月20-21日

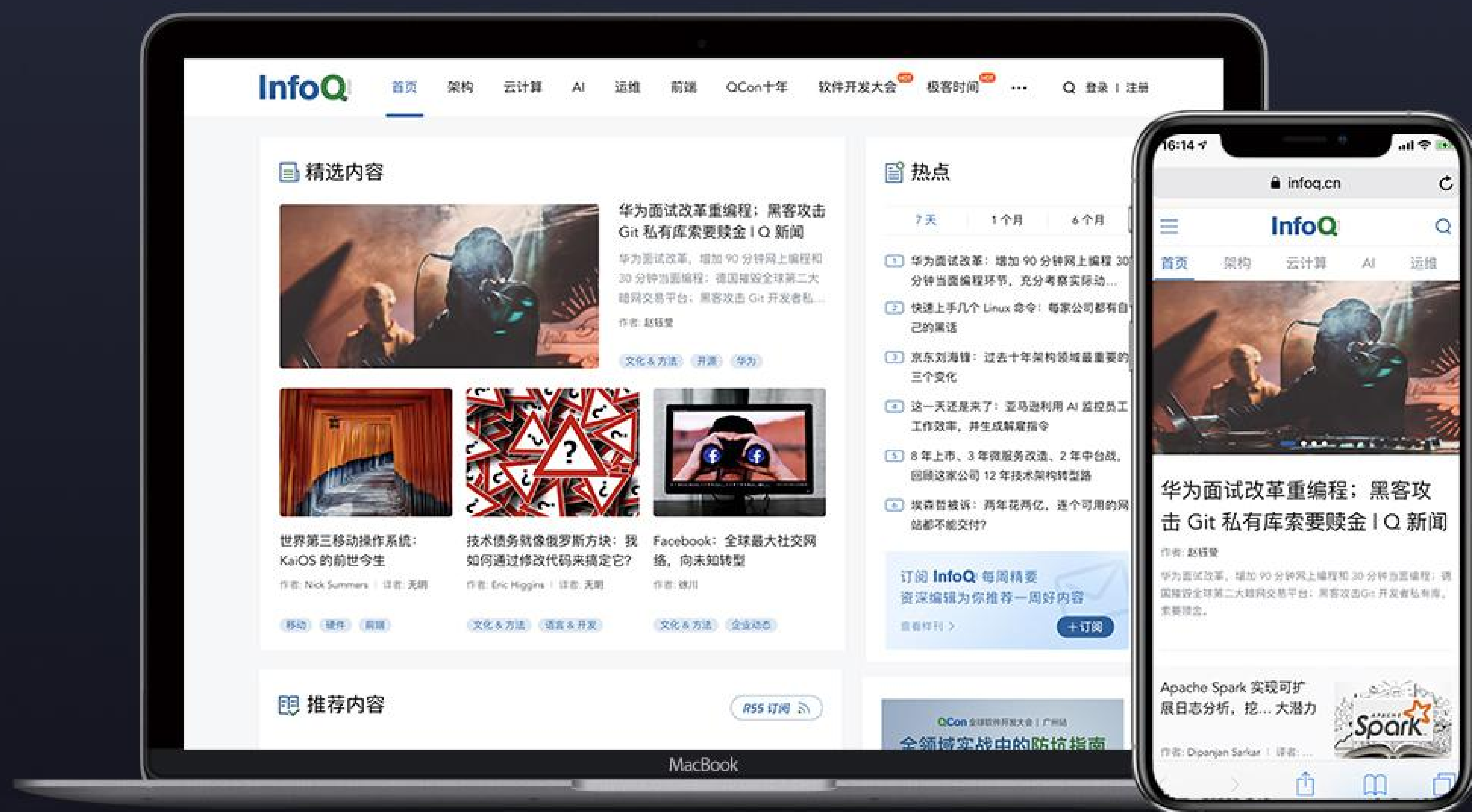
GMTC 深圳
全球大前端技术大会
大会：11月8-9日
培训：11月10-11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21-22日
培训：11月23-24日

Geekbang 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播



关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货

About The

SPEAKER

江 帆 工程师

- TCE 是字节跳动的私有云平台，管理着业界规模领先的 Kubernetes 集群，托管了头条、抖音、字节国际化业务等内部上万个在线微服务。作为早期成员，参与了 TCE 的研发工作，拥有大规模 Kubernetes 集群的开发和维护经验，熟悉由 Kubernetes 到 Docker 再到 Cgroups 的整个核心链路。目前正在参与在线、离线大规模混合部署项目，预期实现集群资源利用率的进一步提升。

TABLE OF CONTENTS 大纲

- 背景介绍
- 基于 eBPF 的系统监控，提升系统可见性
- 合理提升资源利用率
- Cgroups 调优，保证延时敏感服务 QoS

背景

- TCE, 字节跳动私有云平台
- Kubernetes, Docker, 规模不断增大，负载越来越高
- Cgroups 隔离性、系统监控可观测性不足，线上运维、故障排查效率低，阻碍资源优化的推进
- 资源利用率低，成本问题

☞ 提升可见性，了解服务之间的相互影响

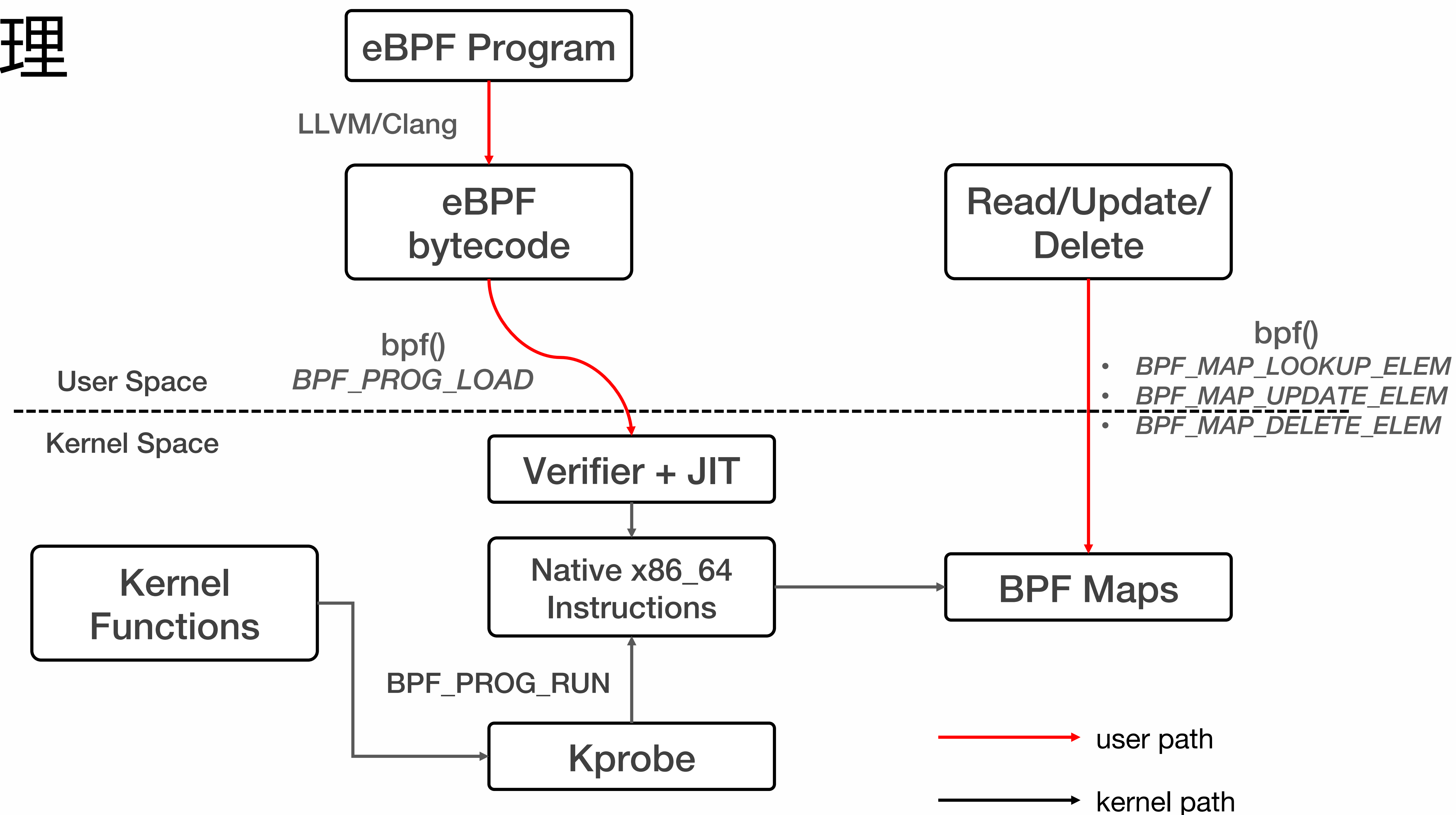
-> 提升问题诊断效率和利用率

传统系统监控的弊端

- 常用系统监控: cAdvisor, Atop, Ganglia, OpenFalcon
  - 只能看到 Kernel 暴露的数据
- 归结为 Kernel 对微服务理解还不够(隔离、监控两个层面)
  - 大而复杂的 codebase, 难于修改，升级很痛苦
  - Cgroups 隔离和监控不够完善，noise neighbor
  - 不知道 Container ID 和 PodName
- *Borg, Omega, and Kubernetes*
  - APPLICATION-ORIENTED INFRASTRUCTURE
  - 需要应用级别监控
- 引入 eBPF，让 Kernel 更懂容器
  - In-kernel virtual machine
  - JIT, native code 执行速度
  - 支持 Trace 系统，几乎可以在任何地方执行，极其灵活

什么是 eBPF？

- eBPF(Extended BPF) 组成和基本原理
  - bpf syscall(multiplexor)
  - eBPF bytecode, llvm
  - Verifier: length, dag, malformed jump
  - Interpreter/JIT
  - 多种 Map
  - 和 Trace 系统(Kprobe)结合

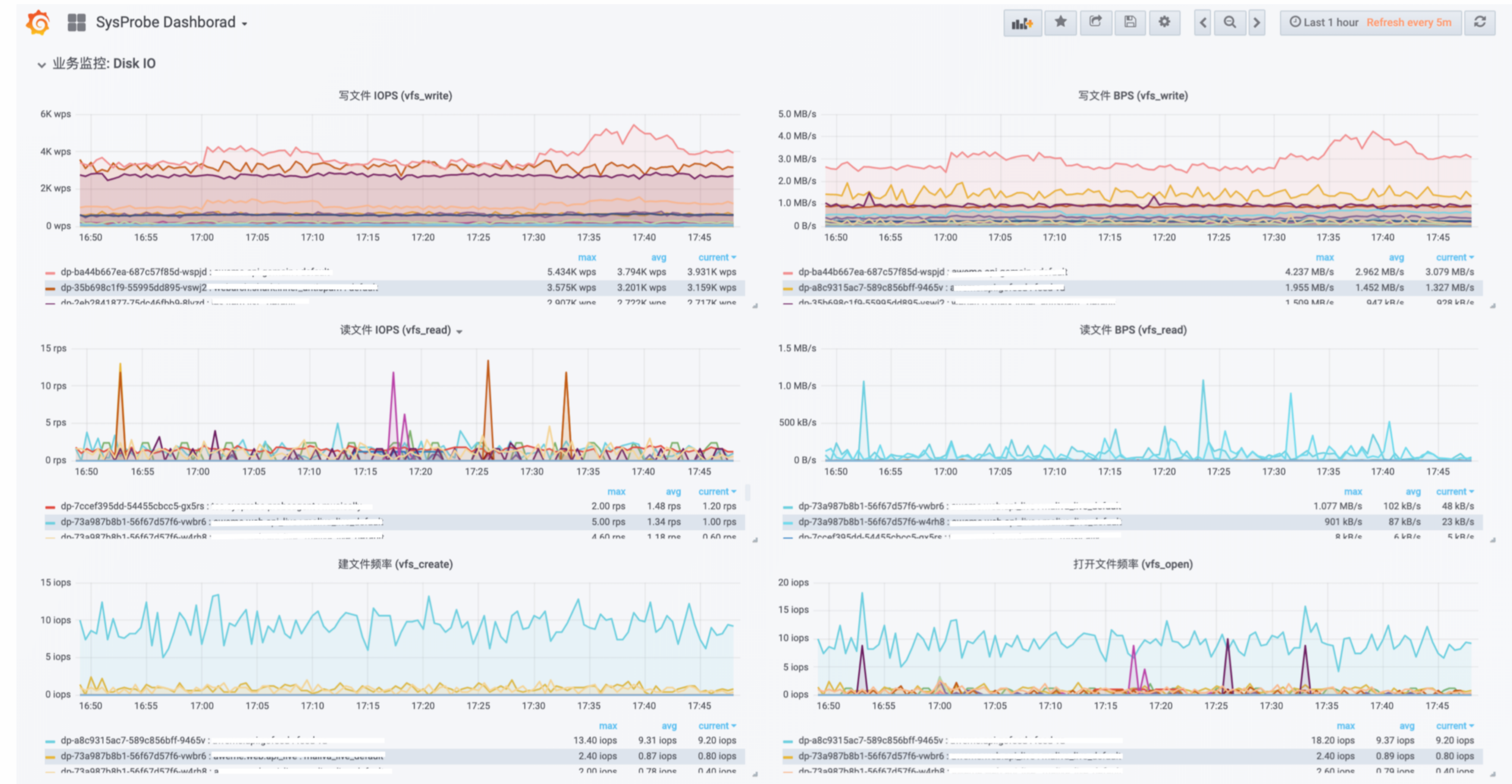


SysProbe, 基于 eBPF 的系统监控

- CPU & Memory，使用 Cgroups 数据，增加 throttle、组线程数和状态、实例 Load
- Block IO，VFS 层，识别实例实际读写行为
- Network Bandwidth & SLA
  - 实例级别 Host 网络带宽、PPS、Socket 状态监控
  - 重传、SRTT 抖动
- Task ID -> Container ID -> PodName, Aggregation
  - 更细粒度、关键的实例级别 Metrics
- 核心思想: 准确识别用户程序在各资源维度的具体行为

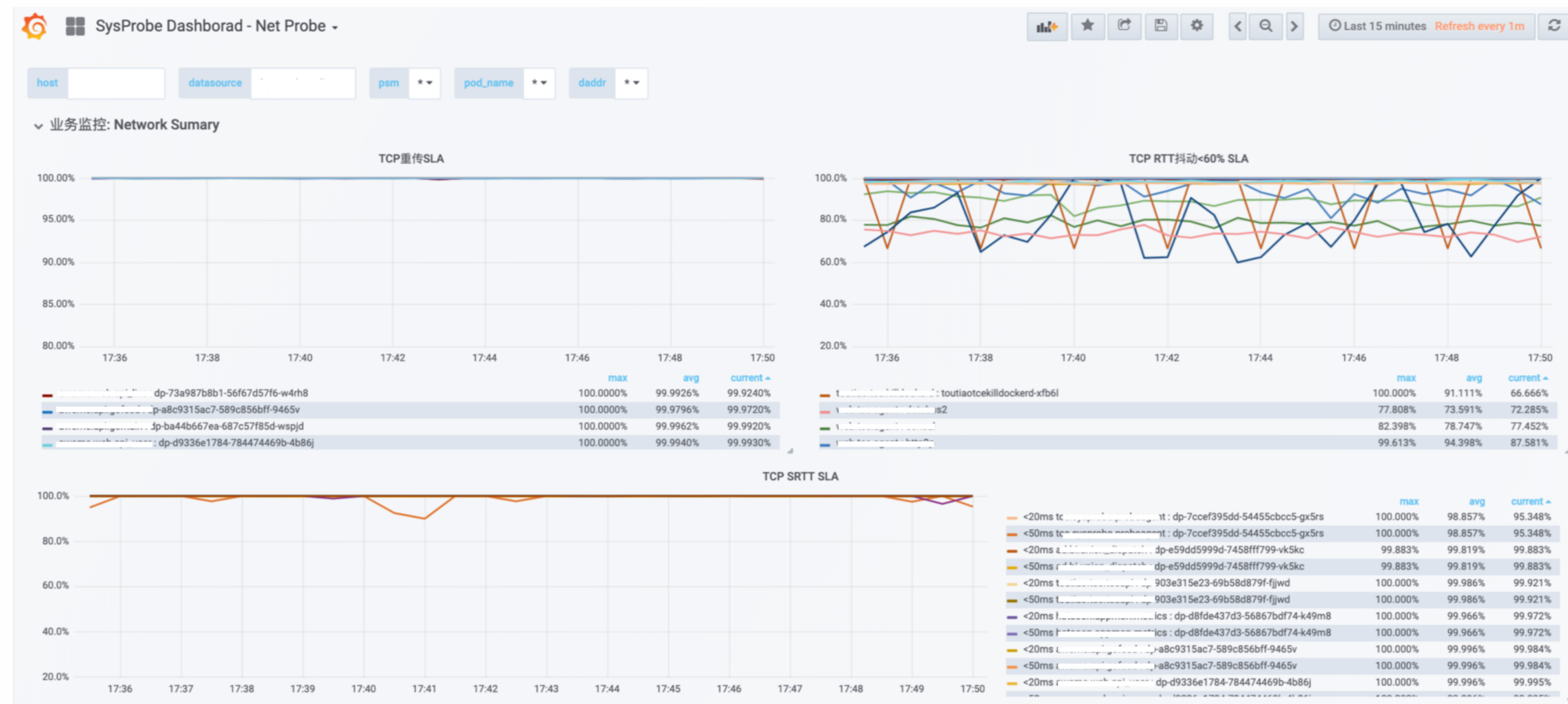
案例: Block IO

- 各种 VFS 层程序行为监控
- 真实 case: 整点抖动问题
  - 多个服务同时在整点出现调用下游超时错误飙升；
  - 从 SysProbe 看到多个业务实例整点调用 fsync （业务程序行为）；
  - 原因: 日志整点切分相当于一个 barrier，导致多个服务同时刷盘。由于 TCE 的 IO 未隔离和混部环境，同 host 多实例之间相互增益，下游抖动对上游的增益，因此出现了多个服务出现显著的整点抖动现象。



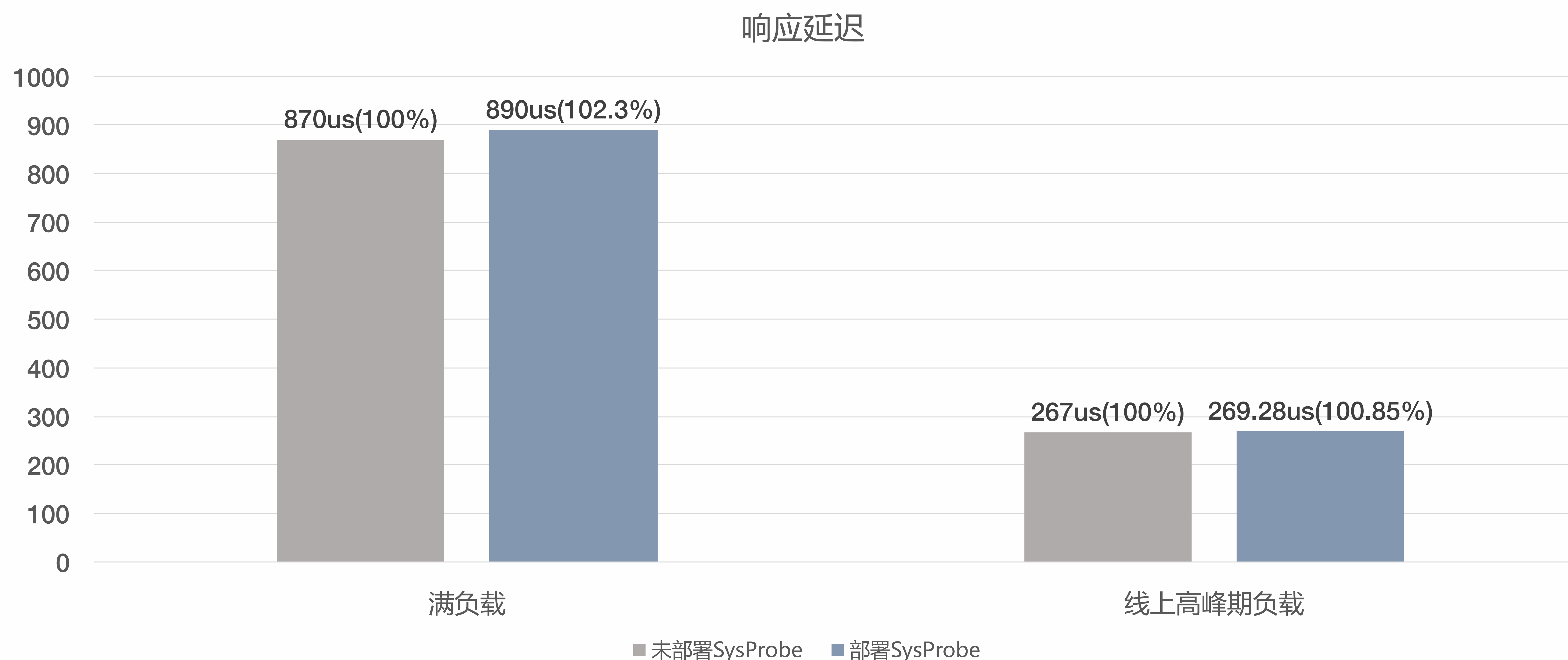
案例: Network SLA

- SLA: 重传 & RTT 抖动
  - byteping: IDC 级别网络 SLA 监控
  - 实例 Socket 级别 SLA，实时准确的 e2e 性能监控



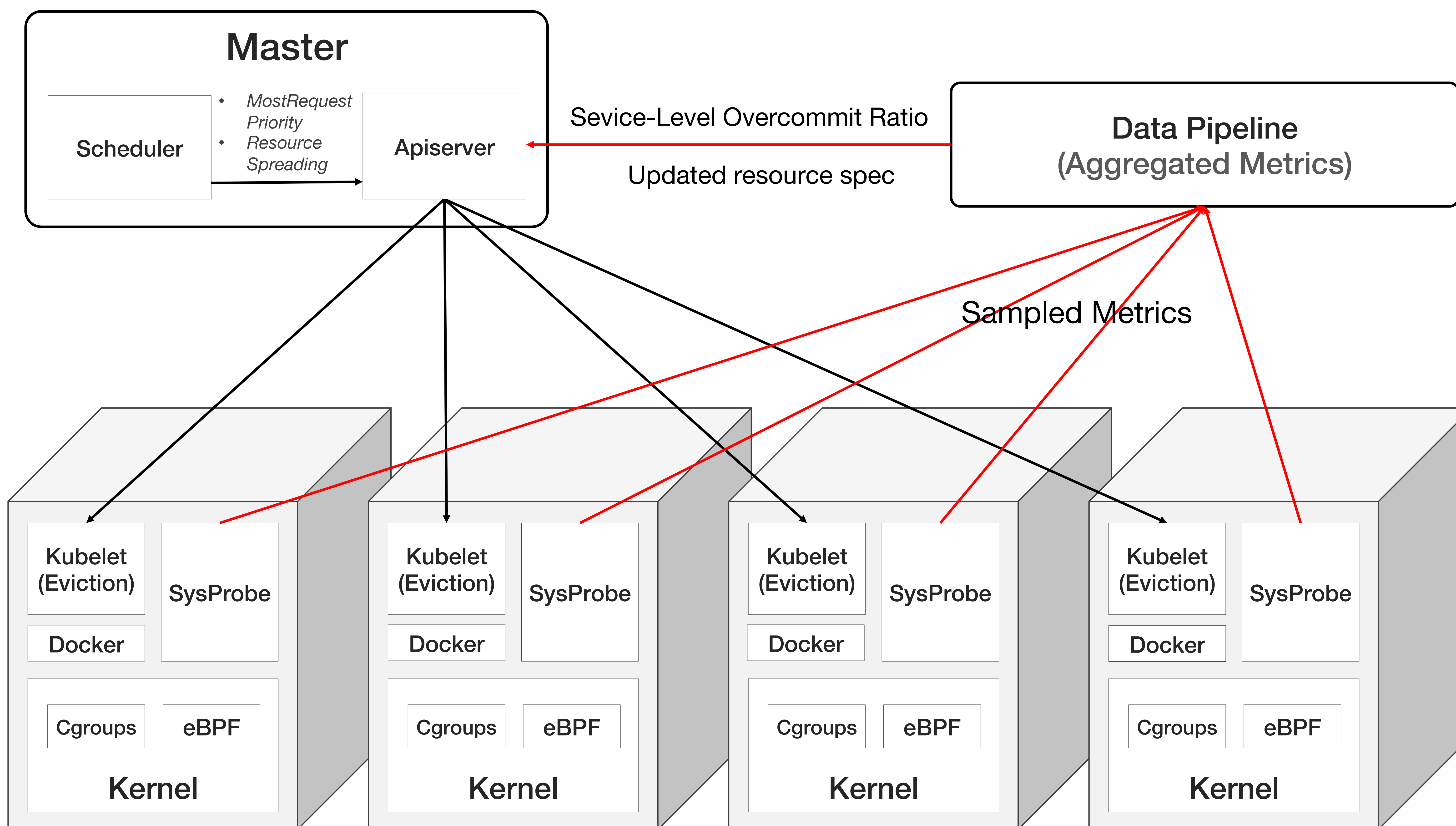
eBPF Benchmark

- 测试场景为两台同 TOR 下的主机，测试应用为 wrk 和 nginx，网络 IO 密集型应用，比较极端的小包场景，代表网络部分的性能，常规 rpc 应用的性能损耗会大大小于测试结果。



动态超售，合理提升资源利用率

- 准确的数据源，有效识别workload，避免资源竞争
  - SysProbe
  - CPU, Mem, Net IO, Block IO
- 服务级别动态超售
  - realtime update + long term data
  - MostRequestedPriority
  - Resource Spreading
  - Auto Trigger for cold service
  - Enhanced Eviction



过载保护

- 优化 Eviction, 保证在关键资源上不出现饿死
  - CPU, Memory, Disk Space
  - CPU: 基于实例 Load 识别 noise neighbor, 对于在线业务尤其有效
  - 基于 Evict API 进行驱逐，受 PDB 约束

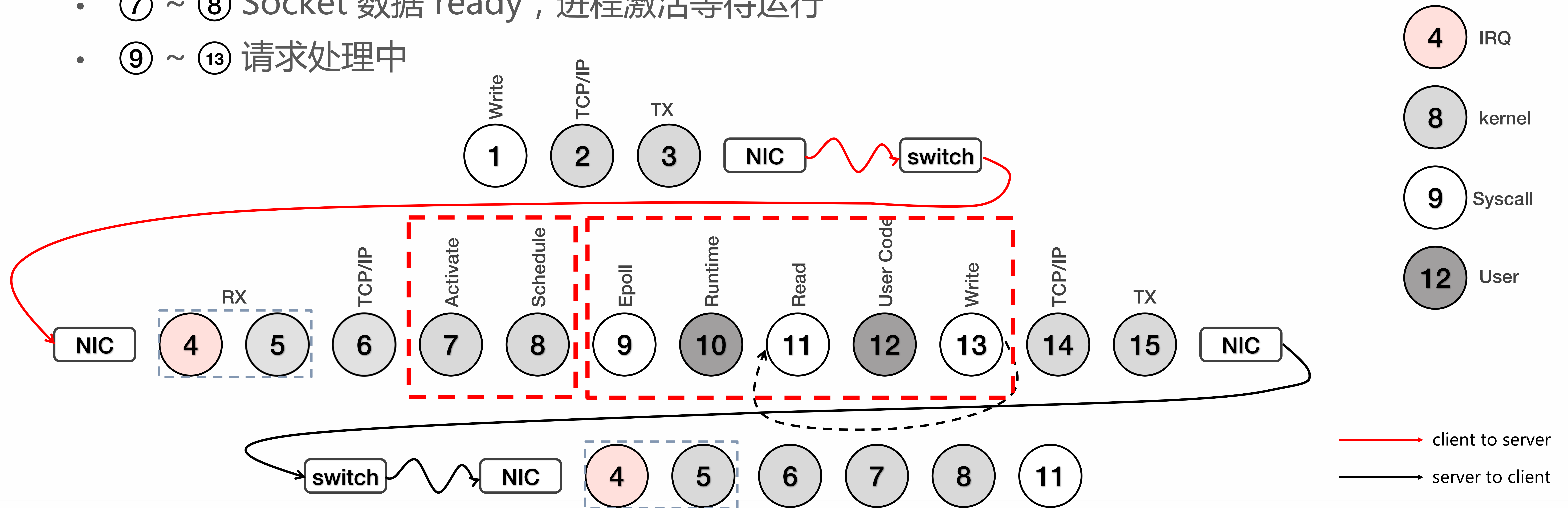
☞ 但只有控制面的驱逐还不够

为何要从内核层面看 pct99 抖动问题?

- 为什么要关注 pct99？
  - *The Tail at Scale, by Jeffrey Dean and Luiz André Barroso*
- 阻碍容器化进程
  - 业务物理机独占部署没问题，上容器 pct99 抖动严重
  - 有时机器负载不高，但业务 pct99 仍然很高
  - 容器控制面无法解释这些问题

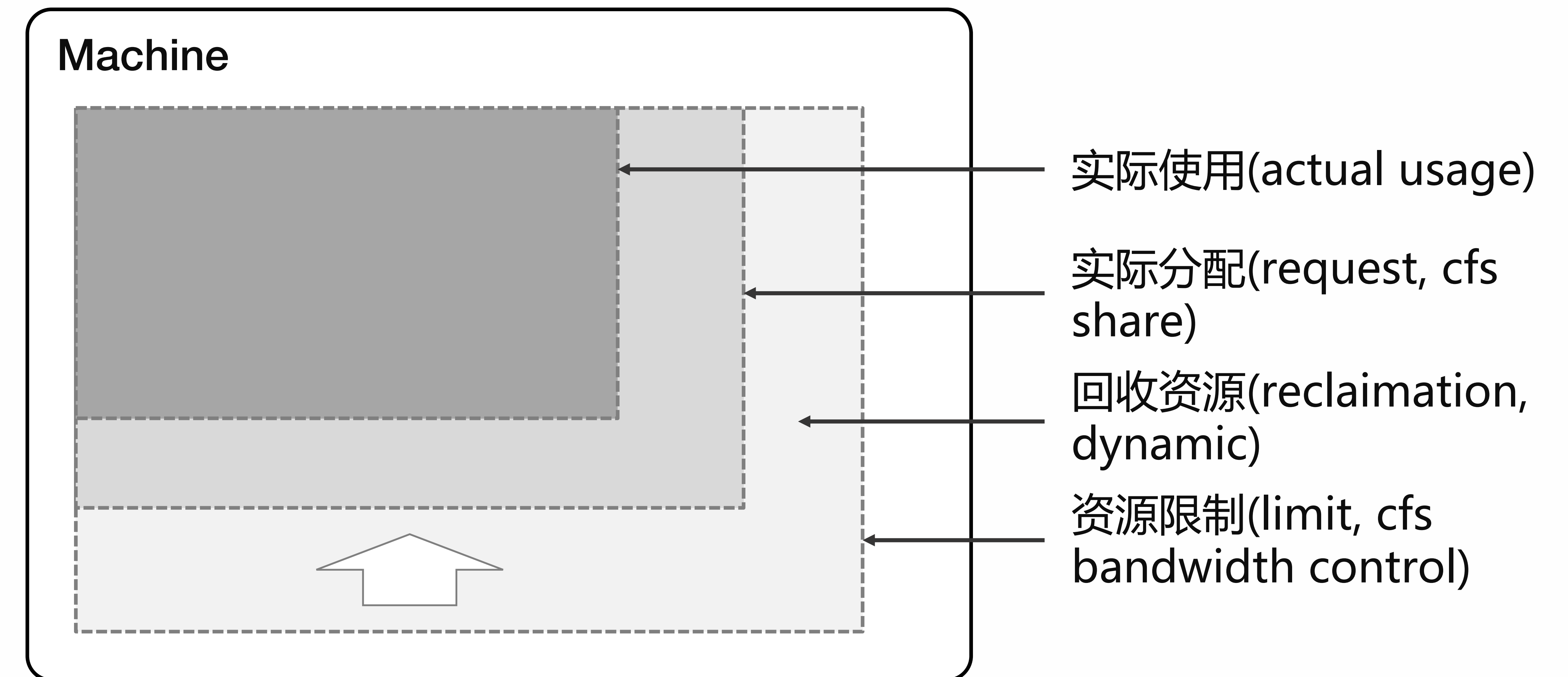
一个 RPC 请求的生命周期

- 典型表现: server 延时低，但 client 端超时(client 和 server 是相对概念)
- 产生延迟的两个阶段
  - ⑦ ~ ⑧ Socket 数据 ready，进程激活等待运行
  - ⑨ ~ ⑬ 请求处理中



Kubernetes 资源模型和 QoS 级别

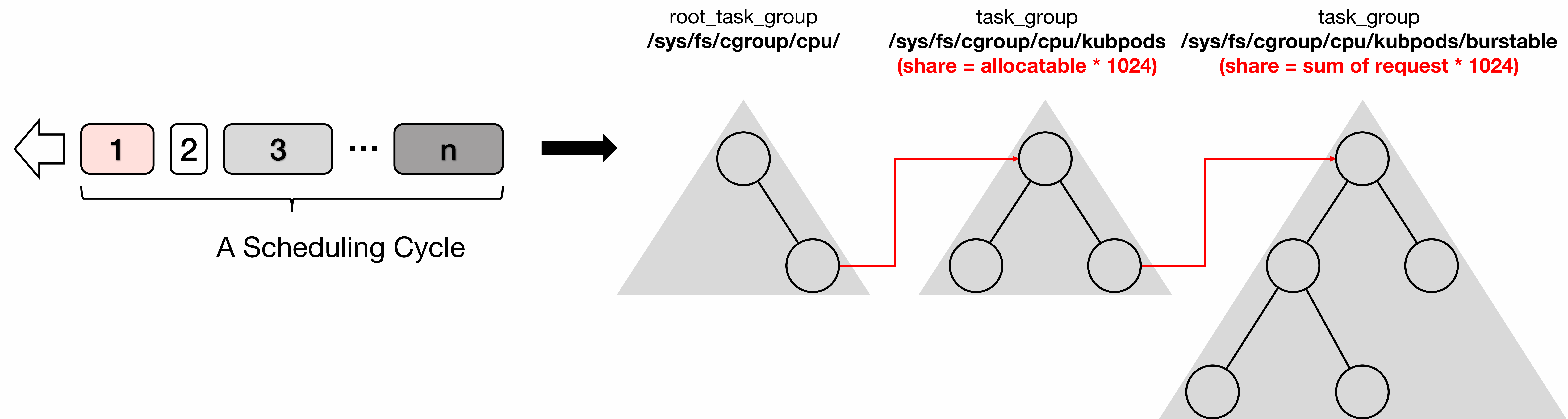
- 资源模型
  - Request & Limit
- CPU QoS 分级
  - Guaranteed: **/kubepods/podxxx**
  - Burstable: **/kubepods/burstable/podxxx**
  - Besteffort: **/kubepods/besteffort/podxxx**



- 超售的控制面表现
  - [node allocatable] = [node capacity] - [kube-reserved] - [system-reserved]
  - [sum of pod's request] < [node allocatable] < [sum of pod's limit]
  - 为什么一定要保证超售后 request > 实际使用量？

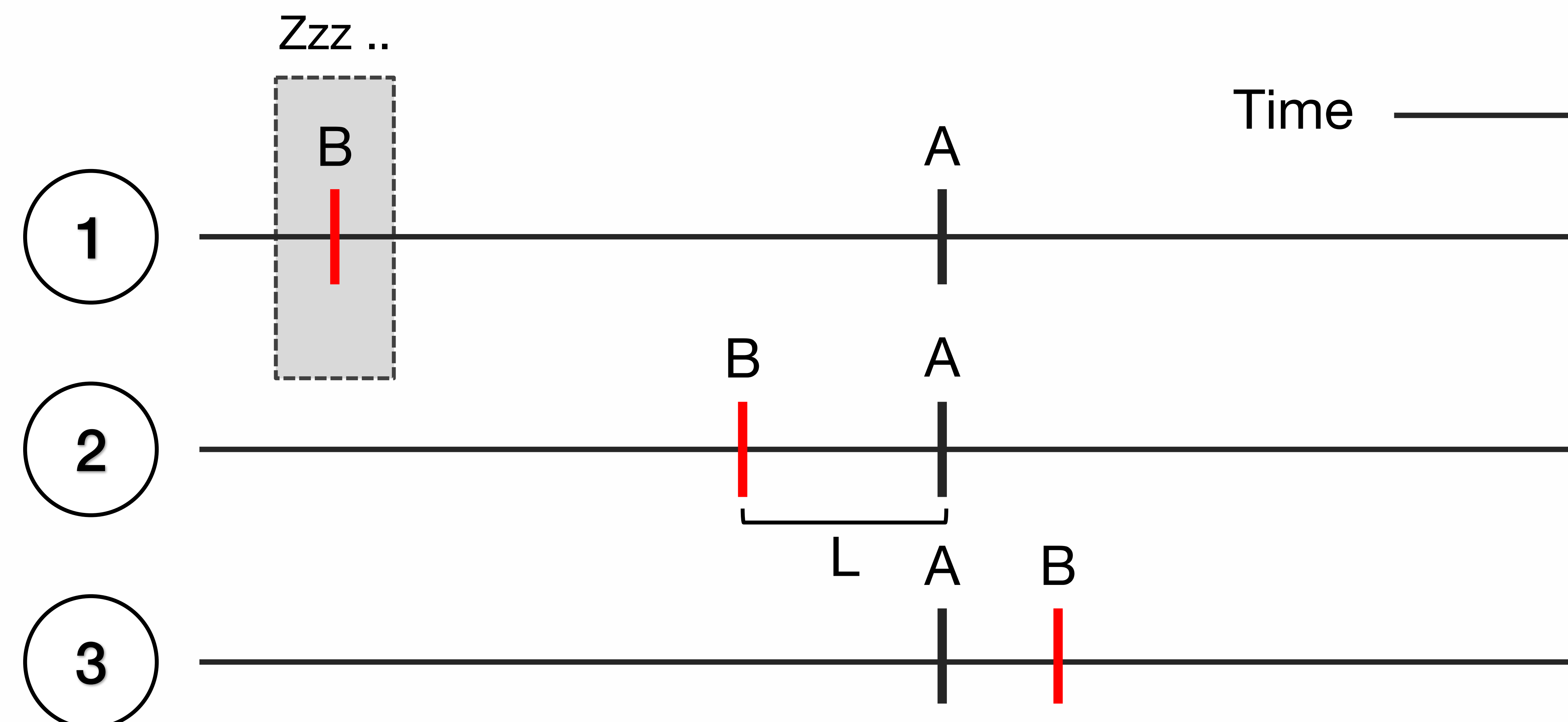
CPU QoS 的含义

- CFS 基本原理和组调度
  - 每个组，即 task_group，在每个核的队列上都有一个自己的 sched_entity，包含一个 rbtree；
  - 多核 CPU 下，每个组的在每个核上的 share 为 cpu cgroup 中设置的 share 值；
- QoS 体现在 share 所占比例上，并且逐层递归



CFS 调度延迟分析

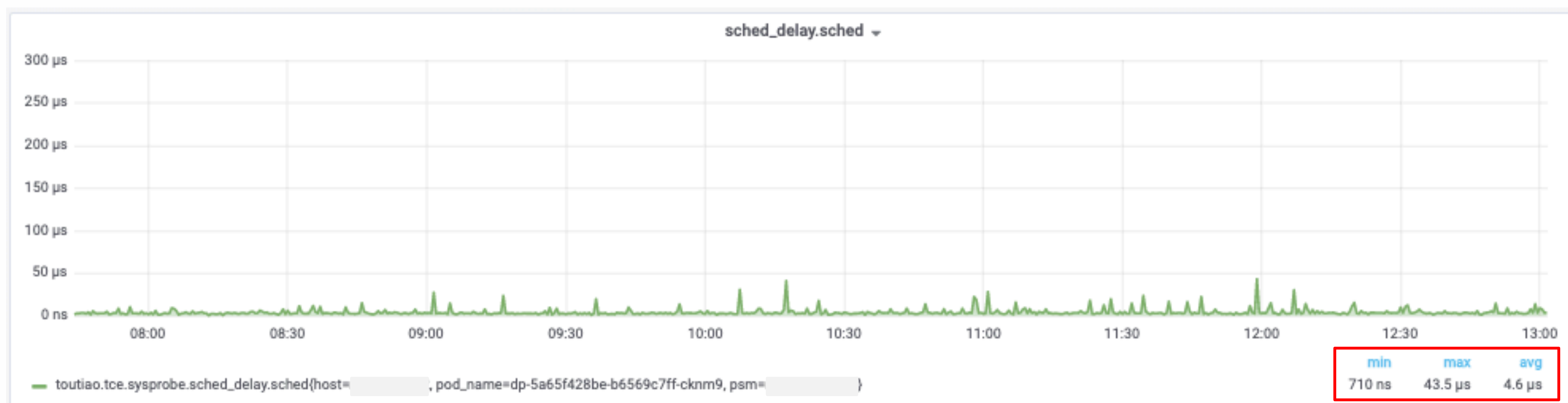
- CFS 调度延迟分析
  - 超售导致 Share 变小，share = user_demand / overcommit_ratio * 1024
  - CFS Wakeup Preemption: vruntime(T) = max(vruntime(T), min(vruntime(*)) – L)
    - L(called “ thresh” in kernel) = 24ms/2 = 12ms
- Request 和 Share 计算解耦
  - share 在 request 基础上给予一定 burst ratio
  - 针对个别延迟极敏感服务



CFS Wakeup Preemption 过程描述：1. 初始 B 处于 Sleeping 状态而 A 为 Running；2. 当 B 唤醒，其 vruntime 被置为小于 A 的 runtime – L；当 B 开始运行，就给 A 带来了调度延迟。

CFS 调度延迟分析

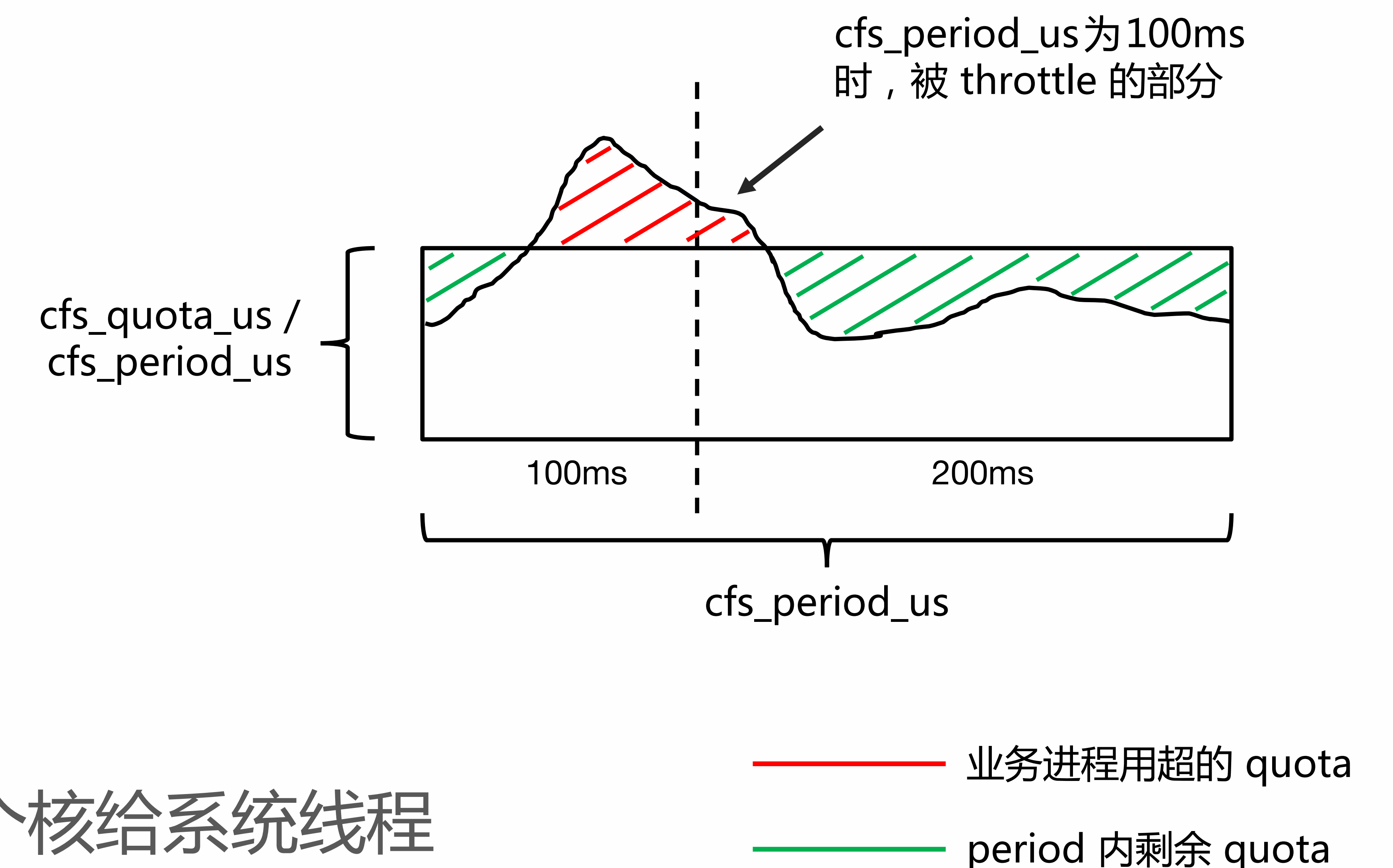
- 基于 eBPF 进行延迟拆解分析
  - 主要阶段
    - 网卡收包、协议栈、进程唤醒等待执行、处理请求、返回响应
  - 关键函数
    - netif_receive_skb_internal
    - ip_rcv
    - tcp_rcv_established
    - tcp_recvmsg



Socket 数据 ready 到进程开始执行的延迟

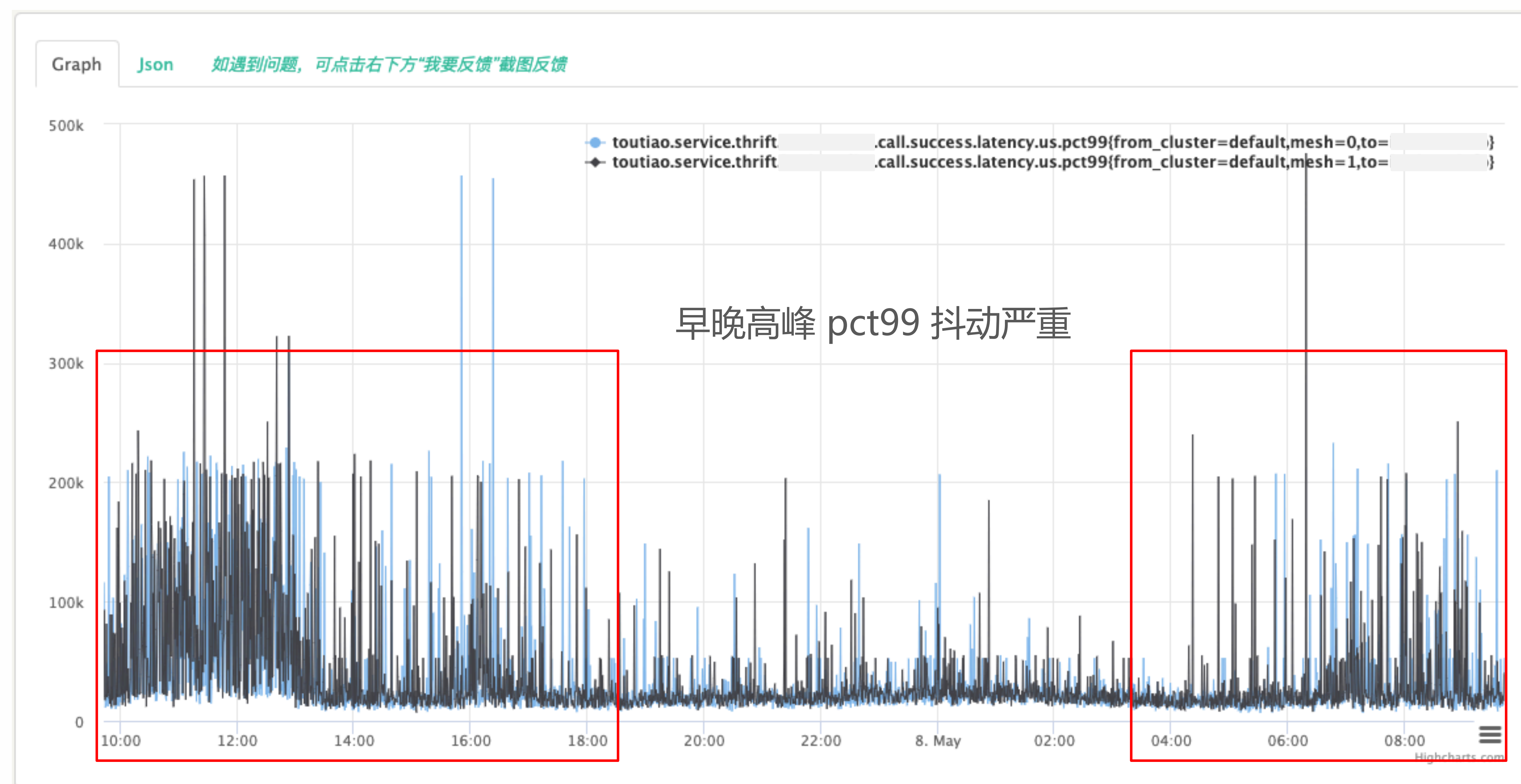
CFS 带宽限制和调优方法

- Share 保证下限，带宽限制确保不超上限
- 两种调节技术
  - 纵向，比值增大: 调大 cfs_quota_us，cfs_quota_us/cfs_period_us
  - 横向，比值不变: 调大 cfs_period_us, 保持 cfs_quota_us/cfs_period_us
    - 关心被 trottle 的次数而不是被挂起的时间
    - 横向调整，效果更佳
- 结合业务层面
  - 为什么 golang 程序问题较多？
    - 异步超时逻辑，旁路 timer 响应比处理任务的协程快
    - GC、协程调度算法 work steal，短时间大量并发线程
    - export GOMAXPROCS=(($MY_CPU_LIMIT - 1))，预留一个核给系统线程
  - uWSGI + nginx: lxcfs 给出 virtual 的 /proc 和 /sys 视图



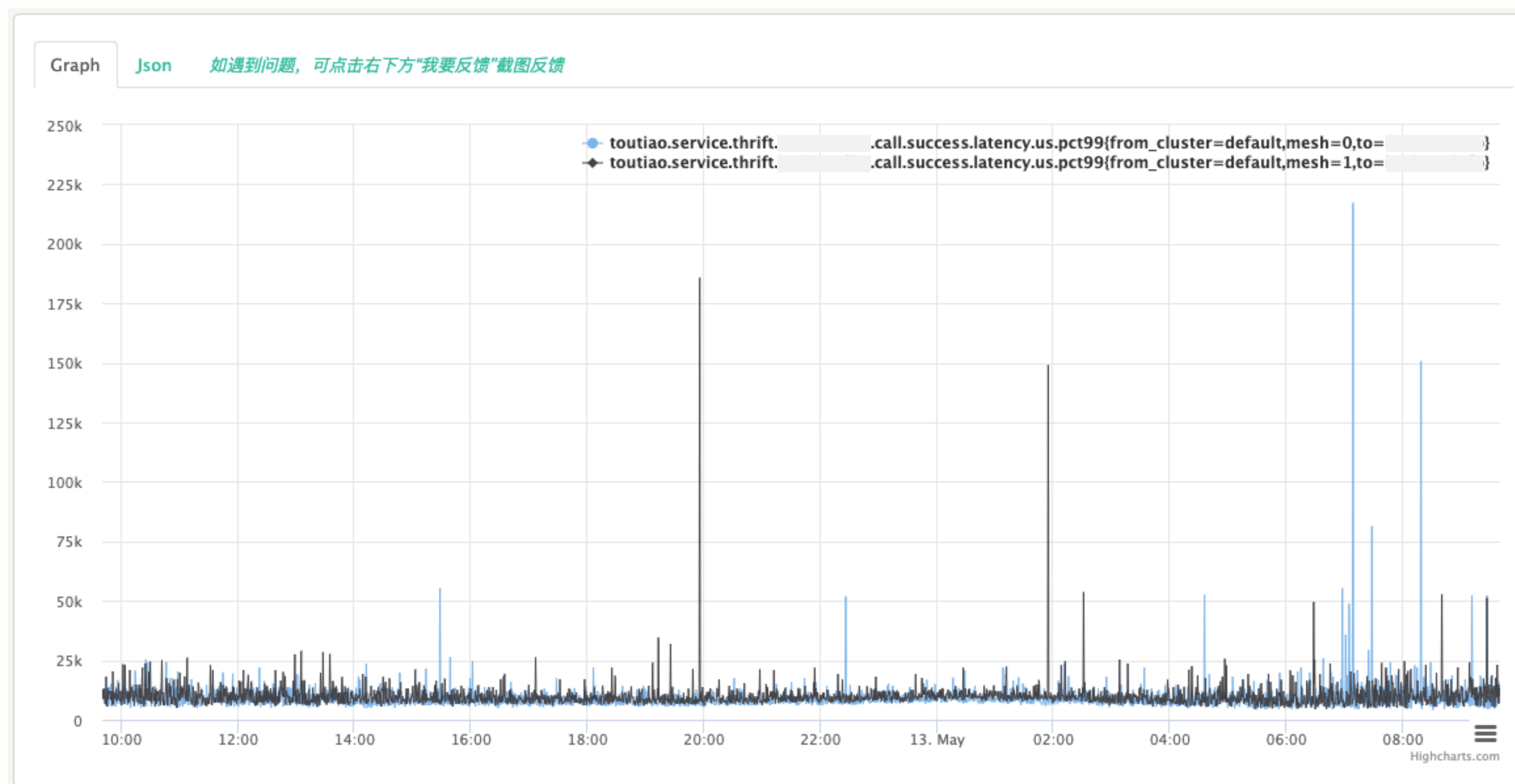
线上服务调优效果(一)

- 服务1: 开启 Service Mesh(线程数增多)，pct99 抖动加剧



线上服务调优效果(一)

- 服务1: 多个小套餐实例合并为一个大实例(调大 Share)，利用率不变，比未开启 Mesh 前性能更好

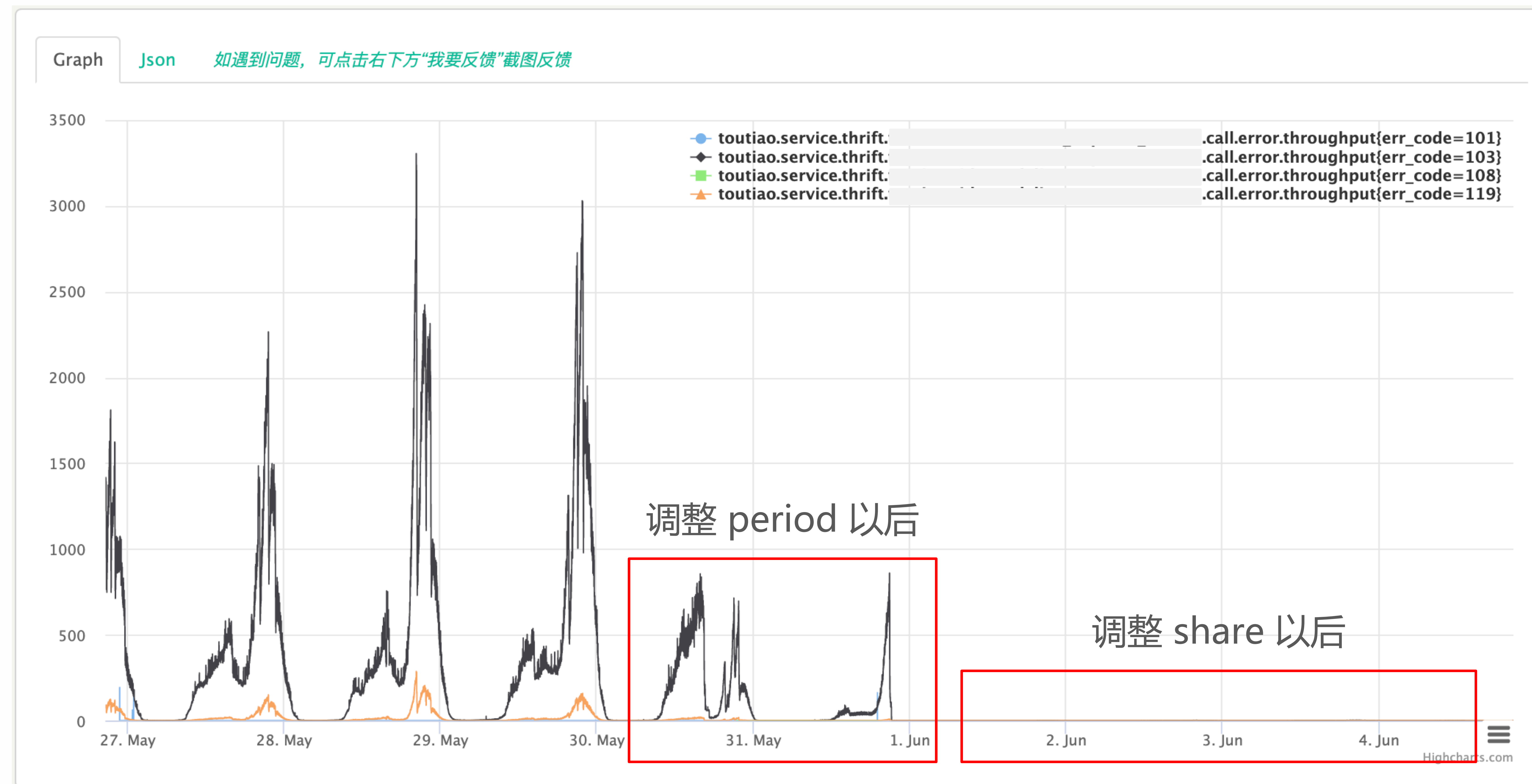


线上服务调优效果(二)

- 服务2: 调大 Period + export GOMAXPROCS=(($MY_CPU_LIMIT - 1))，share 保持不变

线上服务调优效果(三)

- 服务3: 调大 Share + 调大 Period + export GOMAXPROCS=(($MY_CPU_LIMIT - 1))



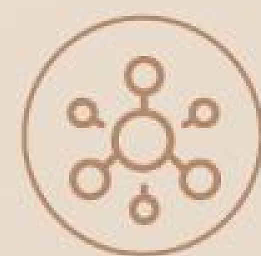
极客时间

69 节高清视频公开课

来自 Google、微软、Facebook、BAT 等一线大厂大咖倾心分享

分享实战经验
一线大厂技术选型的遗憾和经验教训

新锐观点碰撞
人工智能、大数据、微服务、Go、Java、Python等技术解析



实用进阶建议
成为“高薪”程序员需要哪些“软实力”？

亲授面试技巧
大厂面试官面试时看重哪些能力？

扫码立即参与
(限时24小时)

* 附赠：100 本架构师电子书

TGO鲲鹏会

汇聚全球科技领导者的高端社群

全球12大城市

850+高端科技领导者

使命 Mission

为社会输送更多优秀的科技领导者

愿景 Vision

构建全球领先的有技术背景优秀人才的学习成长平台

扫描二维码，了解更多内容

THANKS

Geekbang. 极客邦科技 InfoQ