



分布式行情推送系统



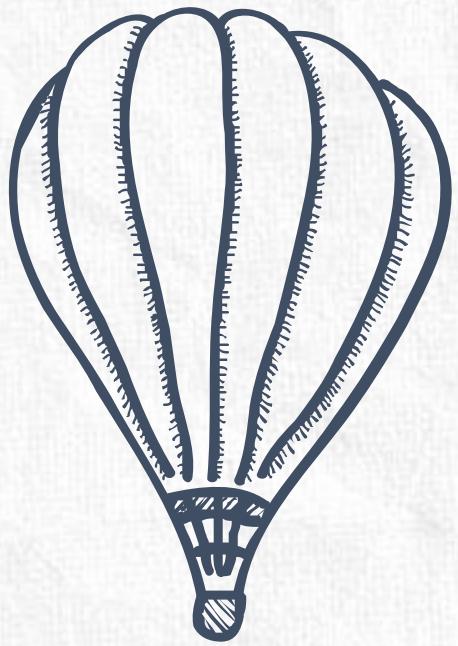
rfyiamcool

xiaorui.cc

github.com/rfyiamcool



CONTENT



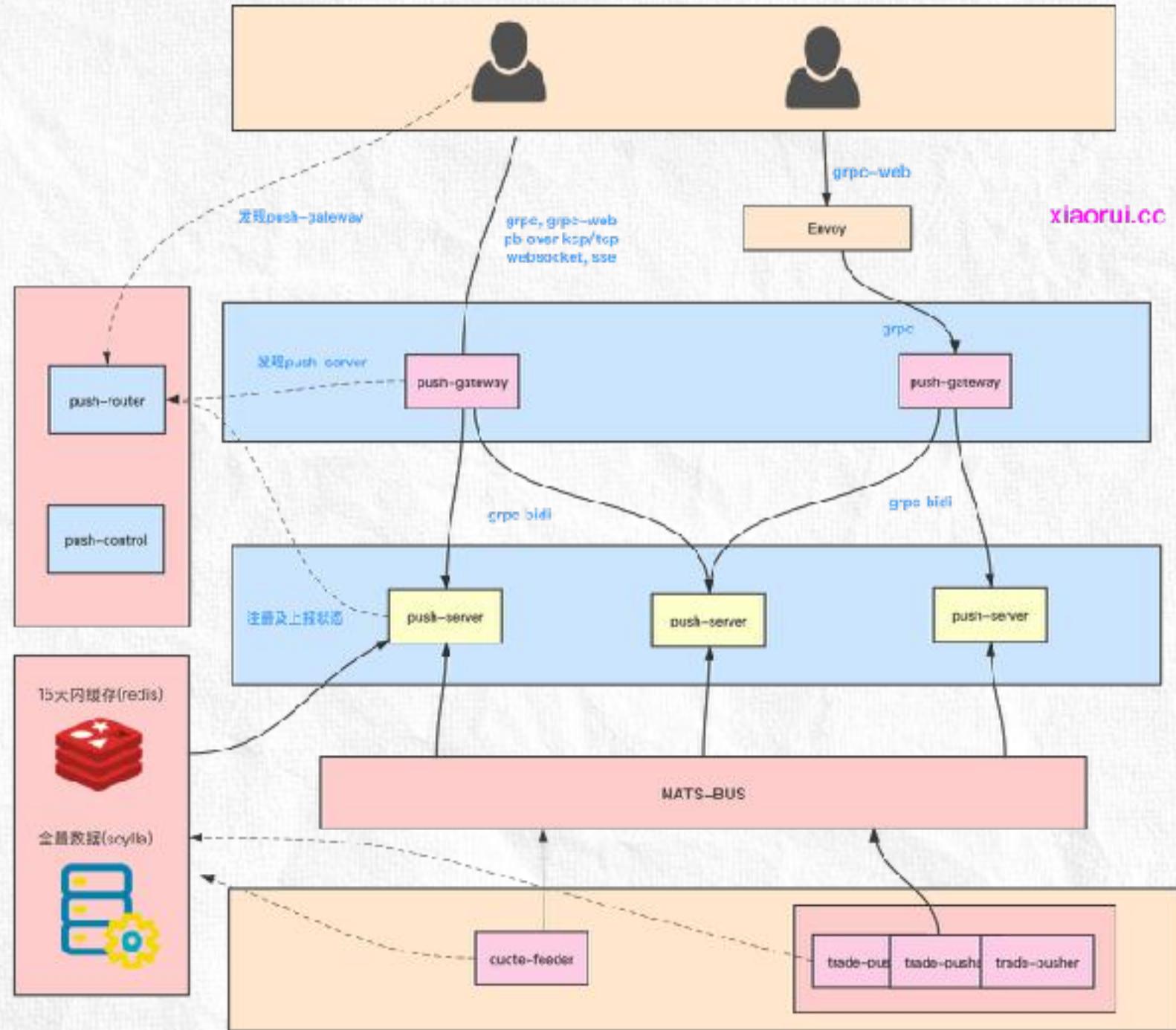
- 1 架构介绍
- 2 性能优化
- 3 排坑记
- 4 总结



1

架构设计





组件介绍

* push-router

- * 服务发现注册及调度

* push-gateway

- * 鉴权及协议转换

- * 支持websocket, grpc, grpc-web, kcp

* push-server

- * 业务逻辑, 维护订阅关系及缓存

* push-control

- * 集群管理

* nats-bus

- * 消息总线

技术选型

* golang

* proto

* grpc

* envoy -> grpc-web

* websocket

* pb over tcp

* pb over kcp

* json

* protobuf

* mq

* nats-stream

* cache

* redis

* database

* scylla

技术选型

- * grpc

- * 内部量化

- * grpc-web

- * web前端

- * websocket

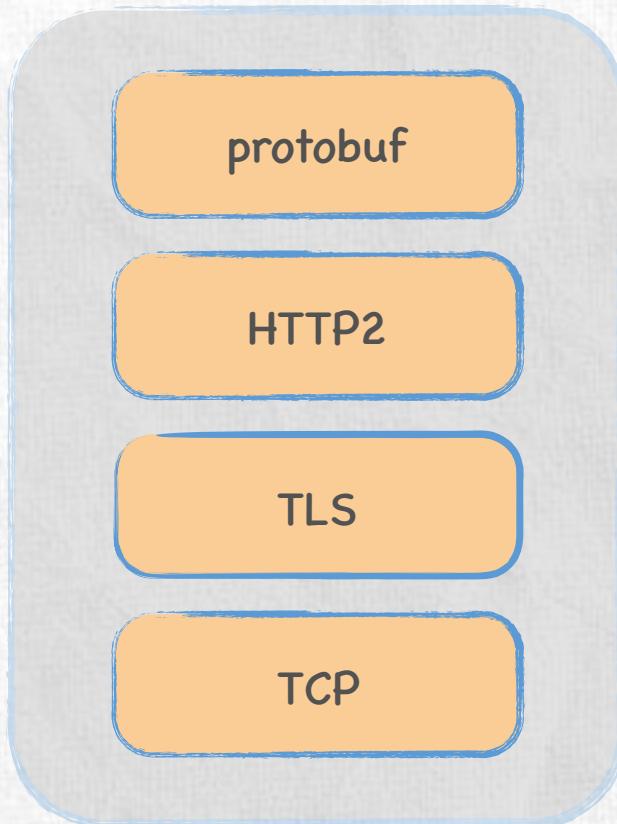
- * 供外部量化api

- * 移动端

- * pb over tcp

- * pb over kcp ...

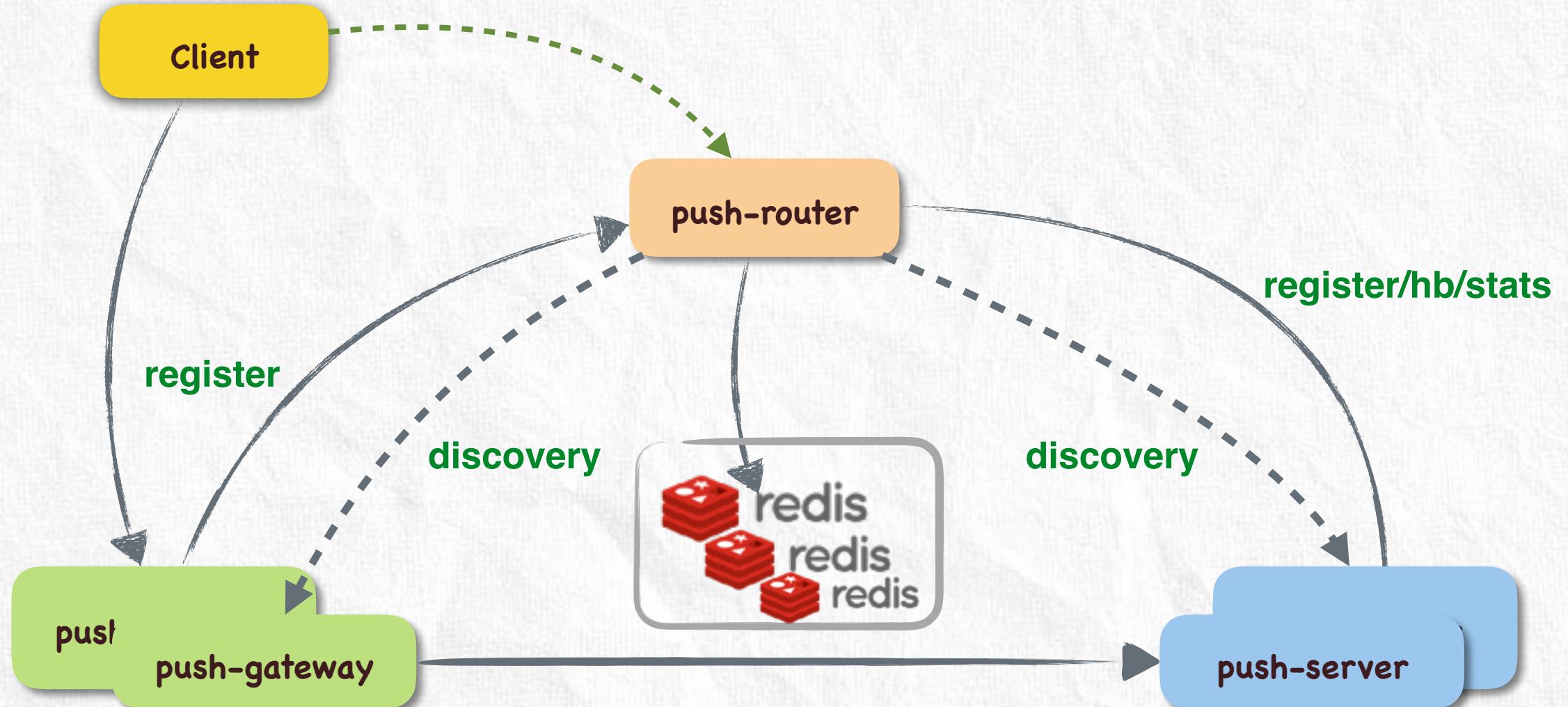
grpc



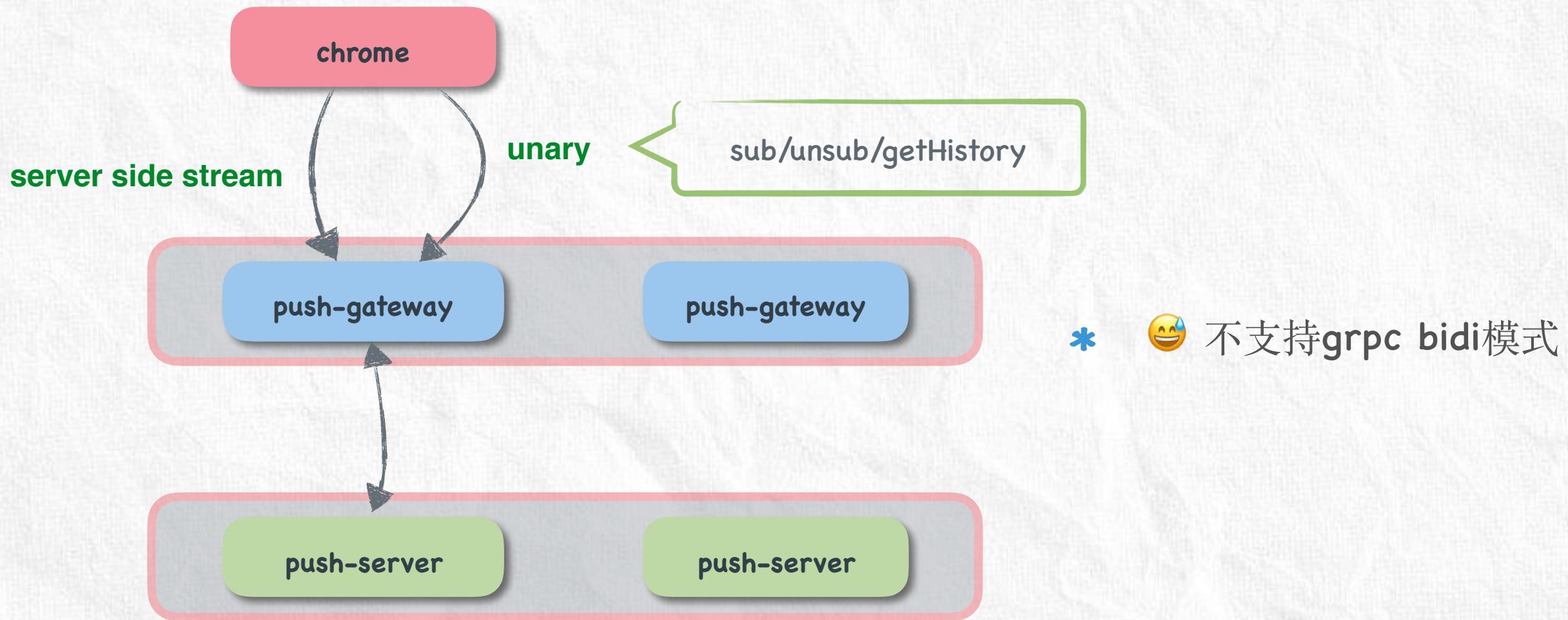
- * 优点

- * 支持各类语言
- * 基于**http2**兼容好
- * 支持**bidi**全双工通信模式
- * **protobuf**
- * 高性能序列化
- * 压缩

服务注册发现



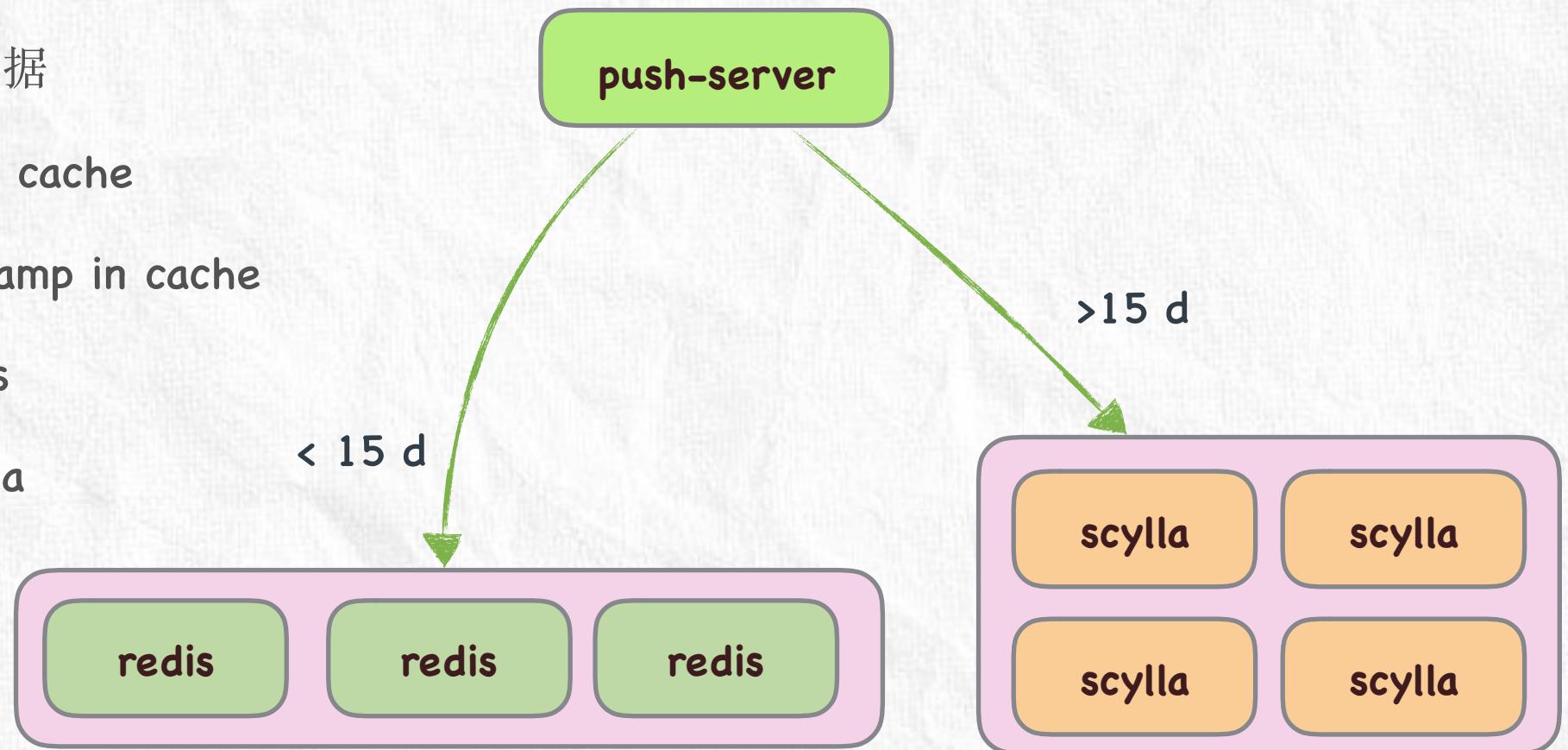
grpc-web in fe



多级缓存

- * push-server

- * 初始化缓存数据
- * expire 24h in cache
- * query timestamp in cache
- * < 15d in redis
- * > 15d in scylla



2

性能优化



推送优化

- * 当某个topic的订阅者超过一个量级
 - * 并发触发会更快
 - * 协程池有效减少栈扩充消耗
 - * 切chunk可减少协程池调度

```
// 伪代码
if msg.data == kline {
    mapping.lock()
    clients := mapping.get(msg.tag)
    mapping.unlock()

    for c := range clients {
        c.notify(msg.data)
    }
}

// 优化后
if msg.data == kline {
    mapping.lock()
    clients := mapping.get(msg.tag)
    mapping.unlock()

    clientChunks := splitChunk(clients)

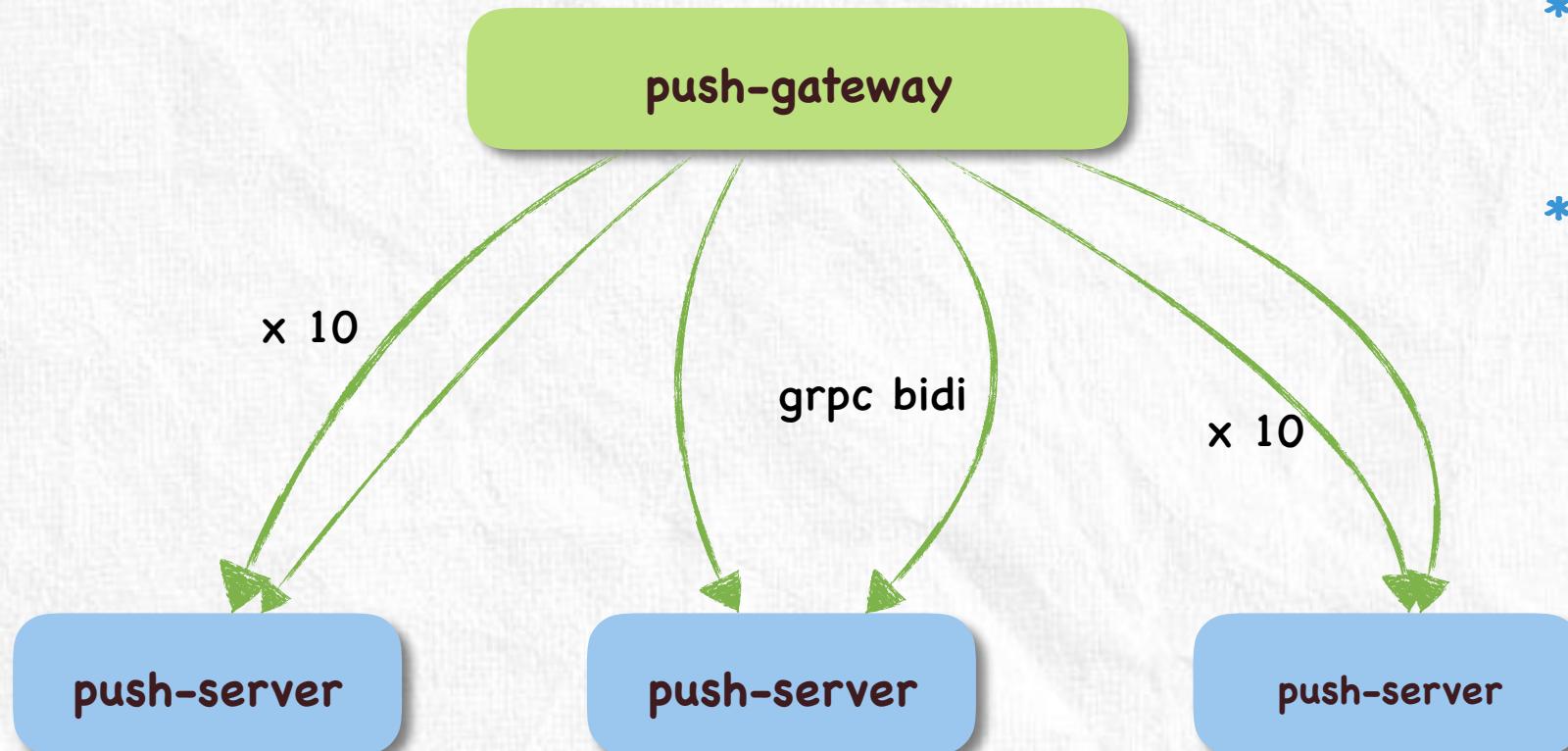
    for chunk := range clientChunks {
        gopool.put(func(){
            notify(chunk)
        })
    }
}
```

推送优化

- * **kline**订阅关系变更
 - * 用户的订阅和撤销订阅，以及上下线
 - * 发布订阅消息时需遍历订阅客户端
- * **kline**是双层嵌套的**map**, 频繁变更带来锁竞争
- * **kline map**
 - * **ticker_btc:usdt**
 - * **client1**
 - * **client2**
 - * ...
 - * **ticker_btc:eth**
 - * **client1**
 - * **client3**
- * 嵌套**map**改成分段为1024个**map**结构
- * 锁粒度尽量降低到**topic**级别

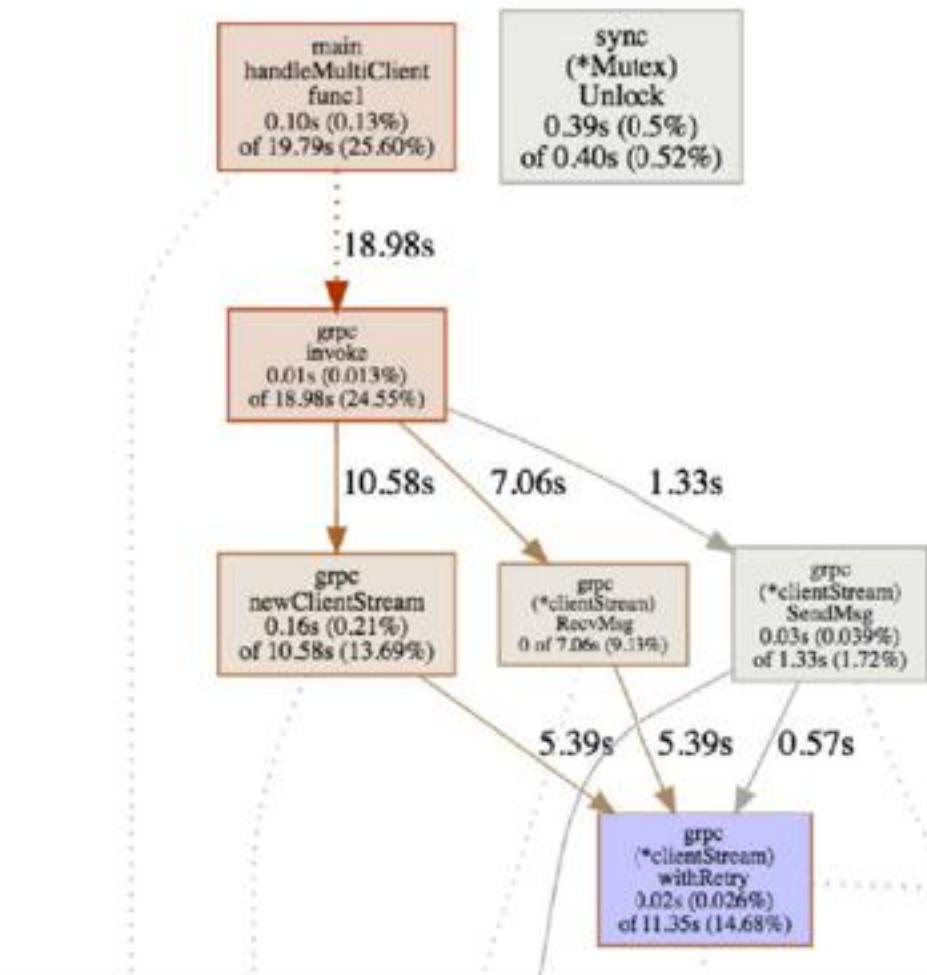
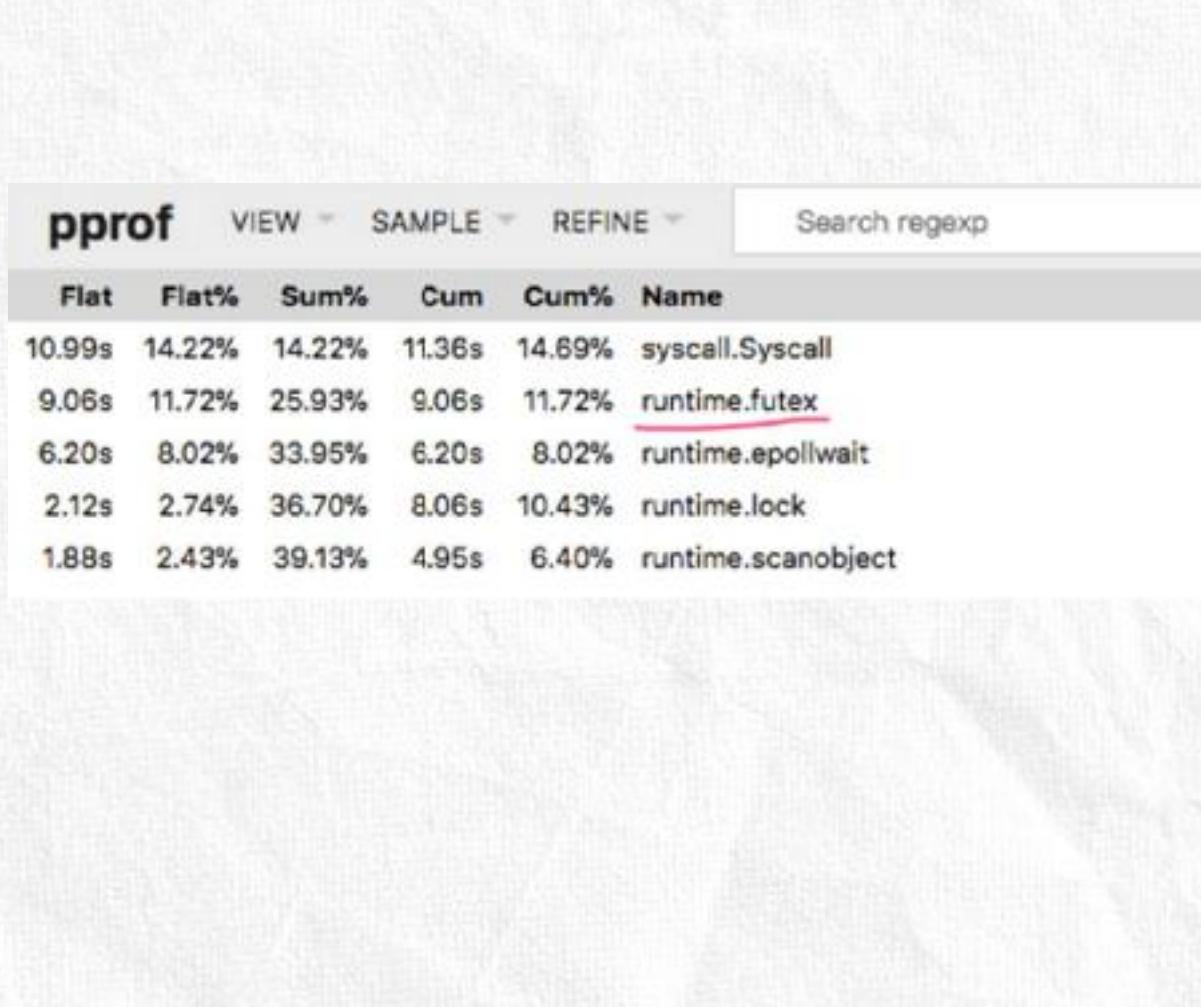


grpc连接池

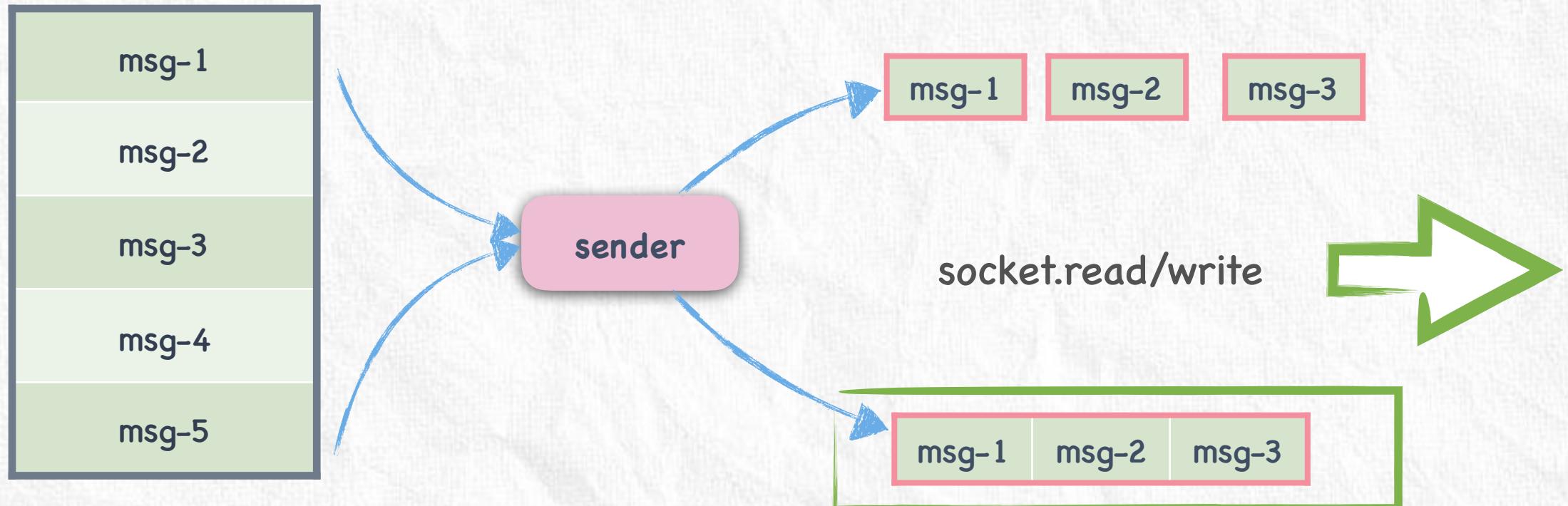


- * 为什么要使用连接池?
- * stream复用产生了锁竞争
- * benchmark
 - * 1 client, 100 goroutine, 8w qps
 - * 1 client, 300 goroutine, 5w qps
 - * 10 client, 300 goroutine, 15w qps
 - * 50 client, 300 goroutine, 30w qps

grpc连接池



减少系统调用



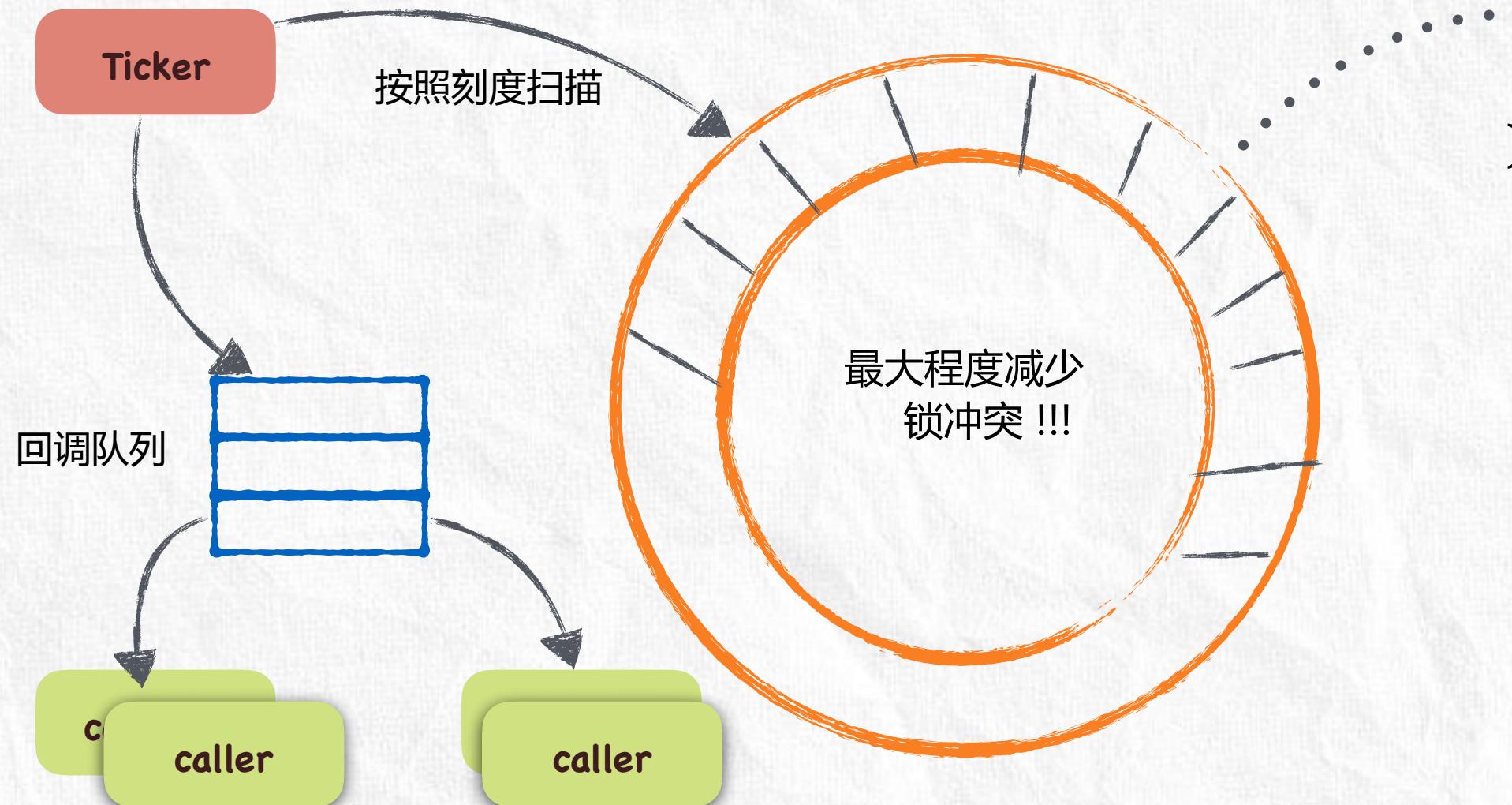
- * 减少上下文切换
- * 减少cpu sys的使用率

心跳定时器优化

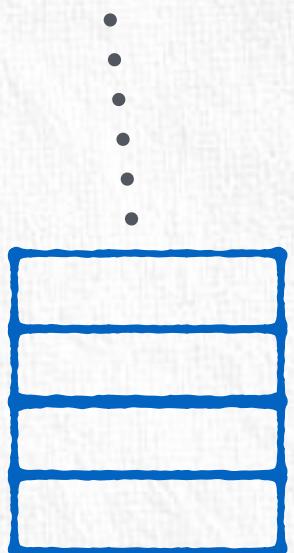
- * 升级golang版本到1.10.3以上
 - * runtime改进为p个timer定时器
 - * 实现自定义时间轮
 - * 锁分散到每个槽位
 - * 使用map存储定时任务
 - * 损失精度来减少锁竞争
- * 严重的锁竞争
- * 业务上允许低时间精度



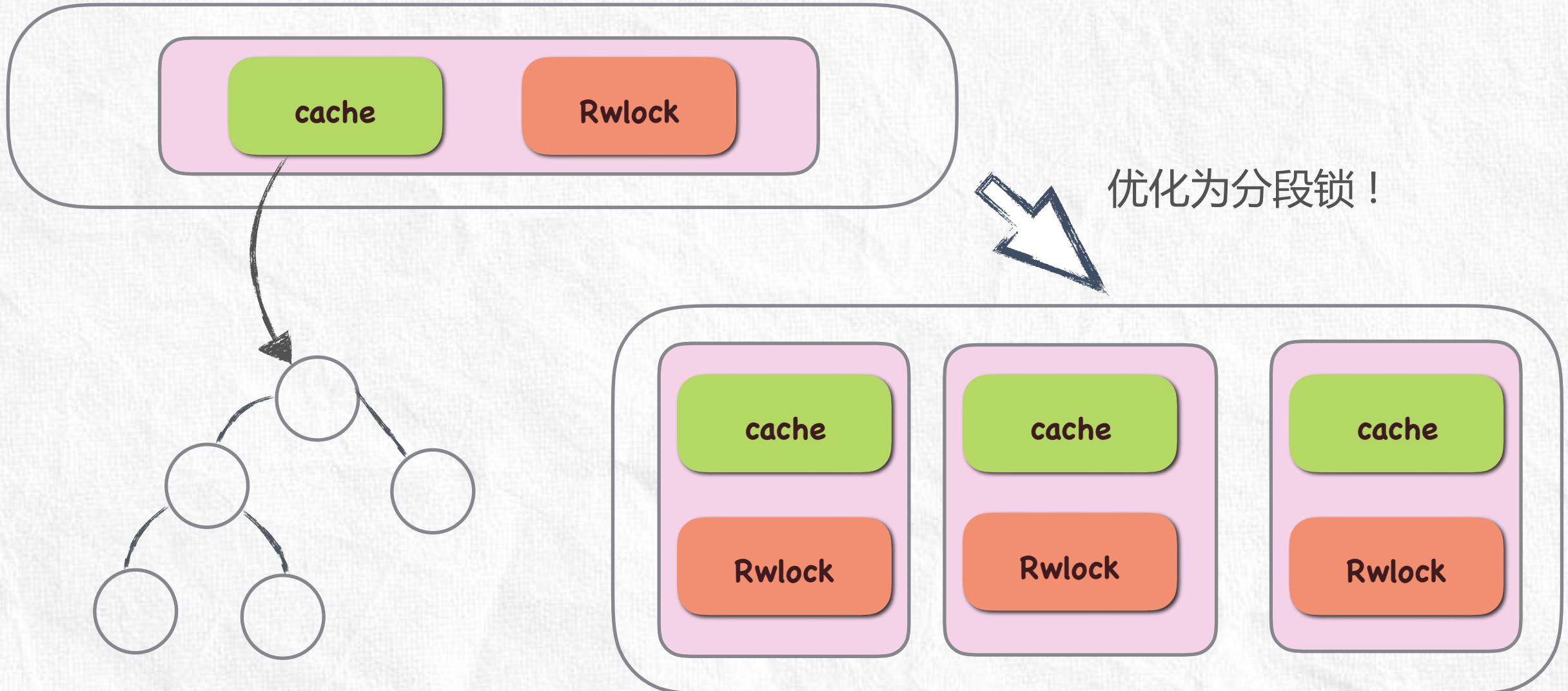
定时器优化



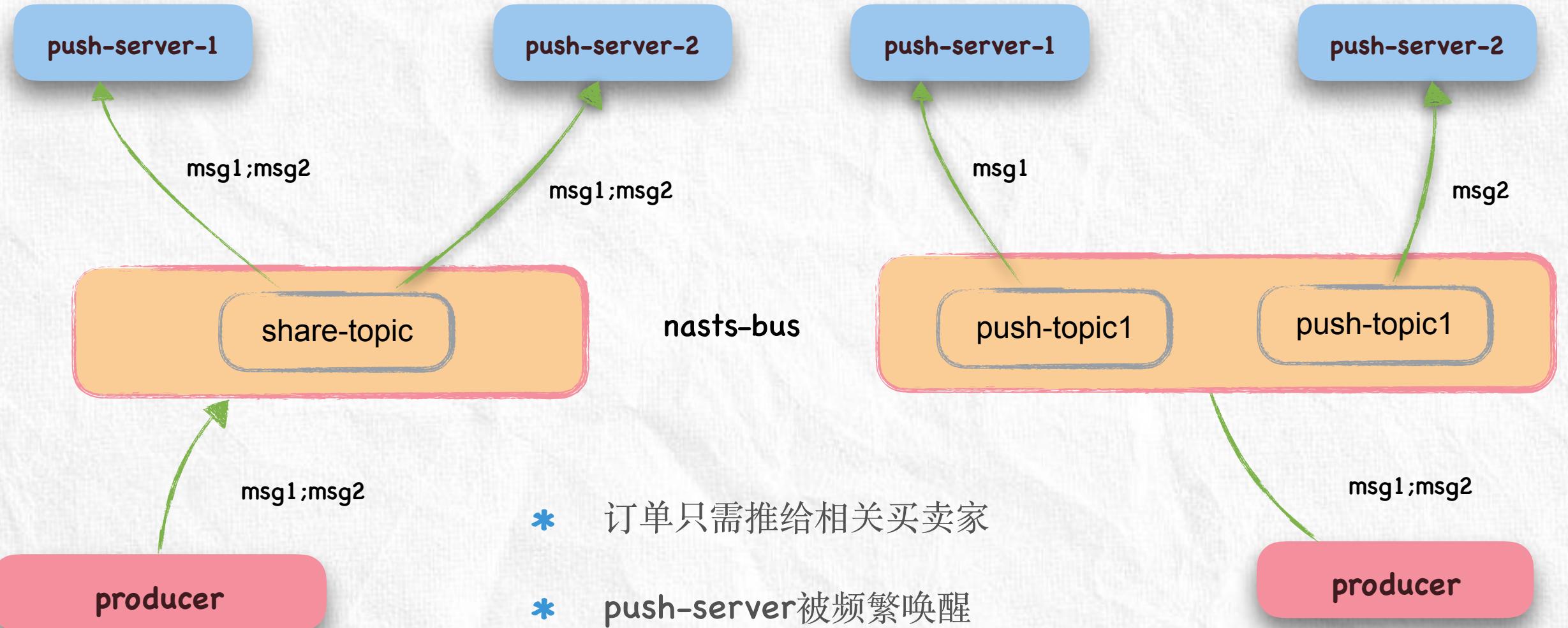
```
struct timerEntry {  
    map  
    cas  
}
```



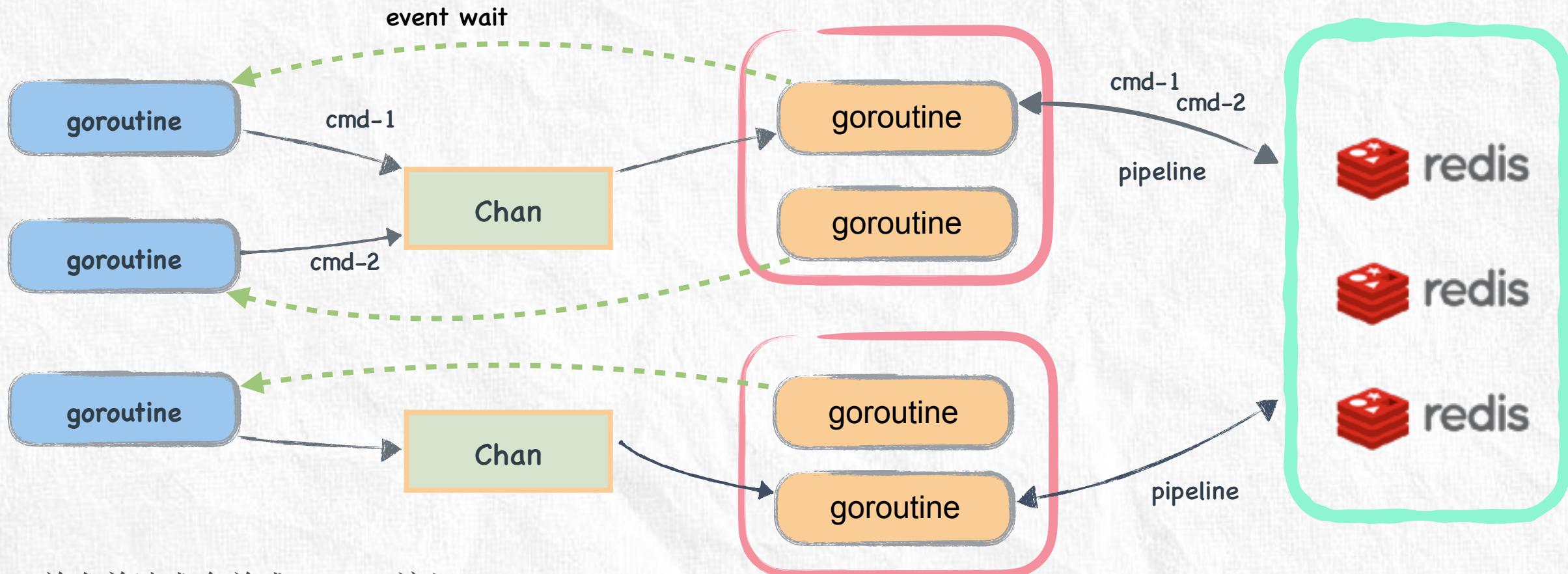
缓存优化



广播惊群

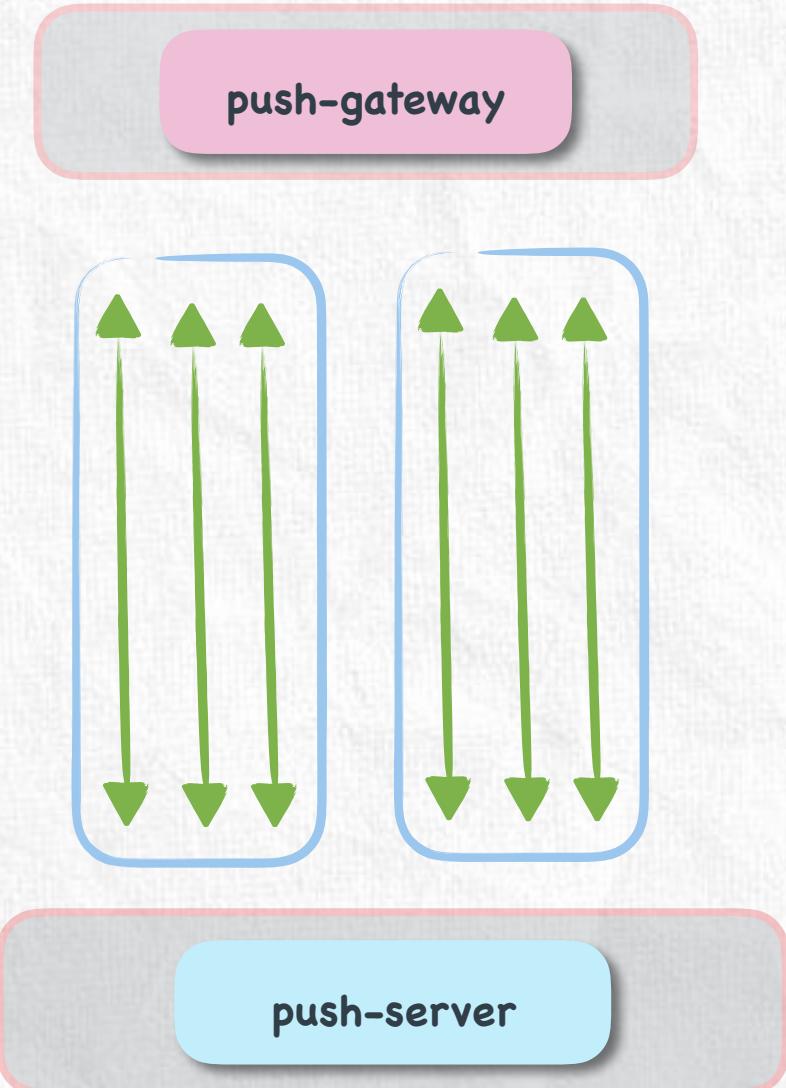


加大吞吐



- * 并发单请求合并成pipeline访问redis
- * 减少系统调用开销

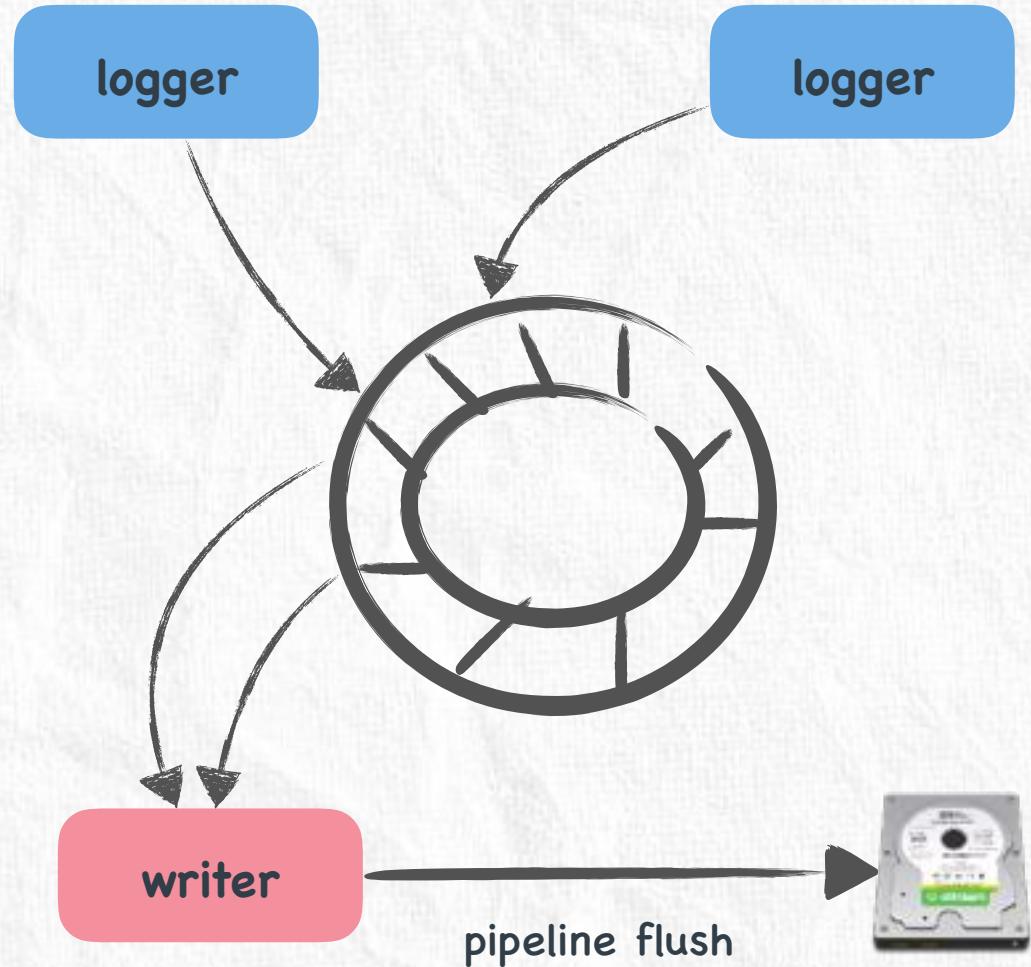
减少协程数



- * 每个订阅请求复用已创建好的**stream**池
- * **push-gateway** -> **push-server**
 - * 初始化 > 10个连接
 - * 共初始化100个通用型的**stream**通道
 - * 每个**stream**为一个go协程

日志引起的问题

- * golang线程数增多
 - * 宿主机**disk io**有时飙高，引起写日志阻塞
 - * 继而造成**runtime sysmon**检测
 - * 由于**syscall**长时间阻塞，启用新线程绑定P
 - * 线程数不会减少
- * 由于**disk io**阻塞业务协程，造成时延升高

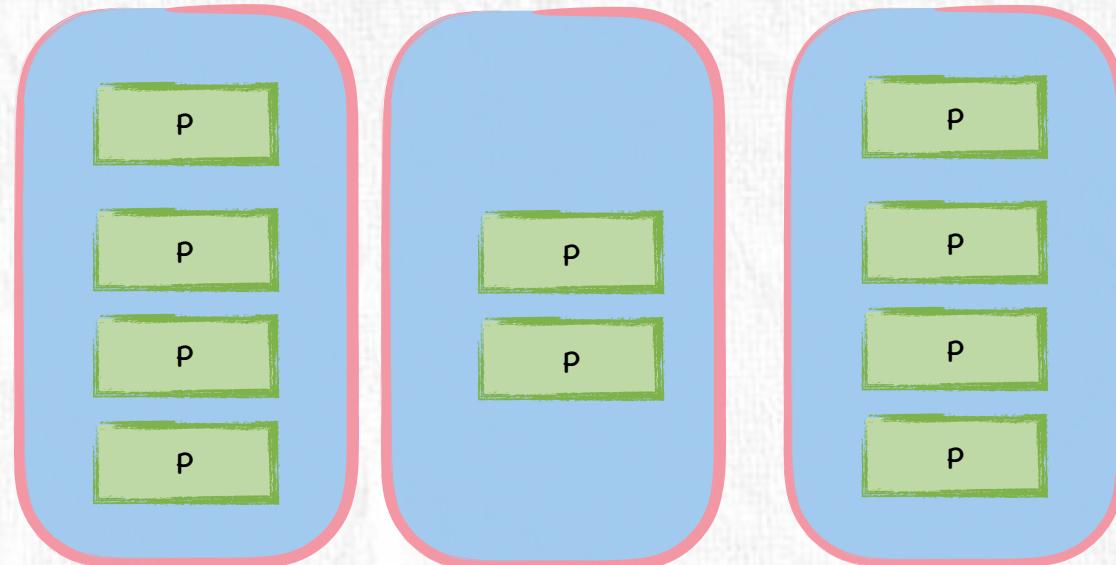


golang in docker

- * golang默认P的数量为取cpu core
- * docker内cpuinfo为宿主机配置
- * P数的增多会增加runtime消耗
- * 根据docker的cpu-quota来动态配置P



<http://github.com/uber-go/automaxprocs>



cpu 64 core

```
root@a4f33fdd0240:/# cat /proc/cpuinfo | grep "processor" | wc -l  
64
```



panic: send on closed channel

- * 现状

- * 每个client会有读写协程及chan

- * 问题

- * 当用户关闭退出时，如何清理回收？

- * 方法

- * 不主动关闭channel
 - * 关闭context通知
 - * 解绑 topic-> client对应关系及删除

```
type StreamNotifier struct {
    Guid string

    InQueue chan interface{}
    OutQueue chan interface{}

    closed int32
    ctx    context.Context
    cancel context.CancelFunc
}

func (sc *StreamNotifier) IsClosed() bool {
    if sc.ctx.Err() == nil {
        return false
    }
    return true
}
```

高可用性

- * if push-router crash ?
 - * 多个router由envoy-ingress负载均衡
- * if push-gateway crash?
 - * push-router会得知健康状态
 - * 客户端从push-router获取可用的gateway
- * if push-server crash ?
 - * gateway从router选择最优push-server
 - * 下发行情订阅请求
 - * 通过上次的ack id下发用户订单订阅

内核优化



- * 开启bbr拥塞控制算法
 - * 国内不明显
 - * 外国效果明显
- * 软中断优化
 - * 多队列网卡
 - * 绑定cpu中断

各类优化

- * 必须注意锁竞争的问题
- * 使用sync.Pool缓存频繁的堆对象
- * 使用bytes.Buffer解析协议
- * 预先设定slice、map的size
- * 减少系统调用
- * 优化defer的调用
- * 通过pipeline提高各端的效率
- * 协程池
- * 控制并发
- * 消除毛刺
- * 减少more stack



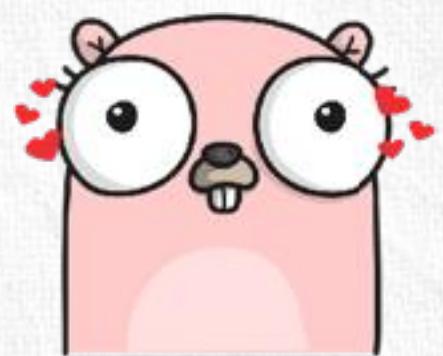
3

排坑记



大坑

- * grpc-web
 - * 不支持bidi mode
 - * 需要envoy做协议转换
- * goroutine per connect模式造成协程过多
- * kcp的表现并不美好

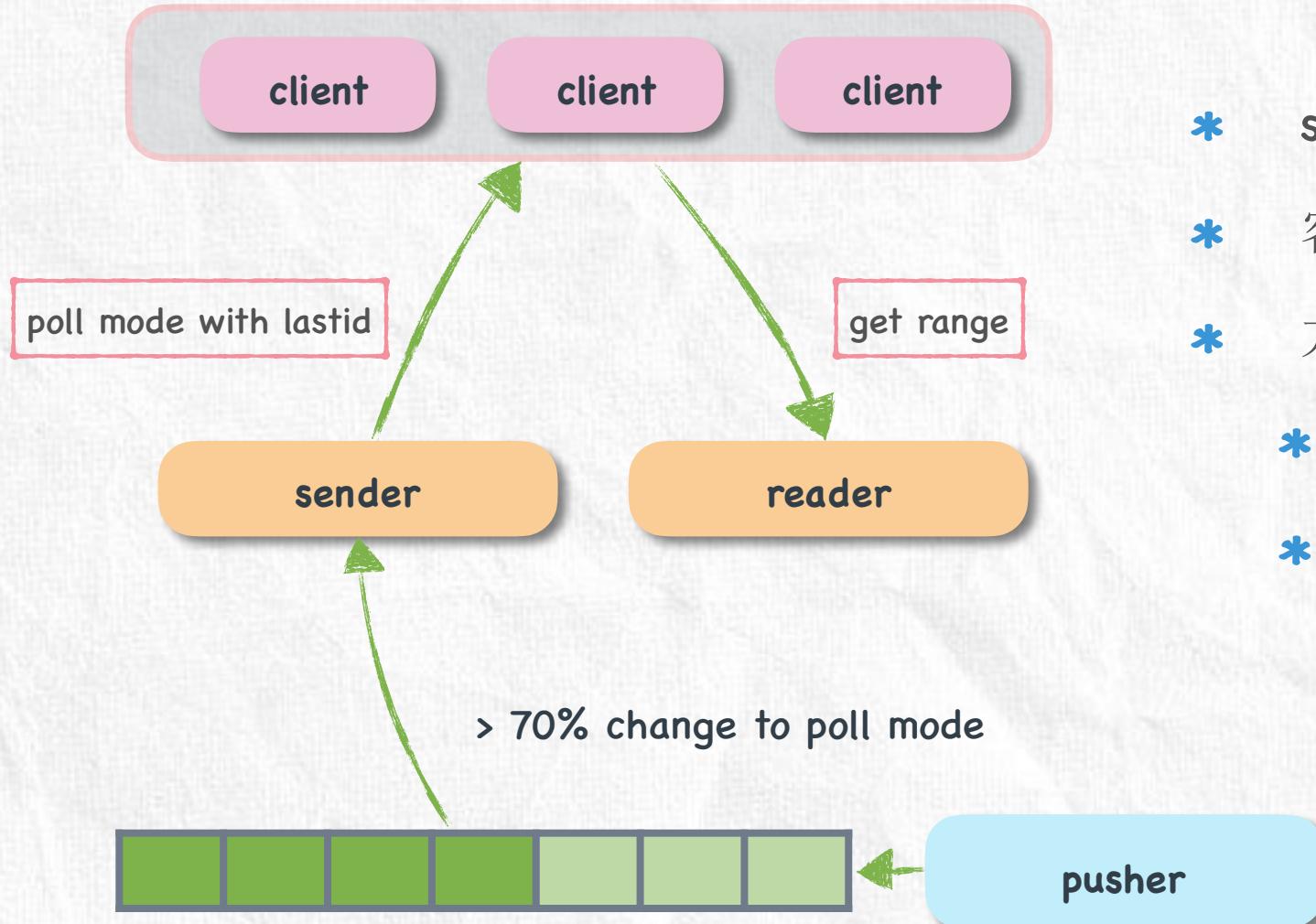


小坑

- * runtime gFree & allgs
- * runtime开销
 - * 过多的退出协程
 - * 过多的休眠协程
- * gc, sysmon, deadlock check ...



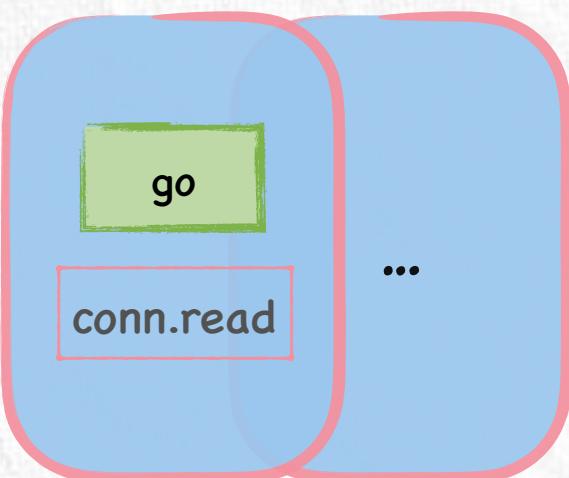
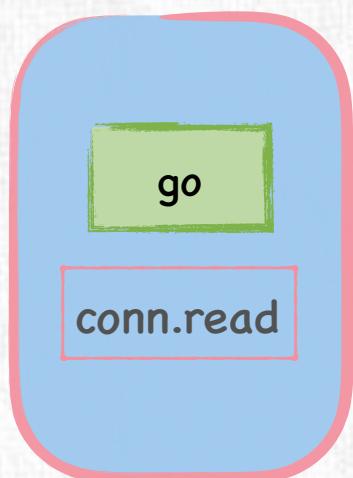
慢客户端问题



- * send msg -> chan -> go -> grpc.stream
- * 客户端因弱网络导致chan满而阻塞
- * 方法
 - * 推送模式改为半推半拉模式
 - * 重置缓冲，重新开始



netpoll vs raw epoll



netpoll in golang runtime



conn.read

conn.write

raw epoll



4

A large yellow starburst shape containing the number '4'.

总结



deploy

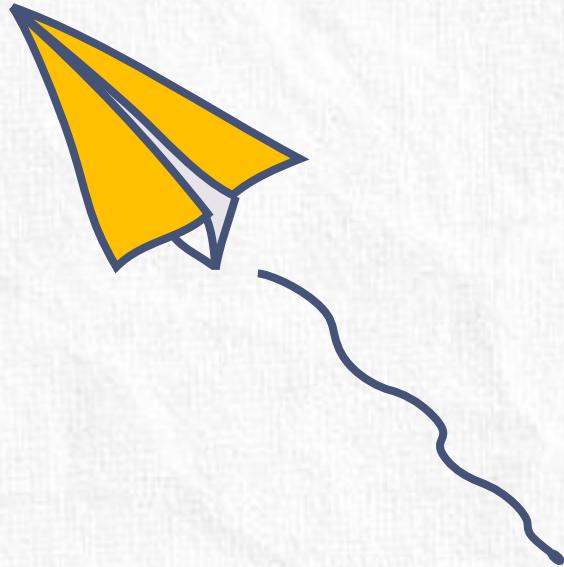
- * 使用drone pipeline持续集成
- * 使用k8s为编排引擎
 - * 设定节点的亲和非亲和策略
 - * 为更好的灵活部署, 开发Operator控制器
- * 规避网络链路长引起性能及时延问题
 - * push-gateway直接使用hostport模式
 - * 减少ingress或iptables带来的延迟损耗。
- * push-server的service为headless模式



测试 & 压测

- * 分布式压测以保证压力源
- * 使用tcpdump导入并放大线上流量到测服集群
- * 使用iptables和tc模拟各类弱网络客户端





Q & A

