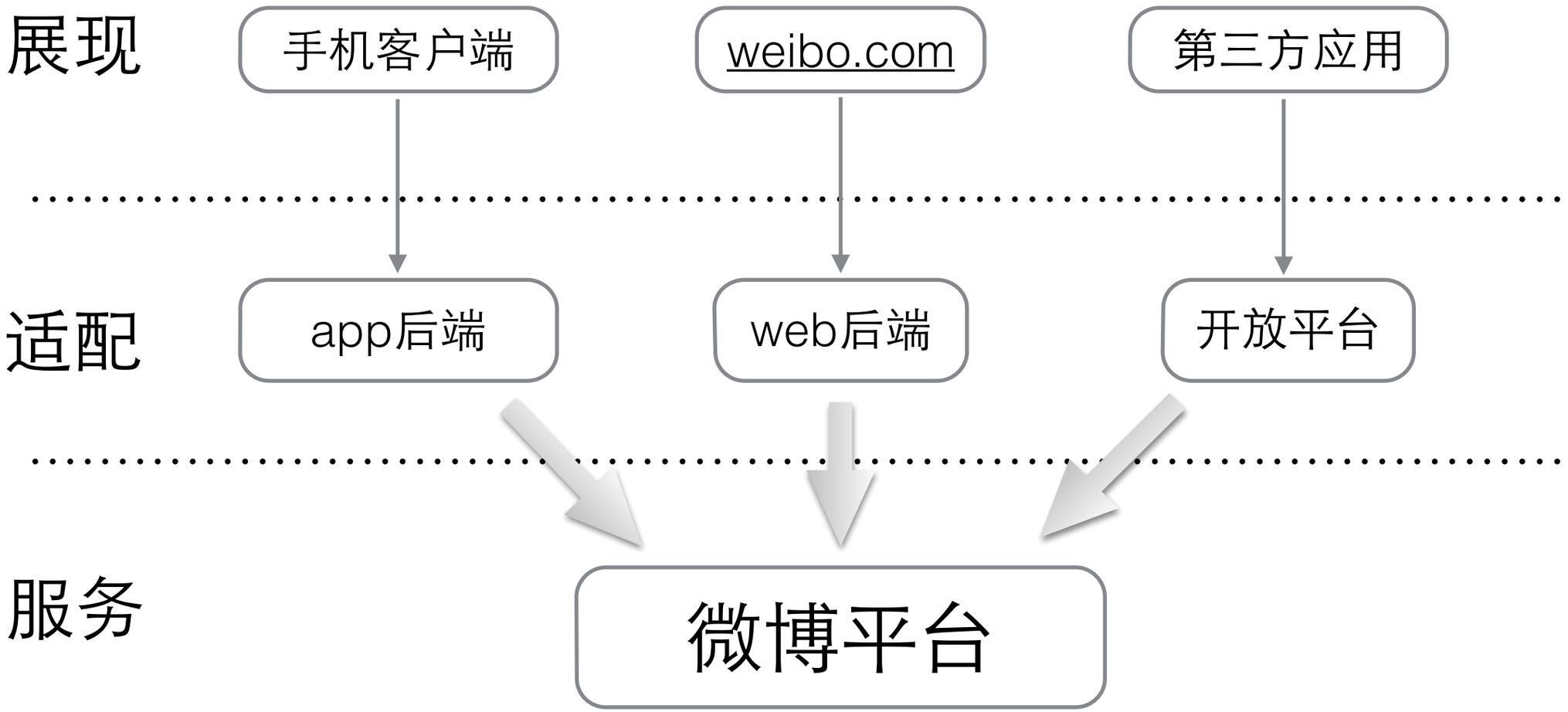
微博在大规模、高负载系统中的典型问题

新浪微博 平台及大数据部
秦迪 @蛋疼的axb

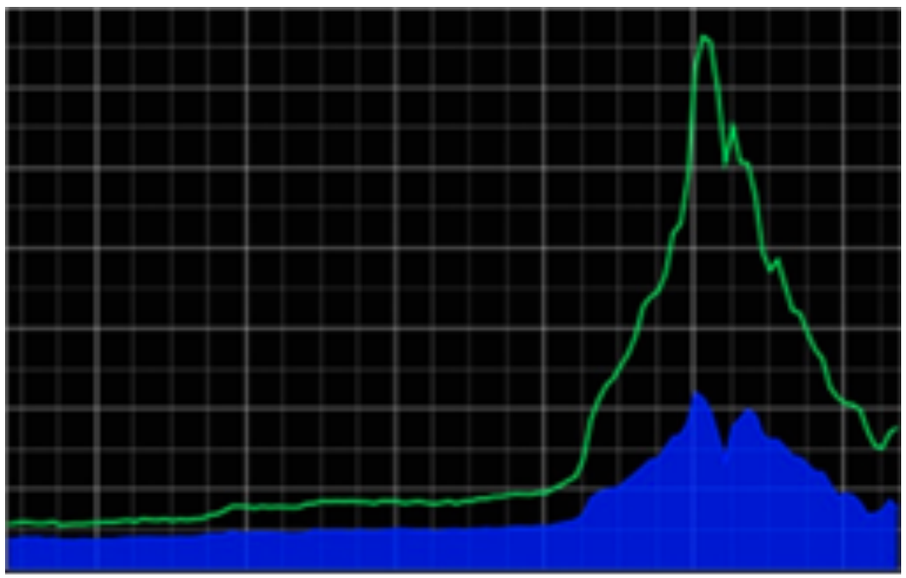
- 有哪些问题
- 如何排查
- 如何预防

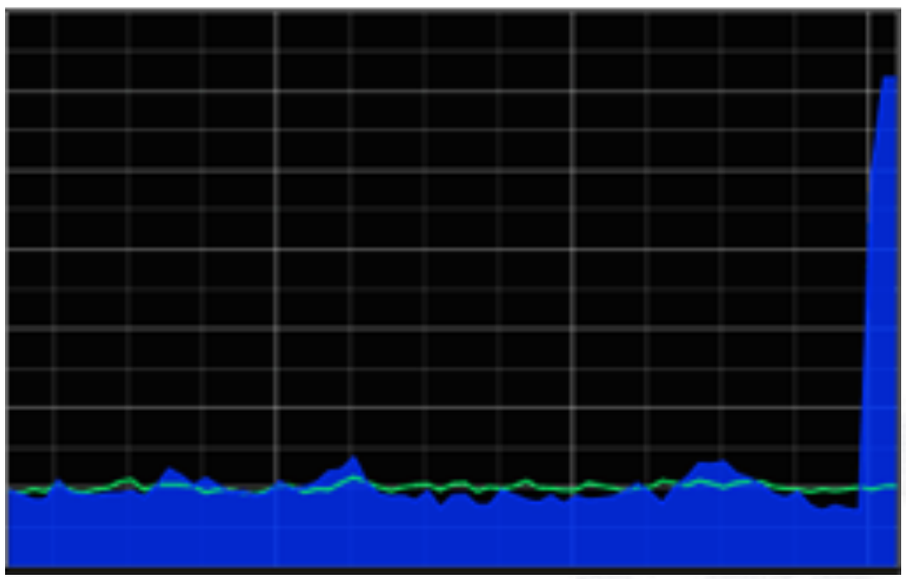
关于微博和微博平台



关于大规模、高负载

- 一些数据
  - 8亿注册用户
  - 8000万+DAU
  - 1.75亿MAU
- 典型场景
  - 固定活动
    - 吐槽春晚
    - 让红包飞
  - 突发事件
    - #马航370#
    - #周一见#
    - 《我错了》





关于问题

功能问题

- 发不出微博
- 未读数不准
- 500/502/503
- ……

关于问题

- 刷微博慢
- 提醒延迟
- 接口响应时间长
- ……

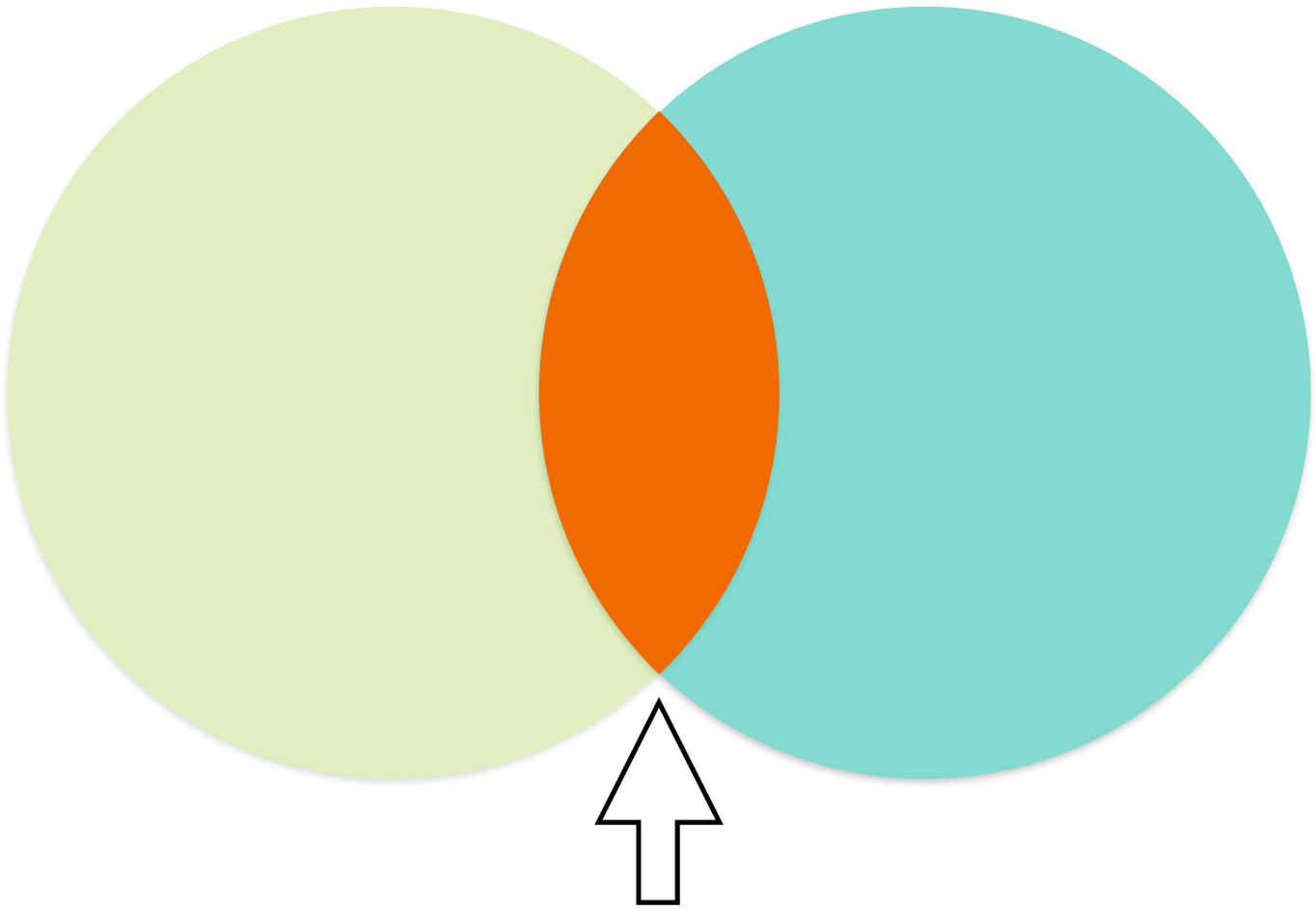
性能问题

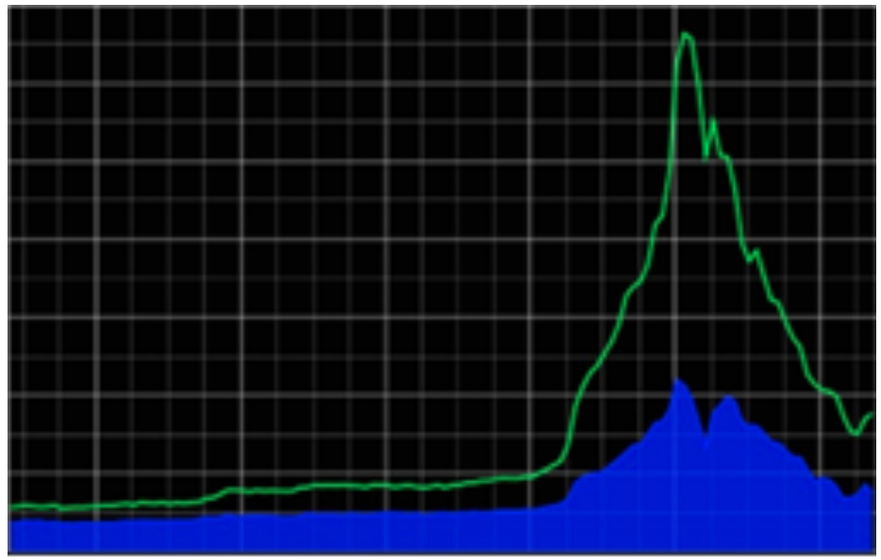
关于问题

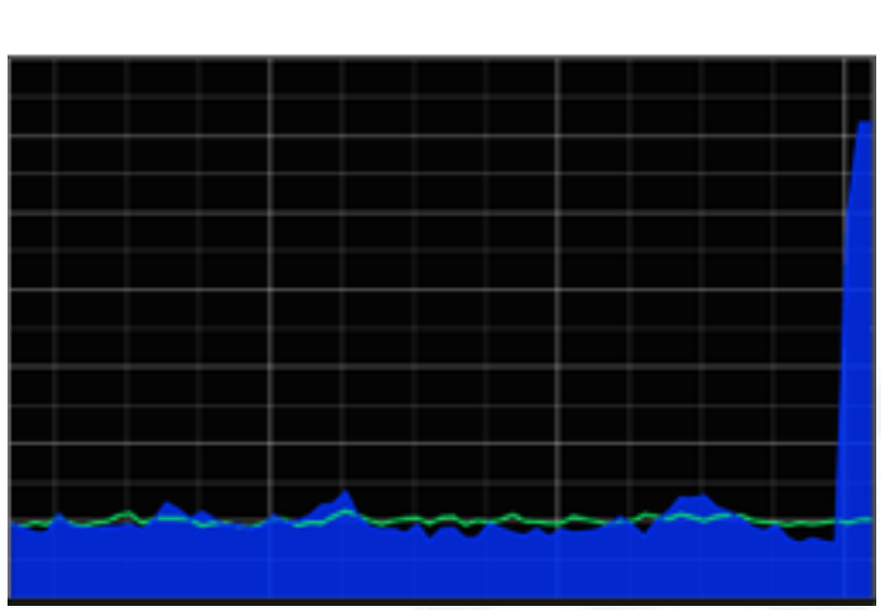
功能问题

性能问题

面对极端流量时的问题

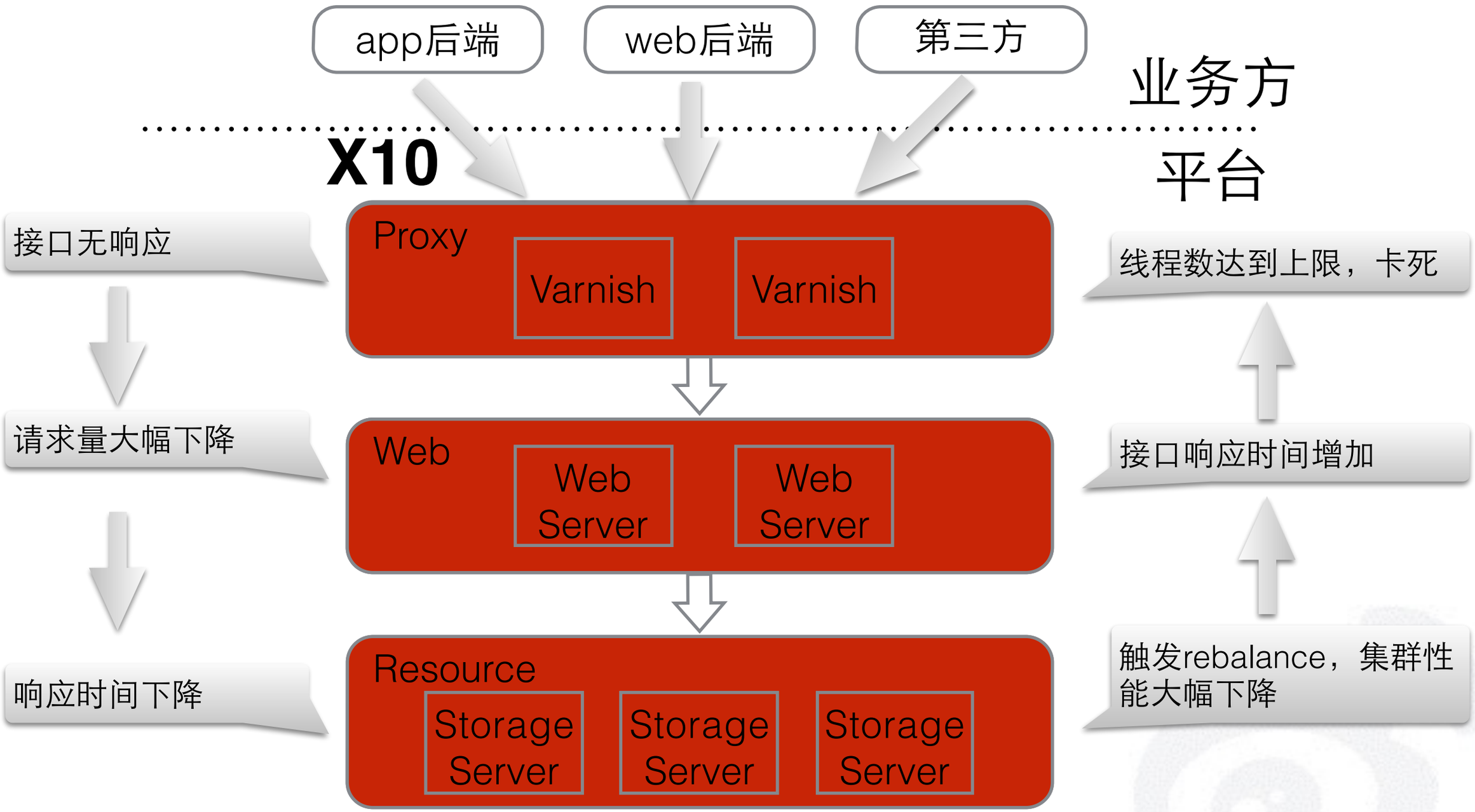






- 功能异常导致性能下降
- 性能问题影响功能可用性

案例一：性能问题影响功能可用性



案例二：功能异常导致性能下降

- 背景:
  - docker@春晚红包
  - 应对极端流量，应用docker进行快速服务调度
- 现象:
  - SocketTimeout
  - Connection Refuse
- 原因:
  - nat方式组网时，底层基于iptables
  - iptables底层基于netfilter内核模块
  - Netfilter模块保持65536个链接做NAT转换

```
Jan 30 20:14:11 localhost kernel: __ratelimit: 7035 callbacks suppressed
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:11 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: __ratelimit: 9166 callbacks suppressed
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:16 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:21 localhost kernel: __ratelimit: 6954 callbacks suppressed
Jan 30 20:14:21 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:21 localhost kernel: nf_conntrack: table full, dropping packet.
Jan 30 20:14:21 localhost kernel: nf_conntrack: table full, dropping packet.
```

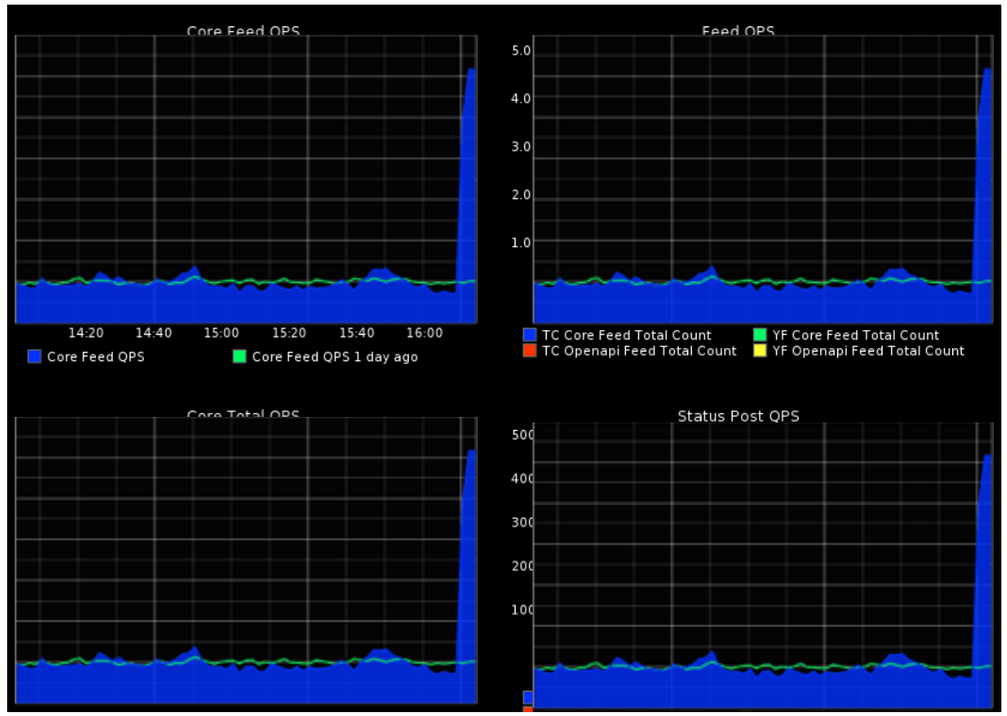
典型问题的特点

- 出现场景
  - 访问量量级增加
  - 引入新组件
- 问题表现
  - 一般表现为应用崩溃、服务不可用或系统雪崩等
- 原因排查
  - 量变到质变，低负载经验不适用
  - 问题往往牵扯多个领域

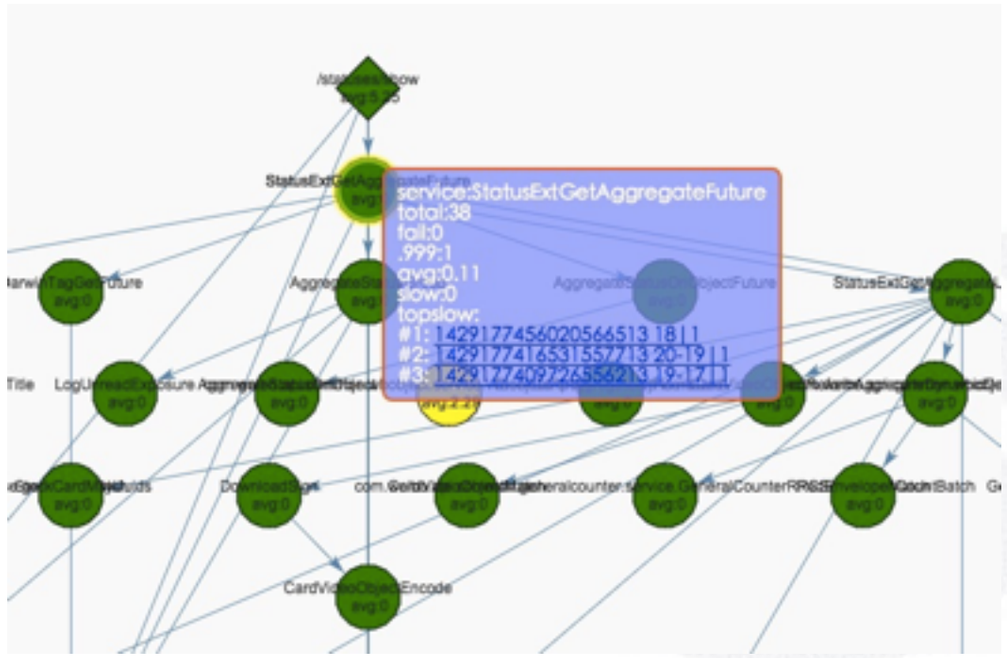
- 有哪些问题
- 如何排查
- 如何预防

监控

- Dashboard
  - 基于graphite
  - 集中展示
  - 定量分析

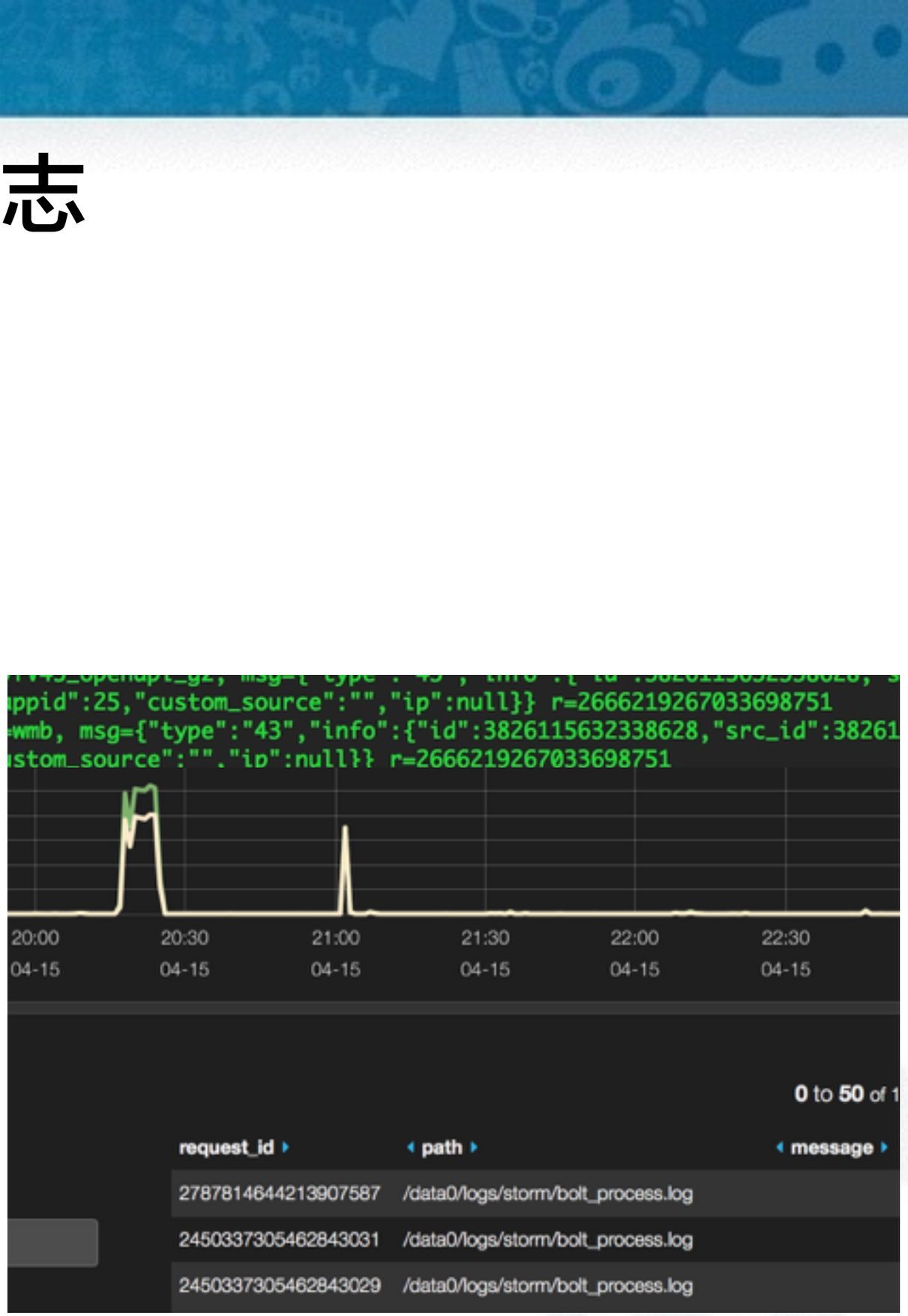


- Trace
  - 追踪链路，显示请求调用链
  - 分析节点异常：平均值、历史数据

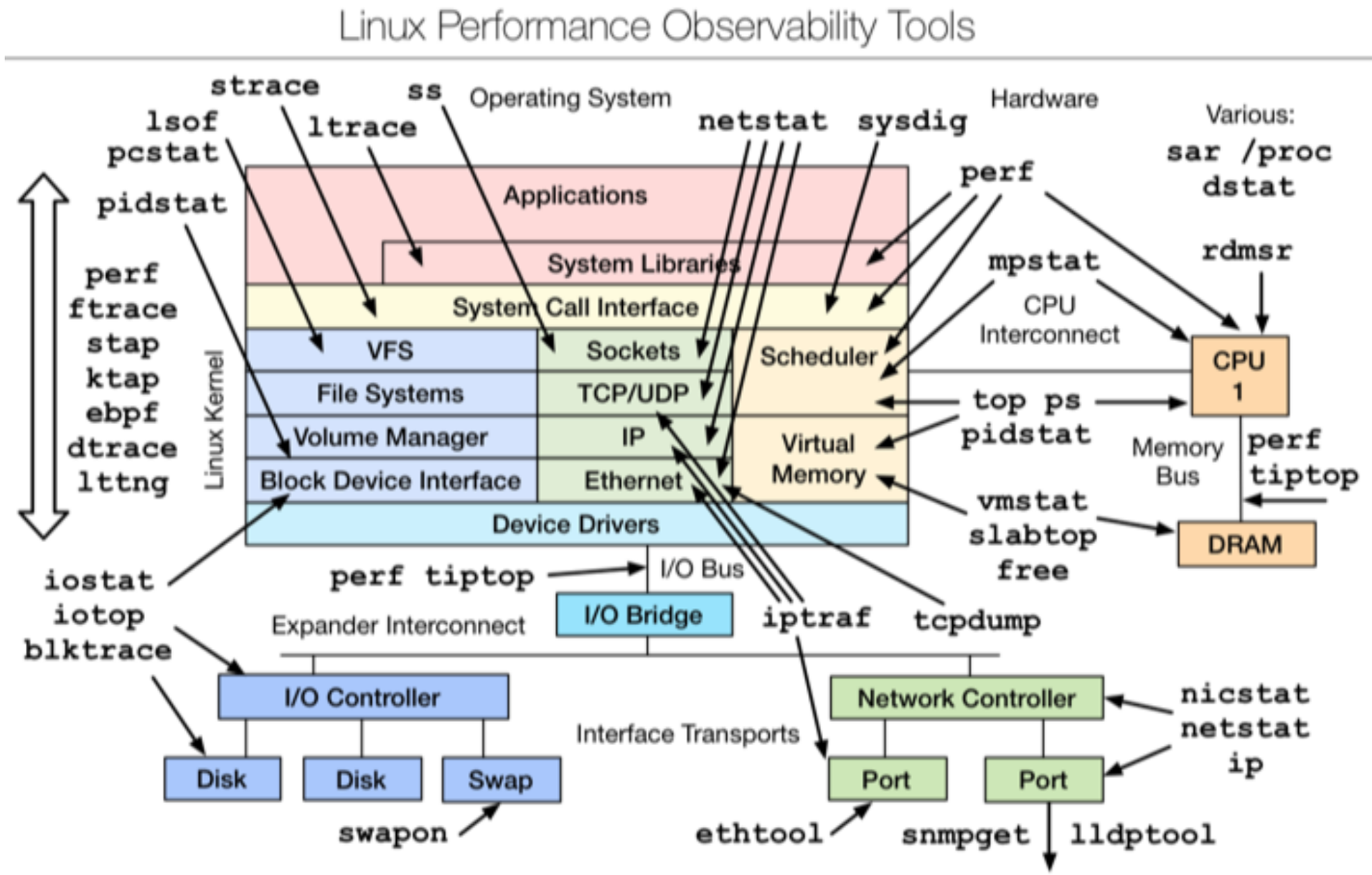


日志

- 信息要完整
  - 业务日志：包含关键路径与异常
  - 性能日志：性能统计与分步耗时
  - 容器日志、系统日志也很重要
    - gc log、/var/log
- 分维度过滤
  - 时间：出问题的时间点
  - 请求：uid、requestid
  - 级别：WARN/ERROR
- 集中检索
  - ELK/agent(jpool)
  - 平衡效率和成本

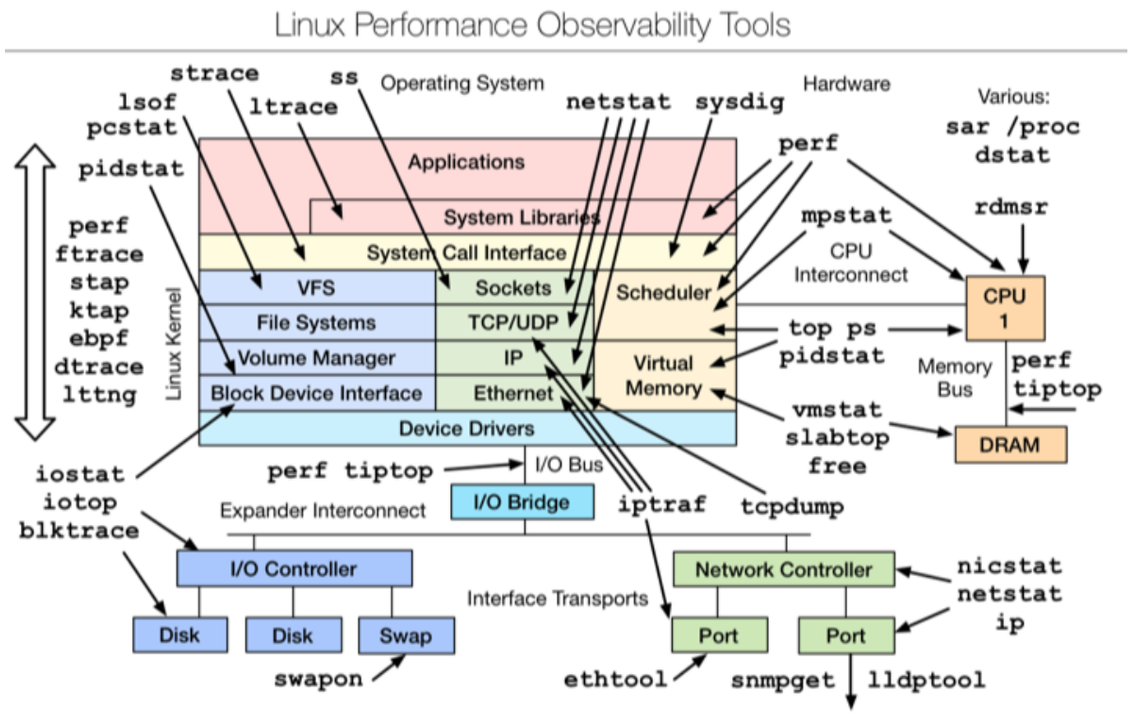


查看现场


Linux Performance Observability Tools
strace
ss
Operating System
Hardware
Various:
lsof
pcstat
ltrace
netstat
sysdig
perf
sar /proc
dstat
pidstat
Applications
System Libraries
rdmsr
perf
ftrace
stap
ktap
ebpf
dtrace
lttng
System Call Interface
mpstat
CPU
Interconnect
VFS
Sockets
Scheduler
CPU
1
File Systems
TCP/UDP
top ps
pidstat
Volume Manager
IP
Virtual
Memory
Memory
Bus
perf
tiptop
Linux Kernel
Block Device Interface
Ethernet
vmstat
slabtop
free
Device Drivers
DRAM
iostat
iotop
blktrace
perf tiptop
I/O Bus
I/O Bridge
iptraf
tcpdump
Expander Interconnect
I/O Controller
Network Controller
nicstat
netstat
ip
Interface Transports
Disk
Disk
Swap
Port
Port
swapon
ethtool
snmpget
lldptool
http://www.brendangregg.com/linuxperf.html 2014

查看现场

- 快照分析
  - 功能：观察程序当前的状态
  - 场景：程序当前处于整体异常状态
  - 举例：gdb、Xmap、mat、jstack
- 调用分析
  - 功能：观察调用和调用栈
  - 场景：请求出错、请求慢、偶发错误
  - 举例：btrace、Xtrace
- 聚合分析
  - 功能：按某些维度采样、聚合和对比数据
  - 场景：查找性能问题
  - 举例：perf、Xstat、Xtop

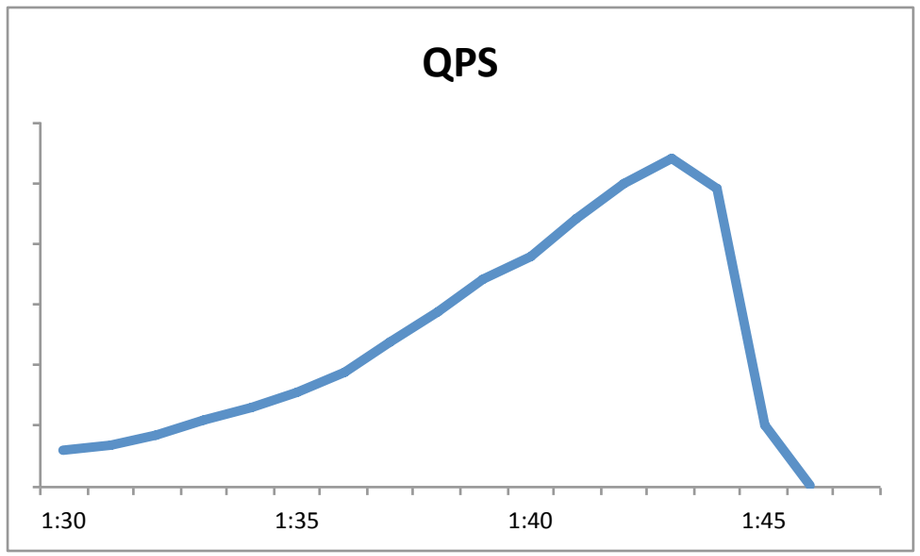


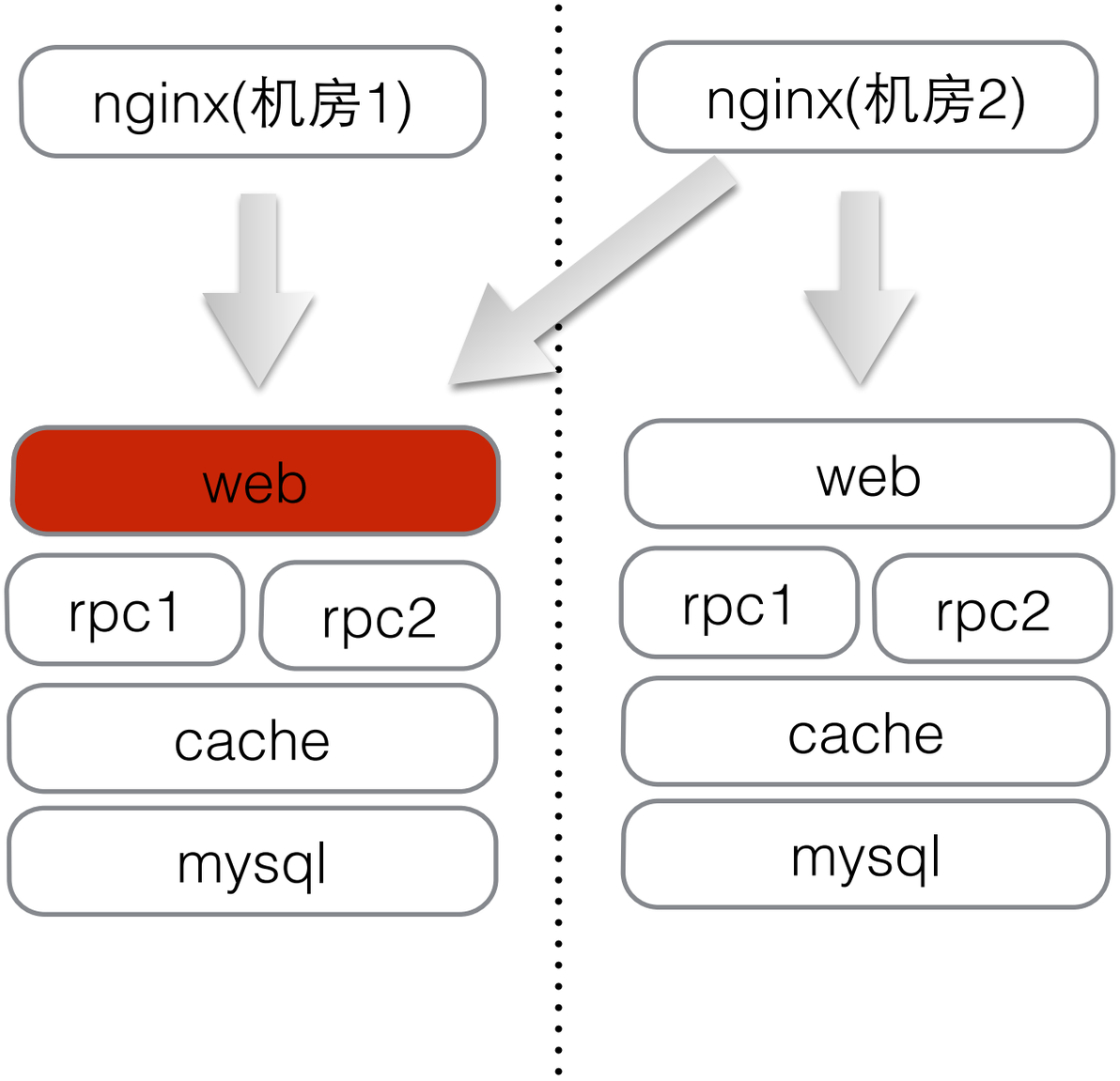
分析原因

- 问题经常隐藏在：
  - 应用：死锁、内存溢出、依赖资源
  - 系统：jvm、kernel、tcp
  - 硬件：硬件故障、网络
- 联合排查
  - 开发 / 运维 / dba / 网络 / 系统 / 硬件

案例：春晚演练时服务出现异常

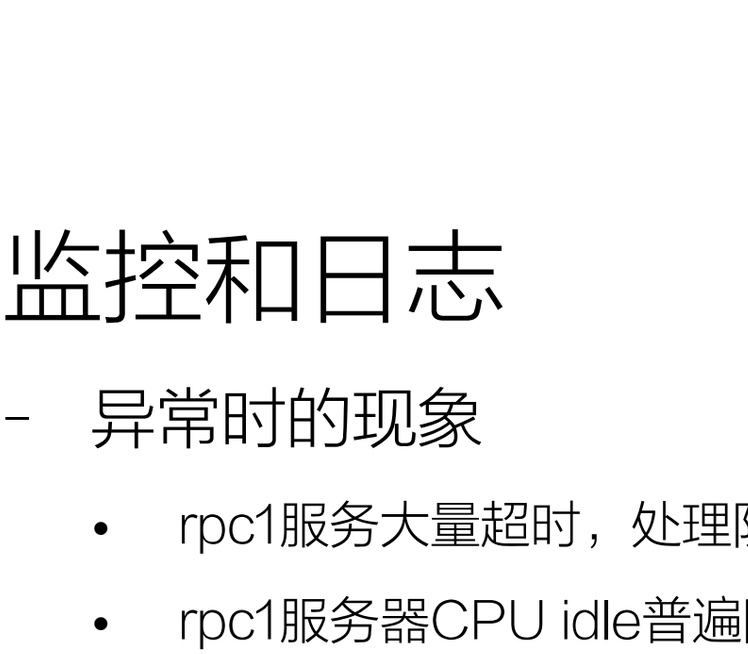
- 背景
  - 演练方案：跨机房迁移用户请求
  - 逐渐迁移流量的过程中，web服务突然僵死，大量返回503
  - 接口吞吐量低于理论值2-3倍



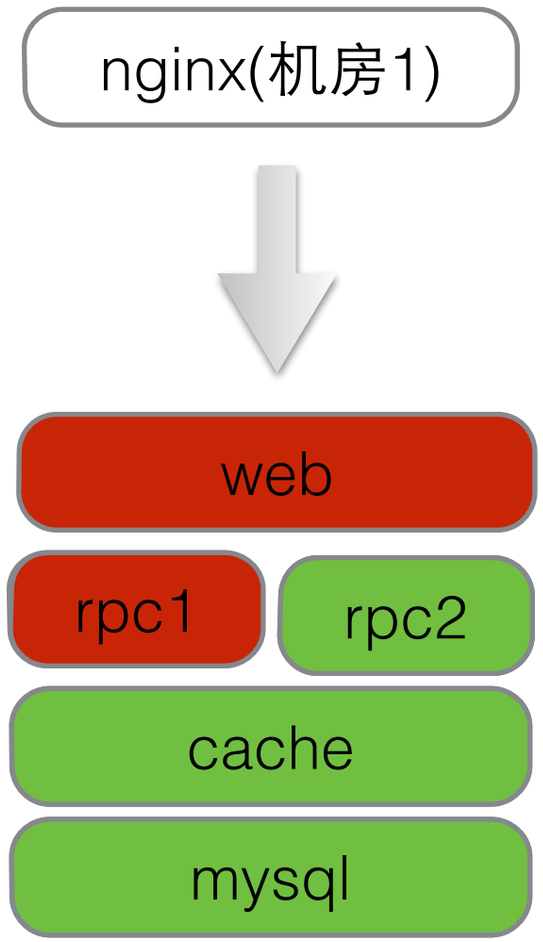


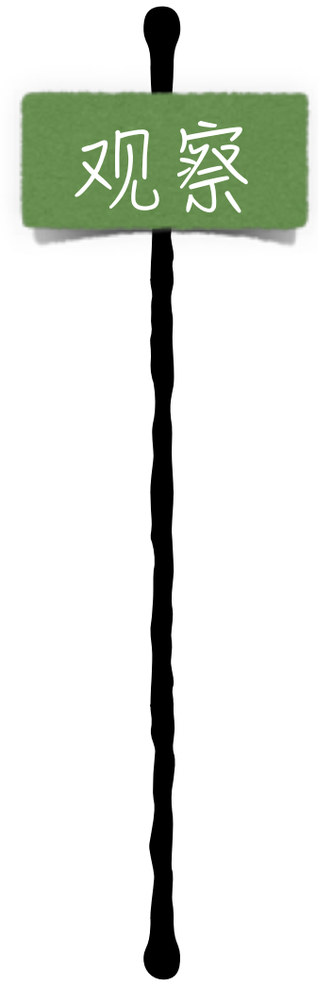
定位问题

- 监控和日志
  - 异常时的现象
    - rpc1服务大量超时，处理队列堵塞
    - rpc1服务器CPU idle普遍降低至10%
    - 后端资源没有波动



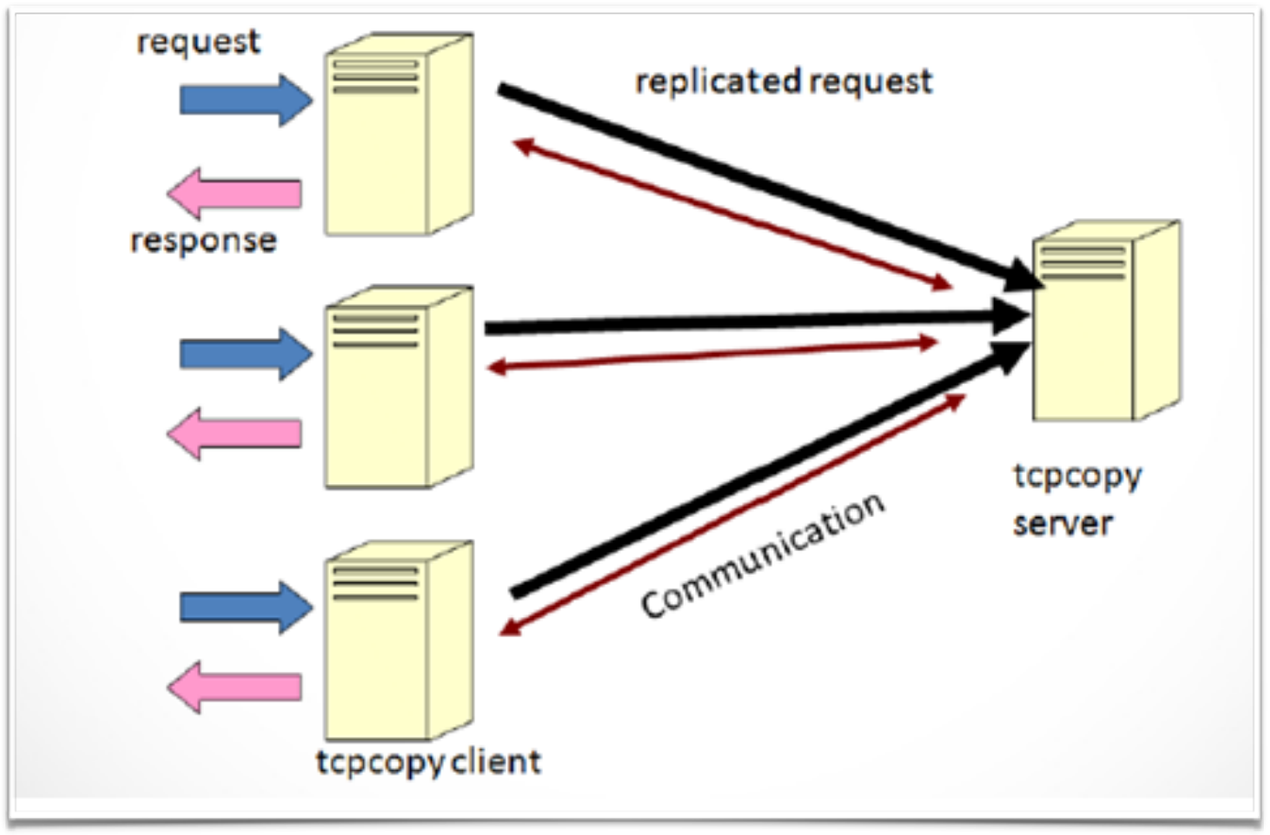
rpc接口响应时间





重现问题

- 重现现场
  - 直接压测没有重现问题
  - 通过tcp copy引流线上流量，问题重现



瓶颈分析

- 瓶颈分析
  - 通过perf发现close系统调用消耗了大量cpu


```
Samples: 1M of event 'cycles', Event count (a
 45.08%  [kernel]       [k] __close_fd
  8.10%  libc-2.12.so   [.] __GI___libc_close
```

瓶颈分析

- 瓶颈分析
  - 通过perf发现close系统调用消耗了大量cpu
  - 通过jstack发现大量线程卡在nio的closeIntFD上


```
"NettyServer-              :8883-3-thread-91" daemon prio=10 tid=0x0
   java.lang.Thread.State: RUNNABLE
        at sun.nio.ch.FileDispatcher.closeIntFD(Native Method)
        at sun.nio.ch.EPollSelectorImpl.implClose(EPollSelectorImp
        at sun.nio.ch.SelectorImpl.implCloseSelector(SelectorImpl.
        - locked <0x00000007fde9b530> (a sun.nio.ch.Util$2)
        - locked <0x00000007fde9b520> (a java.util.Collections$Unm
        - locked <0x00000007fde9b2a8> (a sun.nio.ch.EPollSelectorI
        at java.nio.channels.spi.AbstractSelector.close(AbstractSe
        at cn.vika.memcached.MemCachedClient$NIOLoader.doMulti(Mem
```

瓶颈分析

- 瓶颈分析
  - 通过perf发现close系统调用消耗了大量cpu
  - 通过jstack发现大量线程卡在nio的closeIntFD上
  - 通过strace发现close系统调用的处理时间很长


```
1429767534.584746 close(3)                = 0 <1.000025>
1429767534.584814 close(3)                = 0 <0.921013>
1429767534.584870 close(3)                = 0 <0.874420>
1429767534.584925 close(3)                = 0 <1.500072>
1429767534.584986 close(3)                = 0 <0.189031>
1429767534.585042 close(3)                = 0 <0.724016>
```

瓶颈分析

- 瓶颈分析
  - 通过perf发现close系统调用消耗了大量cpu
  - 通过jstack发现大量线程卡在nio的closeIntFD上
  - 通过strace发现close系统调用的处理时间很长
  - 通过对比压测发现旧版内核频繁调用close时有性能问题

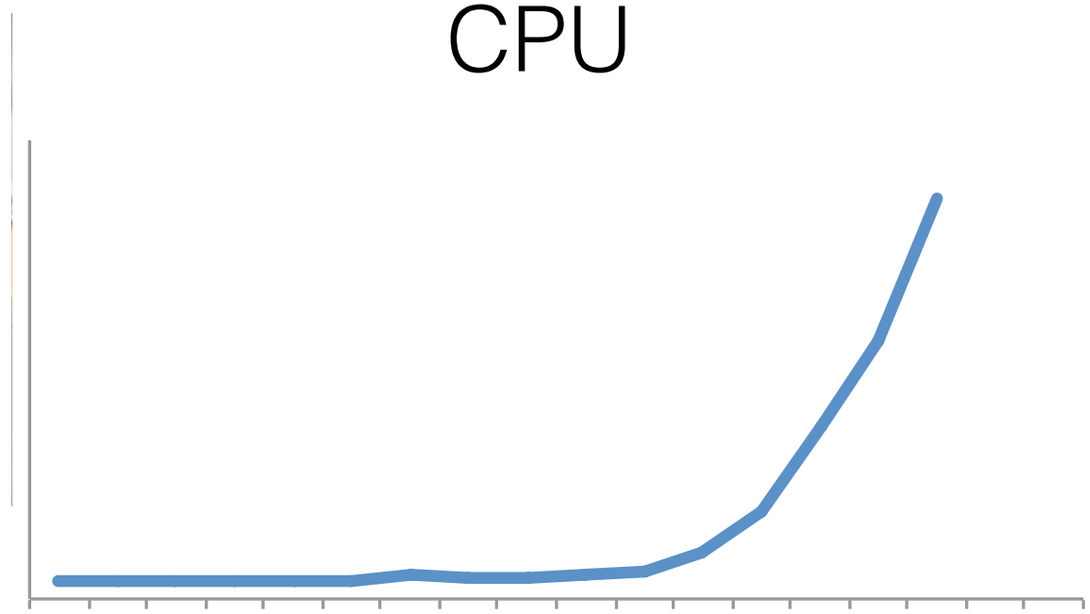
解决问题

- 解决方式
  - 升级内核
- 确保问题被修复
  - 开发环境验证
  - 测试环境验证
  - 线上环境验证

- 确保没有引入新的问题
  - 灰度上线，观察服务状态
    - 功能无异常
    - 性能未下降

tips：当心解决问题的陷阱！

- 案例：追查性能问题
  - 现象：接口突然开始超时
  - 原因：访问量突增，大量读取静态内容缓存，mc出现带宽瓶颈
  - 解决：增加本地缓存
  - 结果：压测时性能仍然有问题
    web服务器CPU瓶颈



顺便加个统计

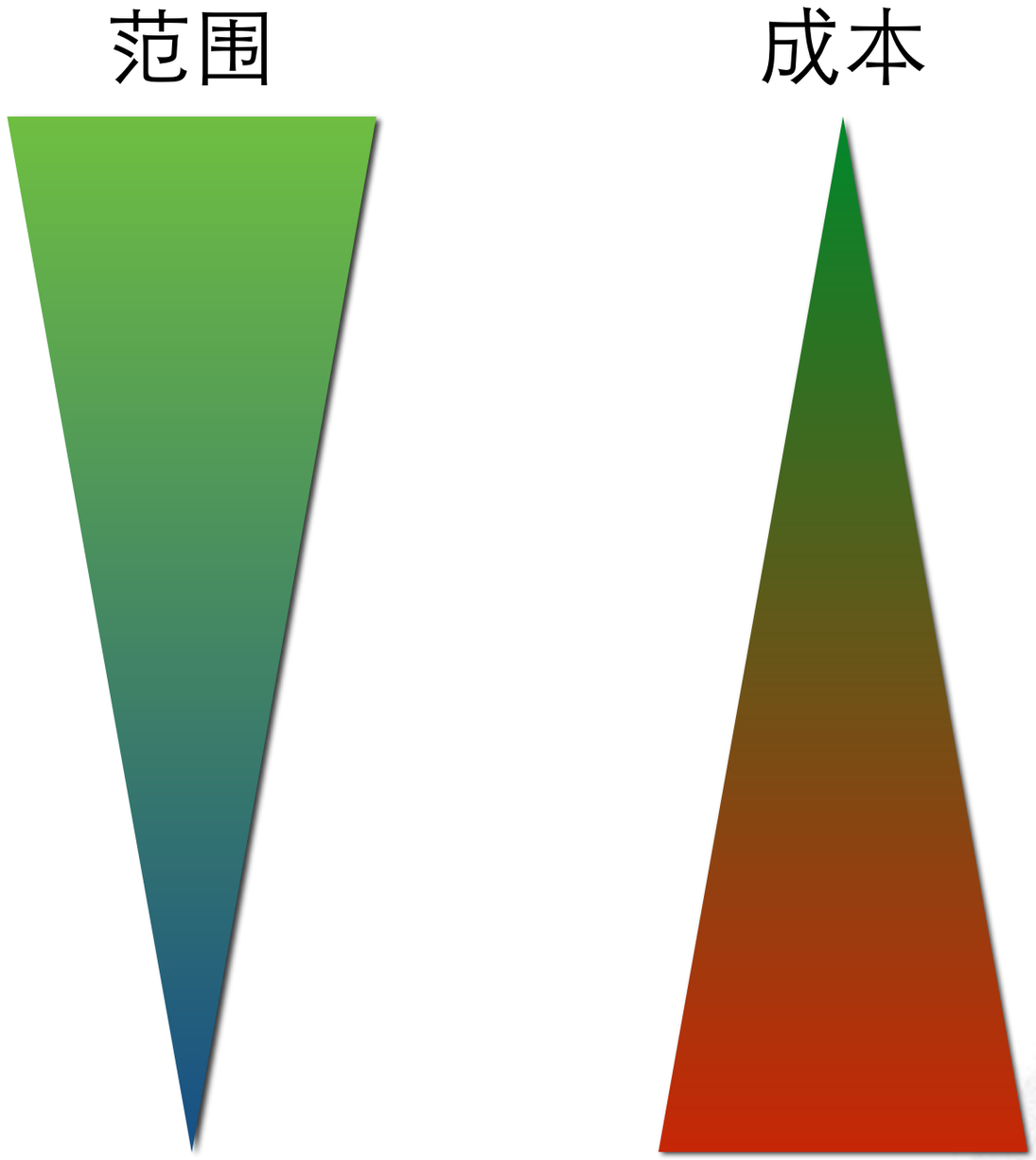
小结：在大高系统中排查问题

观察

重现

分析

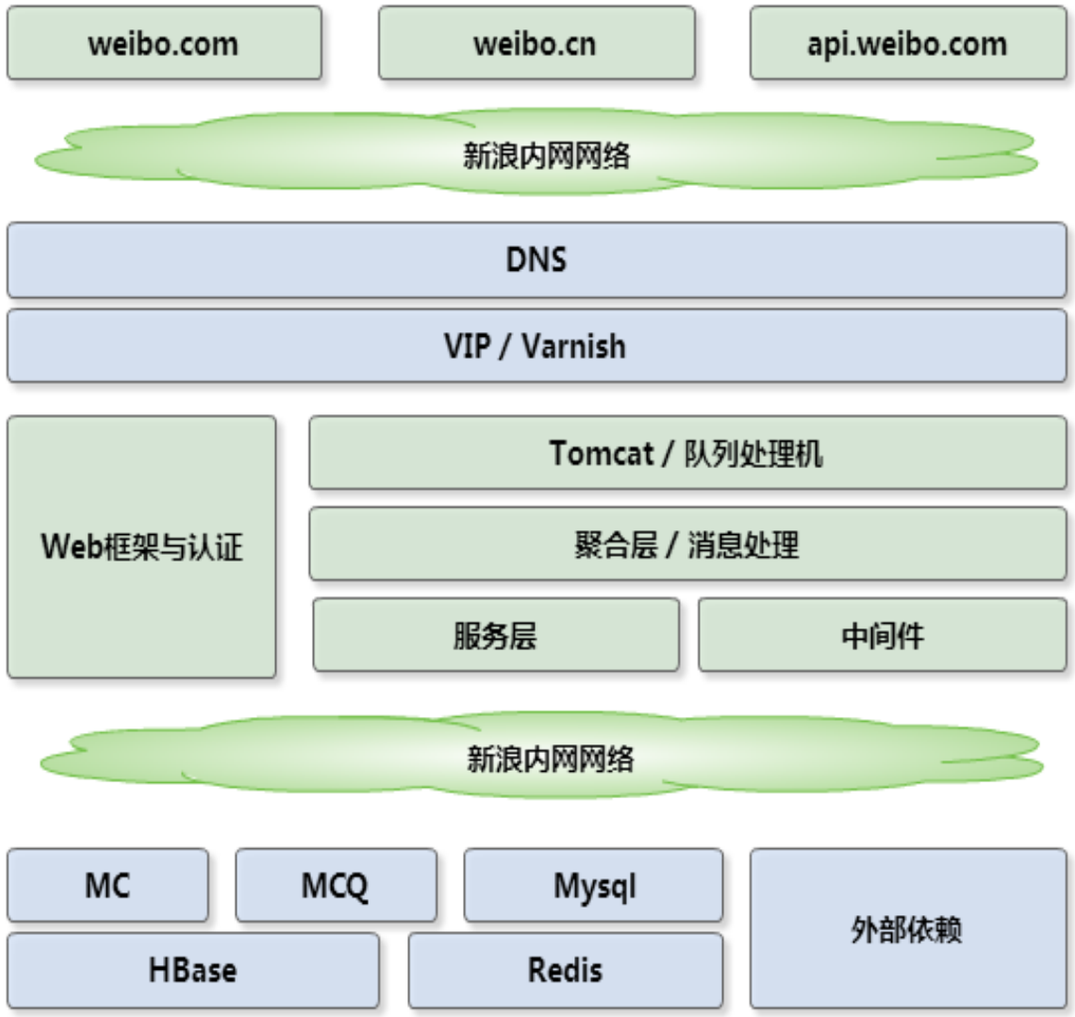
解决



- 有哪些问题
- 如何排查
- 如何预防

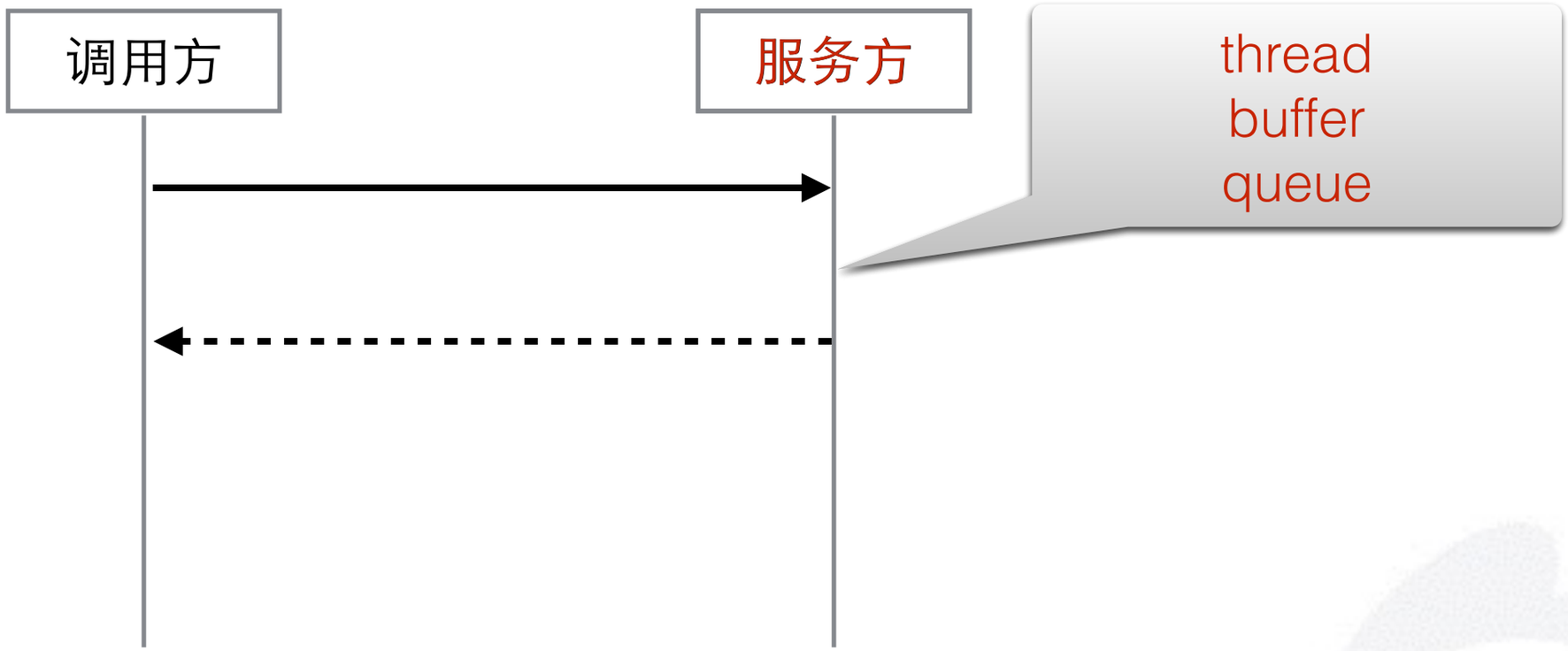
高可用架构设计

- 服务隔离
  - 按部署隔离
    - 分机房部署
    - 核心服务独立部署
    - 服务独立化部署
  - 按调用隔离
    - 异步队列
    - 快速失败



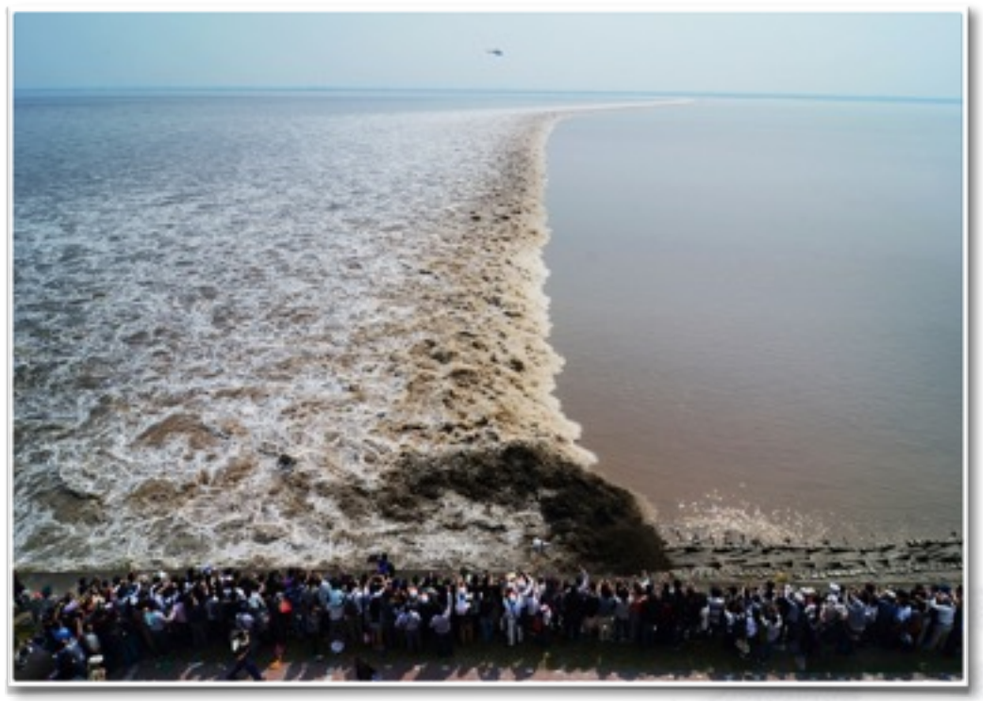
可靠的系统实现

- 耦合方式：同步 / 异步 /丢弃



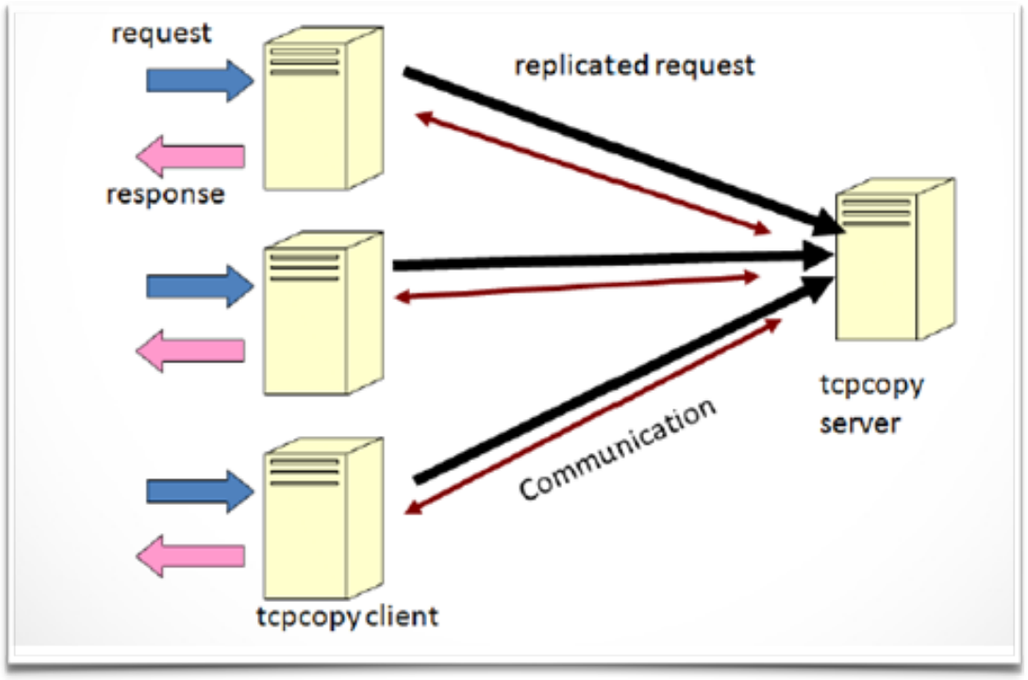
可靠的系统实现

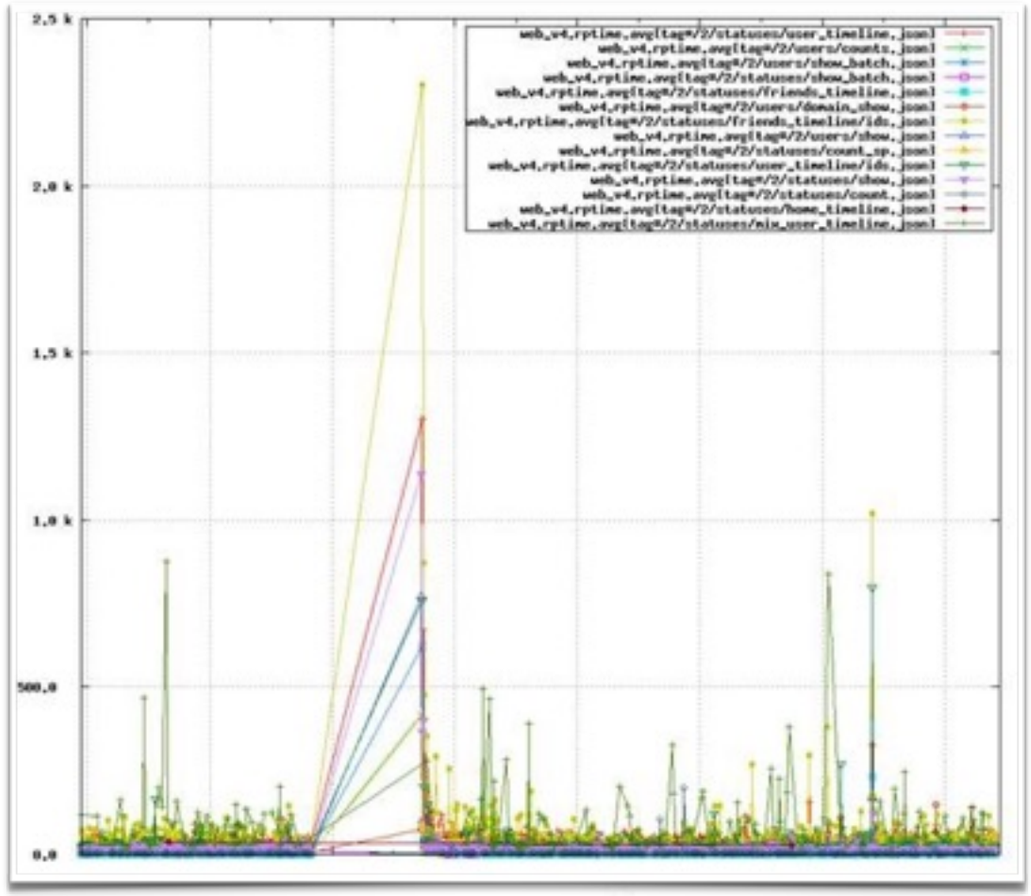
- 耦合方式：同步 / 异步 /丢弃
- 异常处理的异常处理：不要让事情变得更糟



压测与演练

- 真实流量压测
  - 模拟实际请求模型：TCPCopy
  - 模拟后端资源异常：TouchStone(tc)





最后再说几句

- Q：如何判断一个系统在大规模、高负载下是否可靠?
- A：没有实际流量验证前一定不可靠，验证后也不一定是可靠的。

- Q：压测压出问题怎么办?
- A：压测压不出问题怎么办?

- Q：处理预案是越多越好吗?
- A：一个演练过的预案要好过十个没有演练过的预案。

- Q：我特别害怕自己做的系统在高负载时出故障，怎么办?
- A：微博平台欢迎你:)

Q&A

秦迪 @蛋疼的axb

@微博平台架构