目录

- 背景
- 设计
- 实现
- 最佳实践
- 效果展示
- 总结

目录

- 背景
- 设计
- 实现
- 最佳实践
- 效果展示
- 总结

- 微信内部队列使用场景
- 旧队列介绍
- 旧队列问题

微信内部队列使用场景

业务解耦
- 发布订阅模式
- 消息总线

分布式事务
- 拆分为多个本地事务
- 可靠消息传递

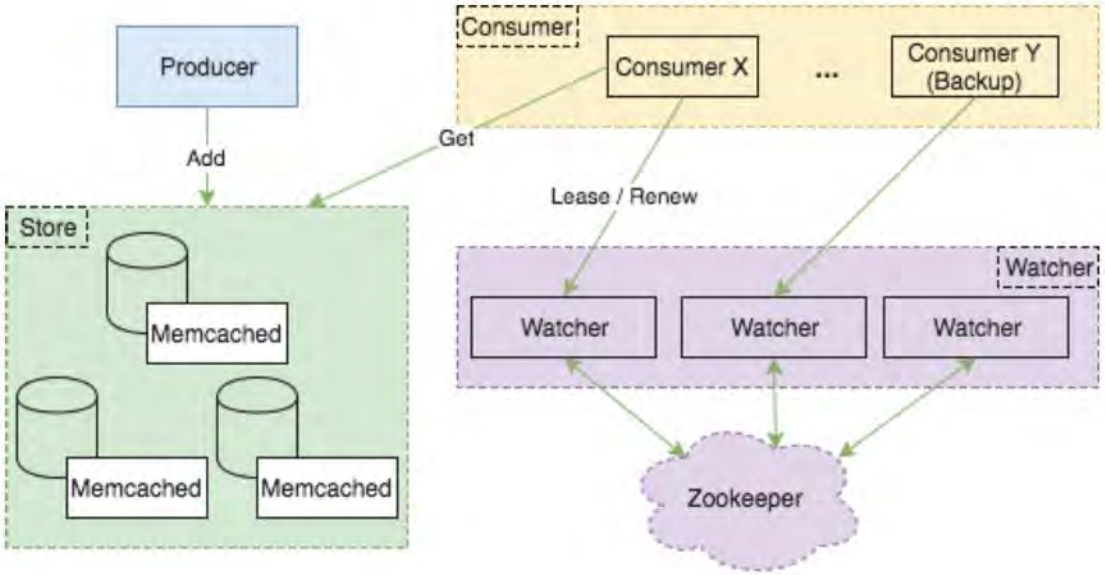
削峰和流控
- 缓存突发消息
- 按能力消费

延迟消费
- 定时任务 - 延后推
- 离线任务 - 随时拉

目前PhxQueue广泛支持微信支付、公众平台等多个重要业务。

日均入队达千亿，分钟入队峰值达一亿。

旧架构及存在问题



典型分布式队列架构

- Producer – 生产者
- Store – 存储(NRW)
- Consumer – 消费者
- Zookeeper – 分布式锁

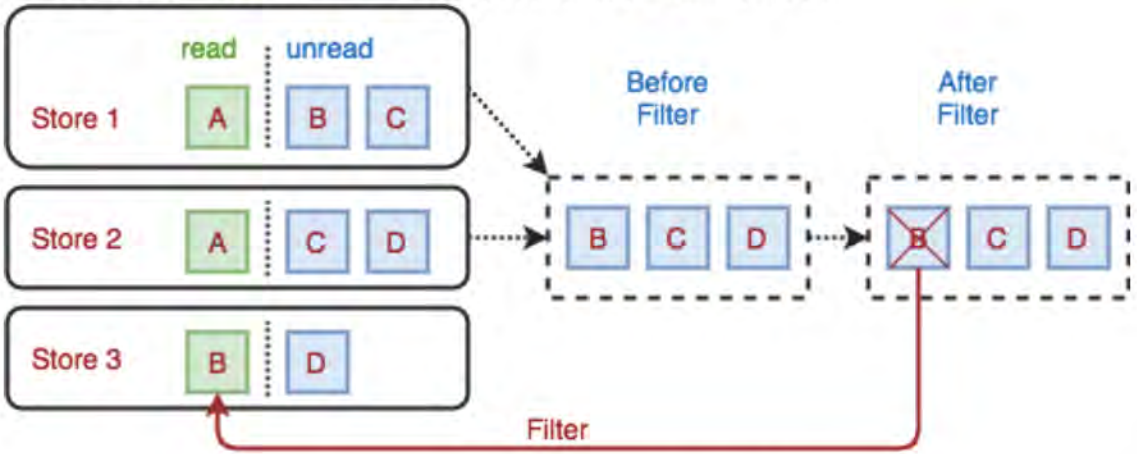
旧架构及存在问题

Store 读写

- N=3, W=2, R=2

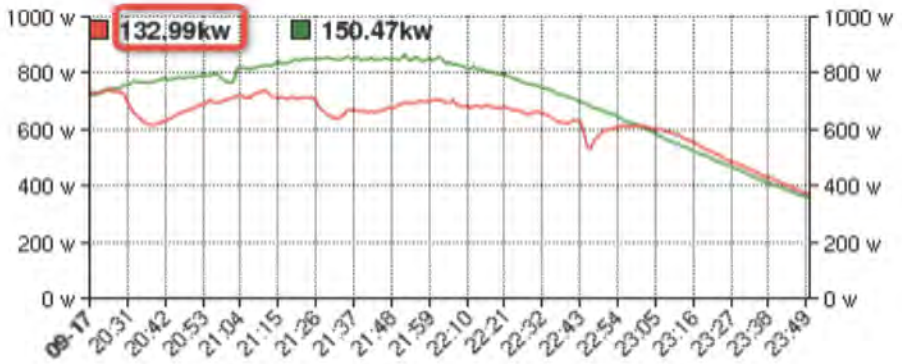
Store 读去重

- Filter=3
- Memcached 维护数据MD5到其偏移的对应关系

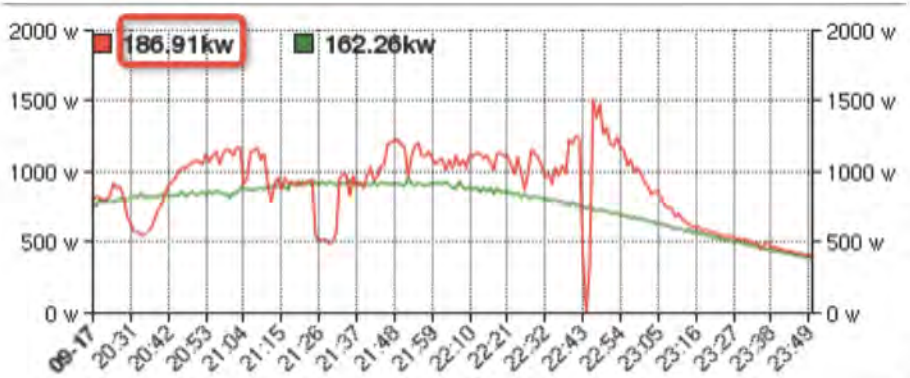


旧架构及存在问题

1. 积压下去重失效
   - 重复消费率达40.9%
   - 原因：
     - NRW需依赖Memcached去重
     - Memcached满容量则去重失效
   - 影响：
     - 降低积压清理效率
     - 部分业务幂等失效



（图1）入队数



（图2）出队数

旧架构及存在问题

2. 出队乱序
   - 出队乱序率达3.37%
   - 原因：NRW天然乱序缺陷
   - 影响：影响用户体验

（图1）出队数

（图2）出队乱序数

旧架构及存在问题

3. 负载均衡效果差

- 原因：
  - 负载均衡与分布式锁逻辑耦合
  - 主动变更锁会导致一段时间重复出队，不可取，从而限制了均衡时机
- 影响: Consumer单机负载过高影响可用

4. 丢数据风险增加

- 原因：异步刷盘
- 故障概率增加：
  - 机器过保
  - 部分地区无法三DC部署

最高负载Consumer:

服务器名	load	总使用率
mmpayhbastrosched18	3.06	12%
mmpayhbastrosched20	2.83	12%

最低负载Consumer:

mmpayhbastrosched26	0.17	1%
mmpayhbastrosched28	0.19	0%

(图) Consumer负载不均衡

目录

- 背景
- 设计
- 实现
- 最佳实践
- 效果展示
- 总结

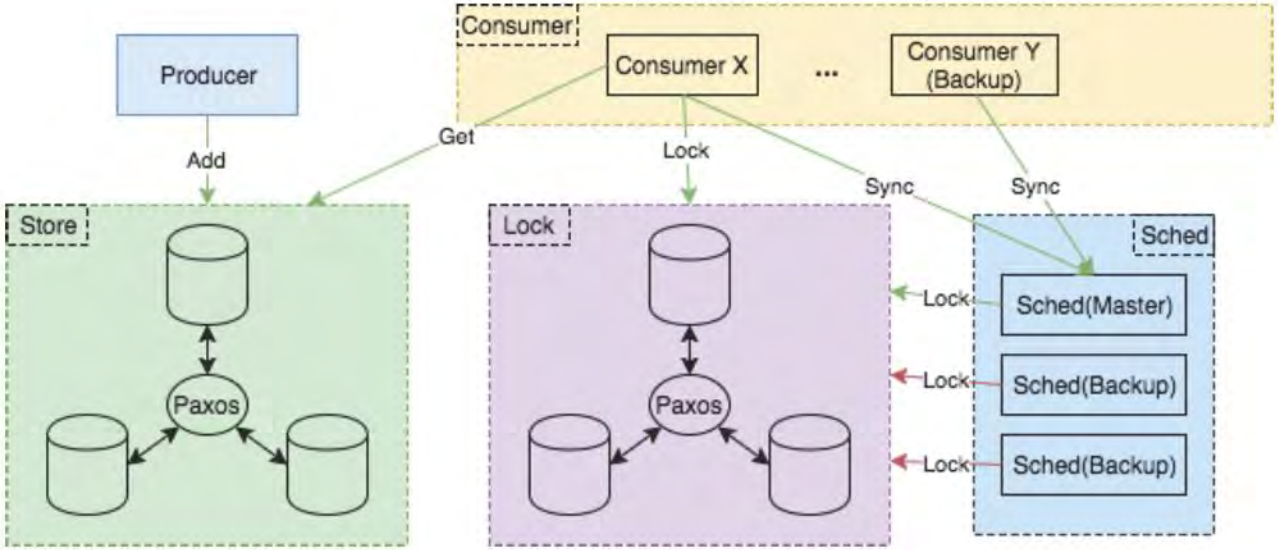
- 拥抱Paxos
- 拆分分布式锁、负载均衡服务

新架构(PhxQueue)设计

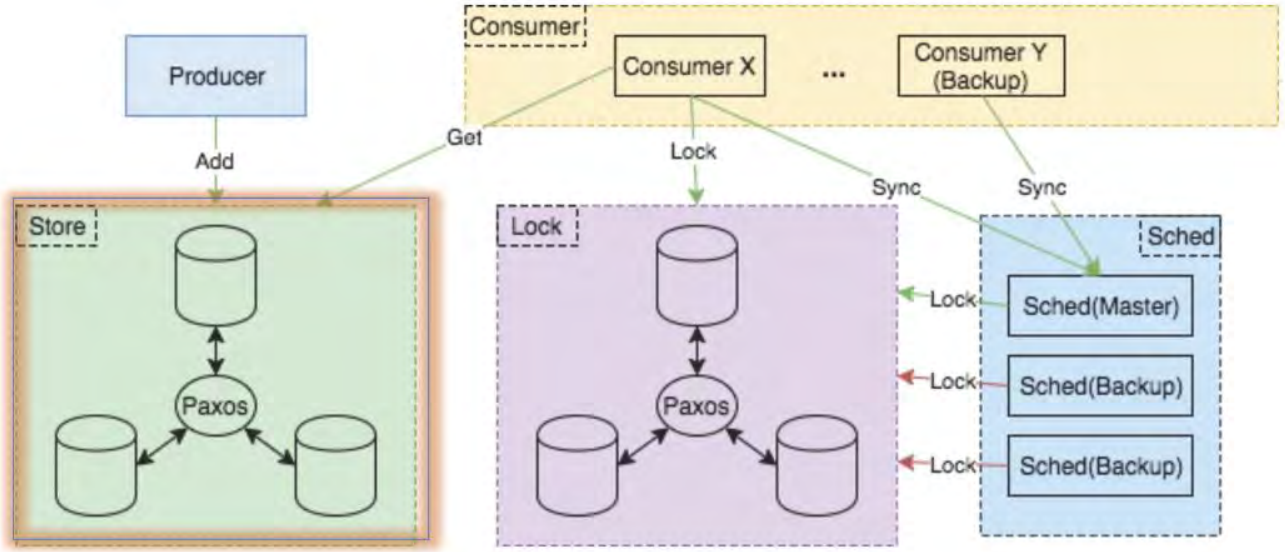
主要特性

- 同步刷盘，入队数据绝对不丢，自带内部实时对账
- 出入队严格有序
- 多订阅
- 出队限速
- 出队重放
- 所有模块均可平行扩展
- 存储层批量刷盘、同步，保证高吞吐
- 存储层支持同城多中心部署
- 存储层自动容灾/接入均衡
- 消费者自动容灾/负载均衡

新架构(PhxQueue)设计



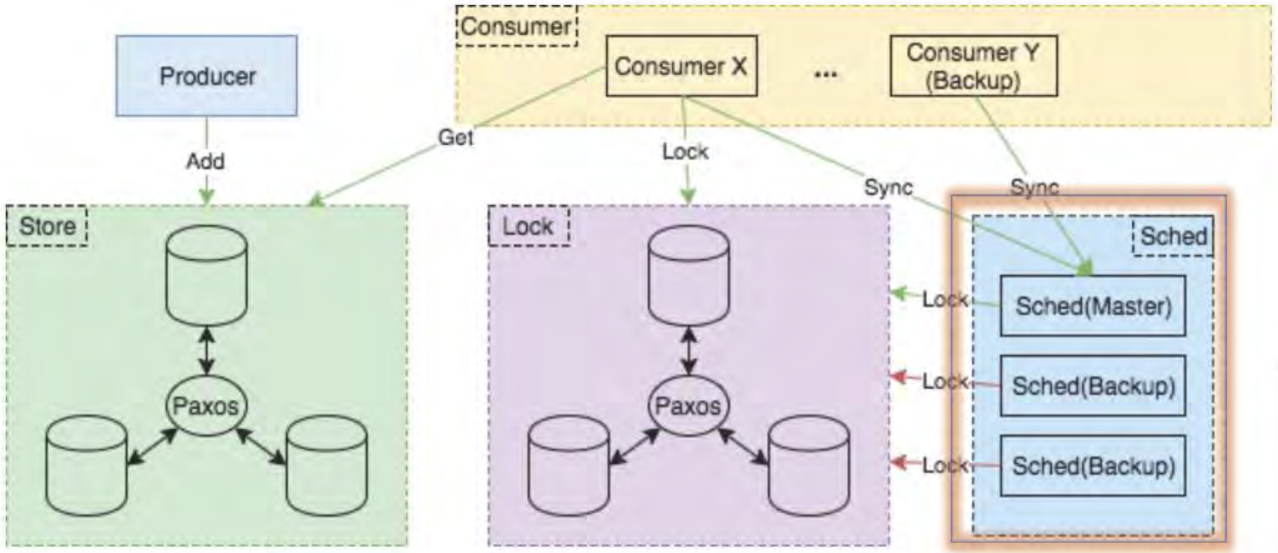
新架构(PhxQueue)设计



Store – 队列存储

- 引入Paxos
- 同步刷盘
- 对外强一致
- 无乱序，免去重

新架构(PhxQueue)设计



Store – 队列存储

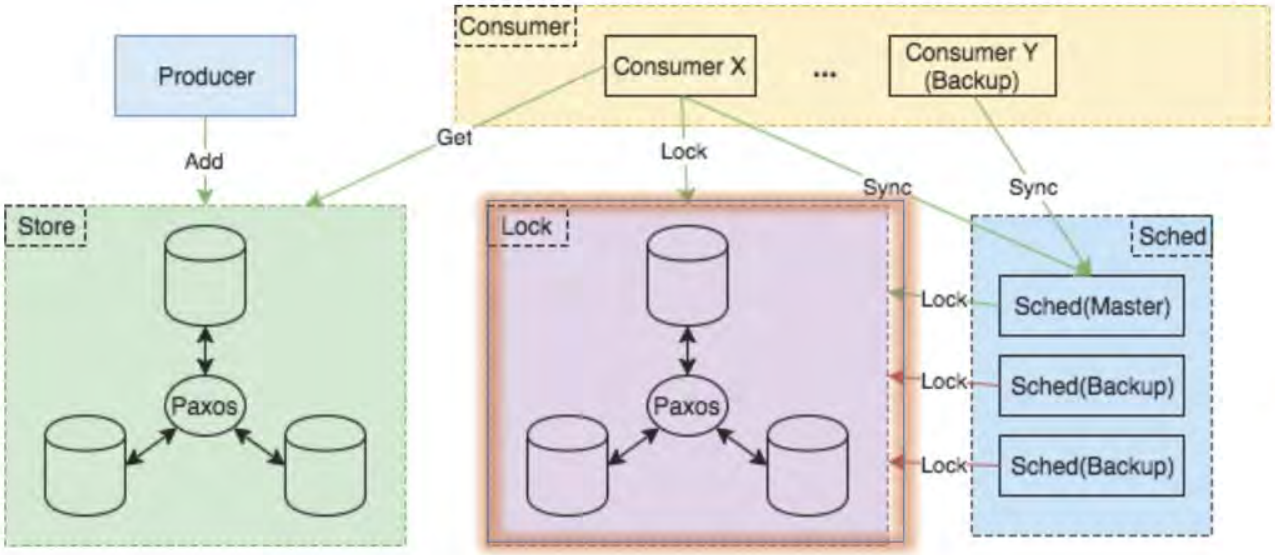
- 引入Paxos
- 同步刷盘
- 对外强一致
- 无乱序，免去重

Sched – 消费者管理

- 消费者容灾
- 消费者负载均衡
- 单机(master)提供服务

1. master挂掉可自行切换
2. master选举过程中仅负载均衡功能失效,整个系统不对Sched强依赖.

新架构(PhxQueue)设计



Store – 队列存储

- 引入Paxos
- 同步刷盘
- 对外强一致
- 无乱序，免去重

Sched – 消费者管理

- 消费者容灾
- 消费者负载均衡
- 单机(master)提供服务

Lock – 分布式锁

- 引入Paxos
- 通用分布式锁服务
- 负责Sched master选举
- 规避队列重复出队

目录

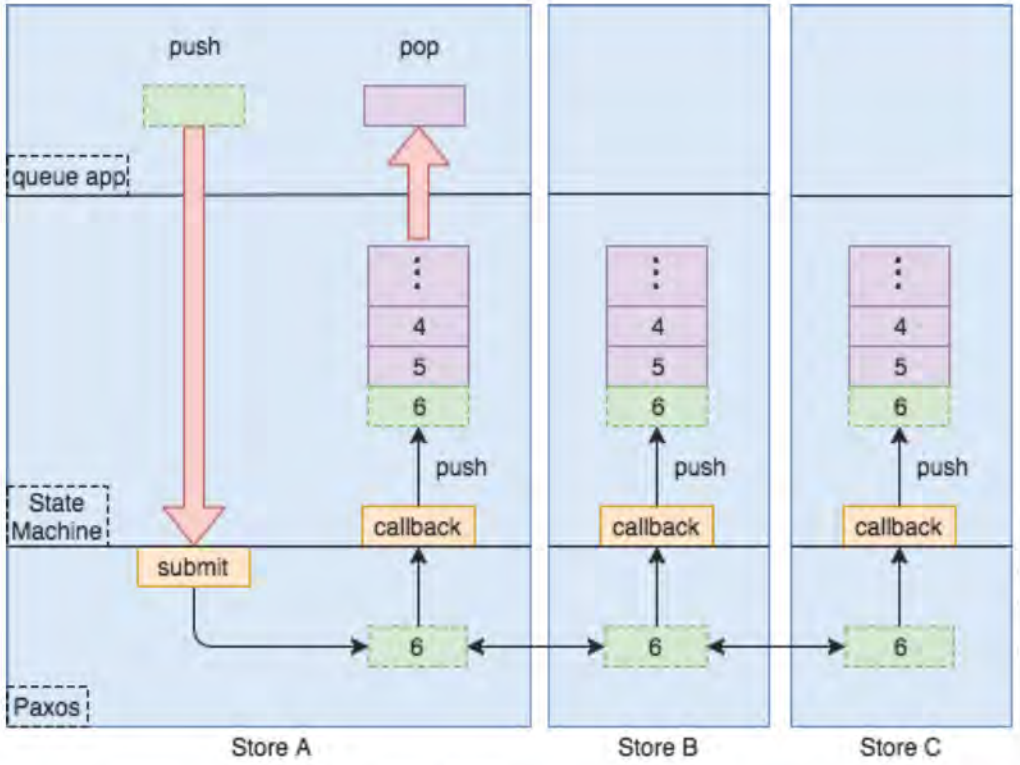
- 背景
- 设计
- 实现
- 最佳实践
- 效果展示
- 总结

- 高可靠
  - 队列状态机设计
- 高性能
  - Plog as queue
  - Group Commit
- 高可用
  - Store租约型读写
  - Store接入均衡
  - 轻量级租约维护
  - Consumer负载均衡

高可靠 - 队列状态机设计

概念映射

队列概念	PhxPaxos 概念
队列数据	paxos log
队列偏移	instance id
队列最小读偏移	check point



极端状况下数据不丢

- Paxos协议的正确运行
- 简单拜占庭检测手段

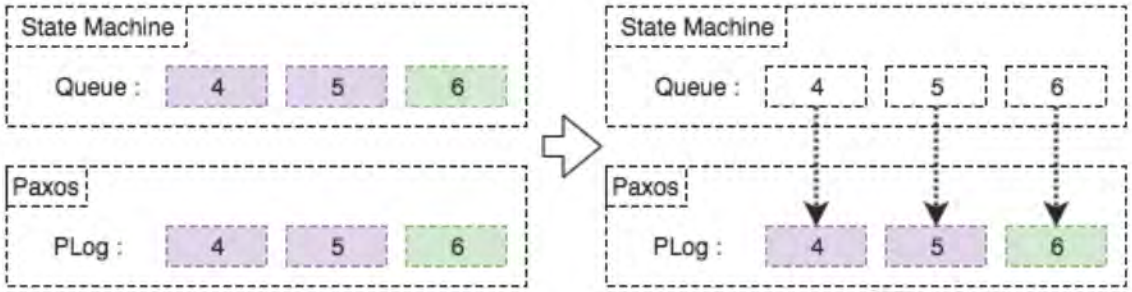
高性能(1/2) – Plog as queue

问题: 副本数从2提高至6

(写量大, 存储成本高)

解决方案: PLog as Queue

效果: 存储减半, 写带宽减半



高性能(2/2) – Group commit

问题1: 同步刷盘写放大 (1.5k -> 4k)

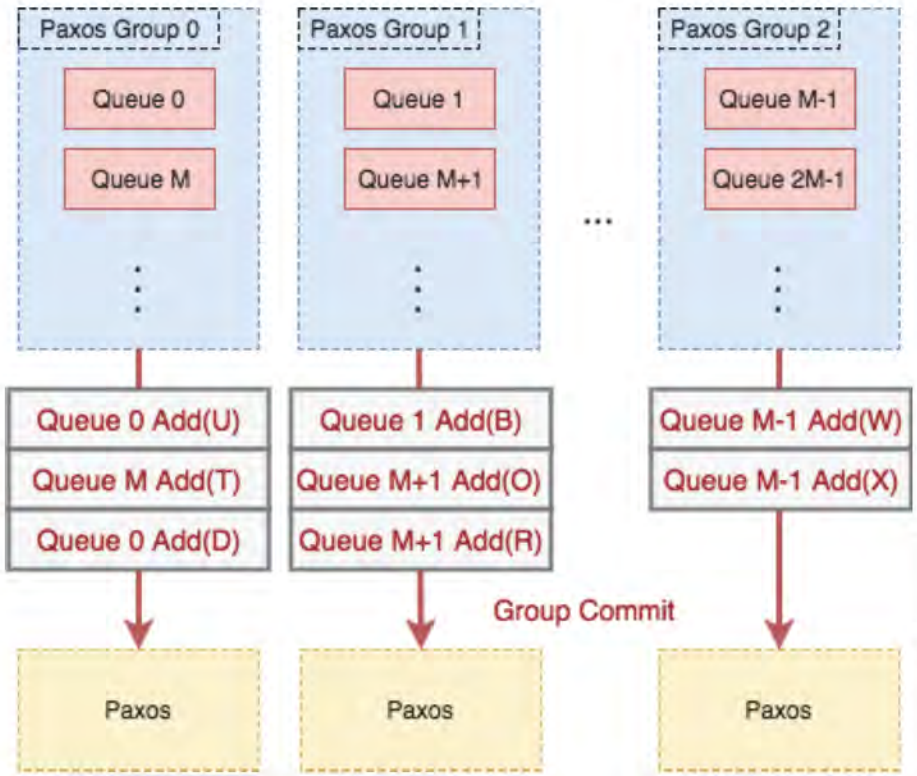
问题2: DC间RTT 4ms, 吞吐低

解决方案: 批量同步刷盘

- Paxos写时机:
  5ms超时 或 请求数达到20
- N Queue : 1 Paxos group
- Paxos group太少 – 并发不足
- Paxos group太多 – 批量效果不足
- 经实测, Paxos group数为100较合适

效果:

- 入队QPS上限50w
- 写放大降低20倍



高可用 (1/4) – Store租约型读写

无租约 or 有租约？

	优点	缺点
无租约	• 可用性高	• 读: 需要访问三机 • 写: 失败时, 需要随机避让避免活锁
有租约	• 读: 只读master • 写: 只写master, 无活锁问题	• master切换有不可用时长

为Store引入租约型读写, 考量如下:

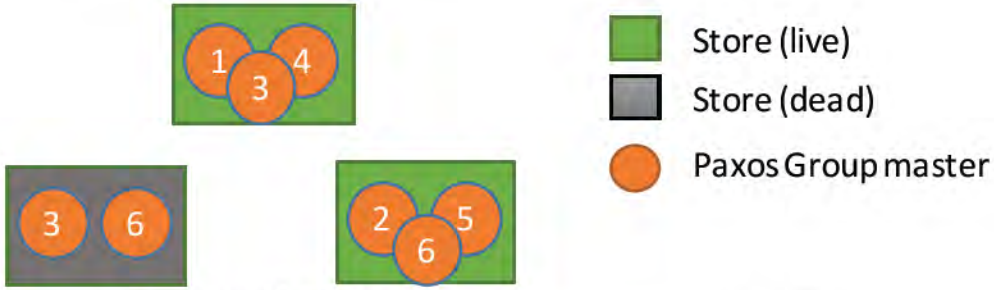
- 即使一个Paxos实例只处理一个queue, 写量也很大, 随机避让严重影响吞吐
- 在非宕机情况下的master切换, 可通过主动结束租约避免不可用时长

高可用 (2/4) – Store接入均衡

问题: 各机请求不均

解决方案: master调度

- 按模摊分master到各机
- 出灾时: 其它节点随机避让抢master
- 恢复时: 按成功率抢回自己负责的master
- 屏蔽时: 主动结束master租约



Master HA 演示动画

高可用 (3/4) – 轻量级租约维护

租约作用：Sched选master / 保证队列消费权唯一

代替Zookeeper的好处

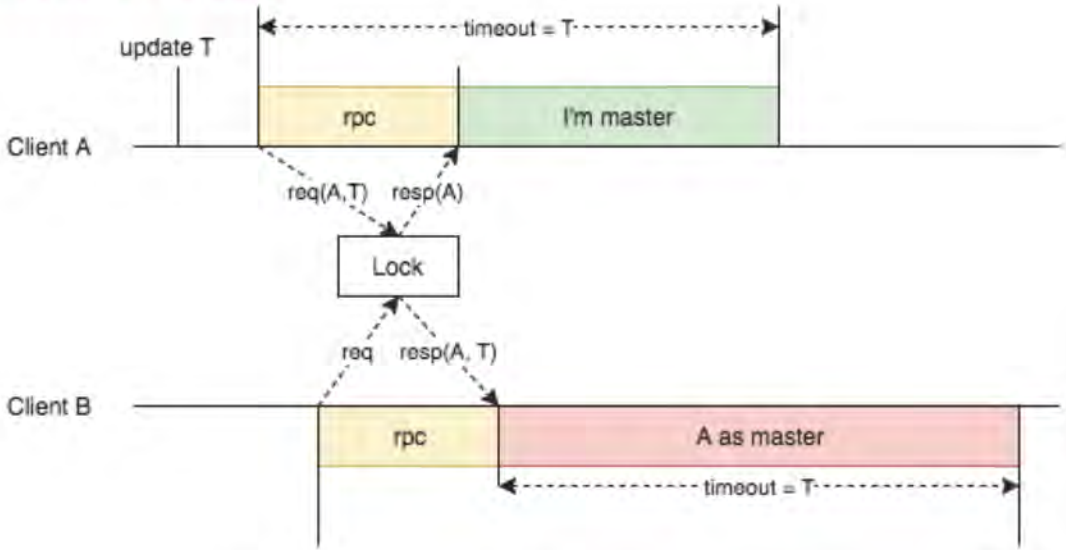
- 简单方案代替复杂黑盒（伪代码不足20行）
- 减低运维、使用成本

难点: 协议设计

- master: 租约时长内续租
- 非master: 避让租约
- 乐观锁

特点: 无死锁

- 可重入
- 过期自动失效



高可用 (4/4) – Consumer负载均衡

Consumer权重式负载均衡

- Sched收集Consumer负载更新权重
- Consumer按权重计算目标处理队列

For each consumer:

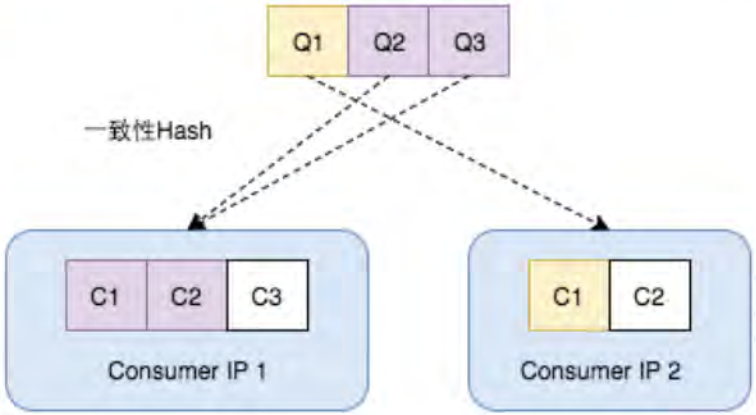
new_weight ↓ =

default_weight * mean_load / load ↑;

问题1: 轻微调整引起全部Consumer抖动

解决方案: 一致性hash

效果: 避免惊群效应, 比Kafka更稳定



问题2: 均衡时存在不可用时长

解决方案: 弃租锁时主动结束租约

效果: 不可用时长从最长20s(租约)降至最长1s

目录

- 背景
- 设计
- 实现
- 最佳实践
- 效果展示
- 总结

- 队列与业务的相互协调
- 主动屏蔽
- 基于接口调用统计的后向流控

队列与业务的相互协调

业务对队列的妥协

- 幂等 - 基于版本号的乐观锁
- 顺序性处理 - 打包有依赖关系的数据

队列对业务的定制化

- 可用性优化 - 换组/队列重试
- 热点处理 – 随机路由避免热点

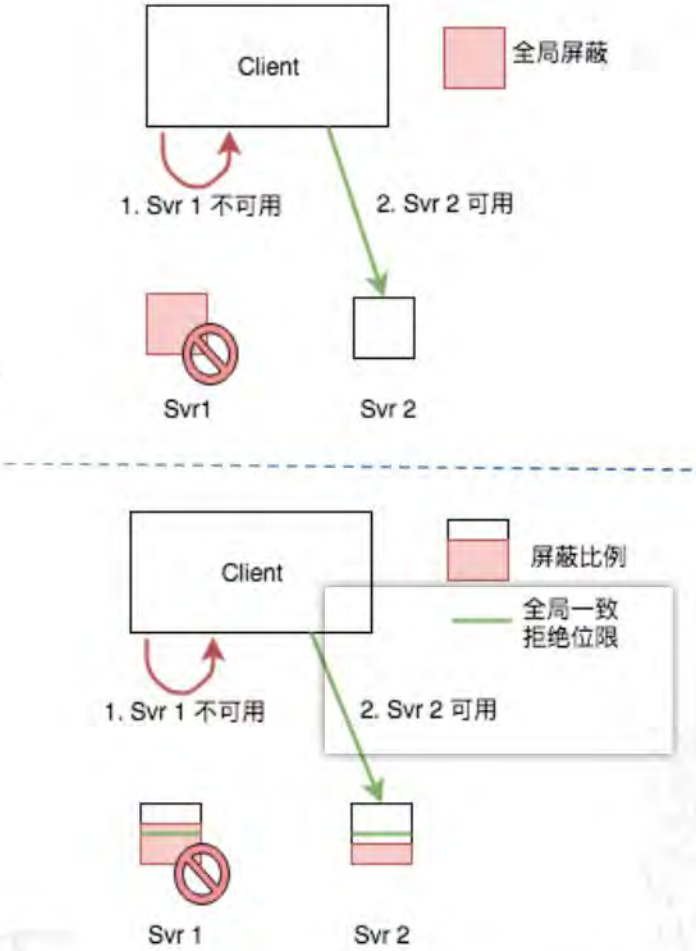
主动屏蔽

目的

- 应对雪崩、爆流量/CPU/内存等异常

手段

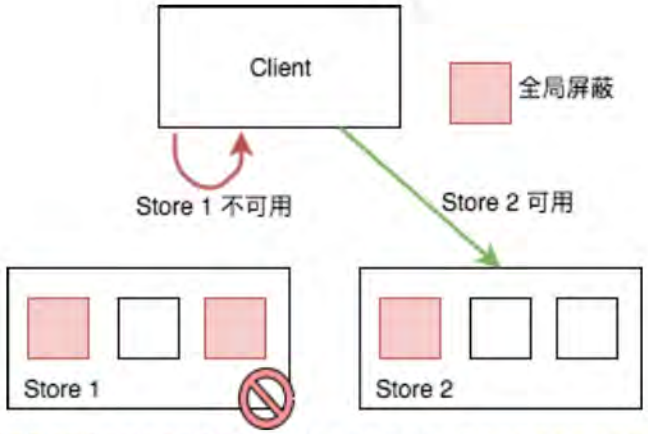
- 全局屏蔽
- 按比例屏蔽



主动屏蔽 - 以组为单位

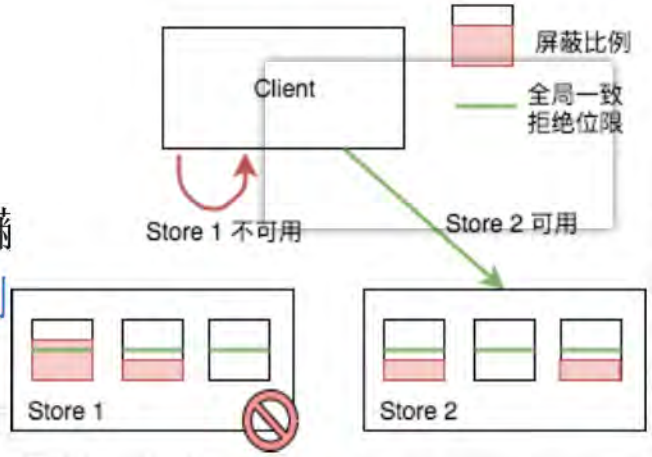
单组全局屏蔽

- 等价于 至少两个节点全局屏蔽



单组按比例屏蔽

- 等价于 该uin至少命中一个节点按比例屏蔽
- 单组屏蔽比例 = 组内最高单节点屏蔽比例



基于接口调用统计的后向流控

目的

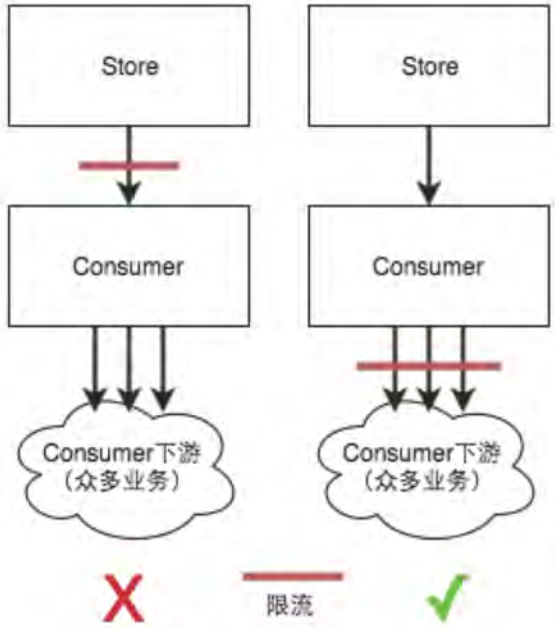
- 削峰，保护Consumer下游

控制拉取数量 X

- 消费一个请求被扩散成多个调用
- 扩散比不稳定

控制接口调用 ✓

- 粒度：业务 + 接口
- 借助RPC API统计接口调用数
- 设定周期内调用上限（天花板）
- 均匀周期内调用：计算拉取数量和间隔



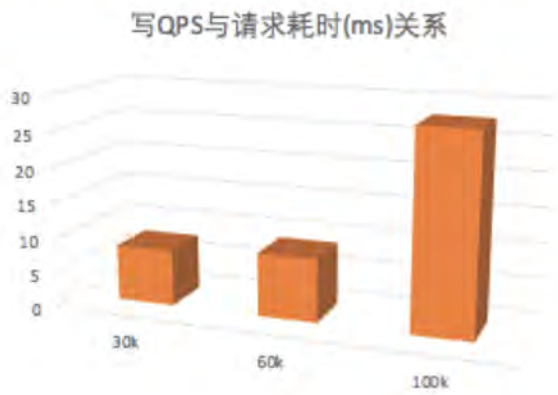
目录

- 背景
- 设计
- 实现
- 最佳实践
- 效果展示
- 总结

- 高性能 – 吞吐表现
- 高可用 – 负载均衡
- 顺序消费
- 基于接口调用统计的后向流控

效果展示 – 高性能 – 吞吐表现

副本数	3
刷盘模式	同步
机型	64*2.4GHz, SSD Raid 10
Buffer大小	2K
读写模型	写+读



- 效果：耗时换吞吐

效果展示 – 高可用 – 负载均衡

旧队列

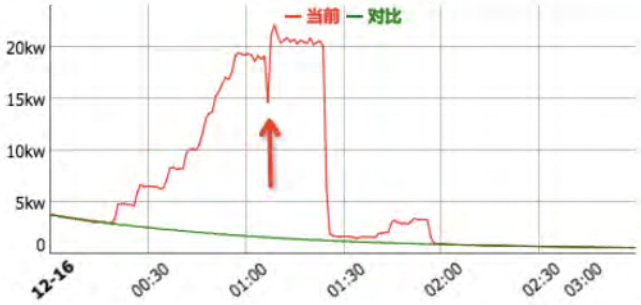
	服务器名	load	总使用率
最高负载	mmpayhbastrosched18	3.06	12%
	mmpayhbastrosched20	2.83	12%
最低负载	mmpayhbastrosched26	0.17	1%
	mmpayhbastrosched28	0.19	0%

PHXQueue

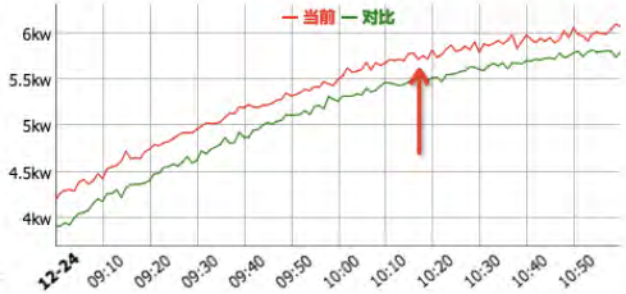
服务器名	load	总使用率
mmphxqueuesnsconsumerml51	1.41	17%
mmphxqueuesnsconsumerml49	1.30	17%
mmphxqueuesnsconsumerml52	1.15	15%
mmphxqueuesnsconsumerml48	1.01	15%

均衡对吞吐影响

无主动弃锁



有主动弃锁



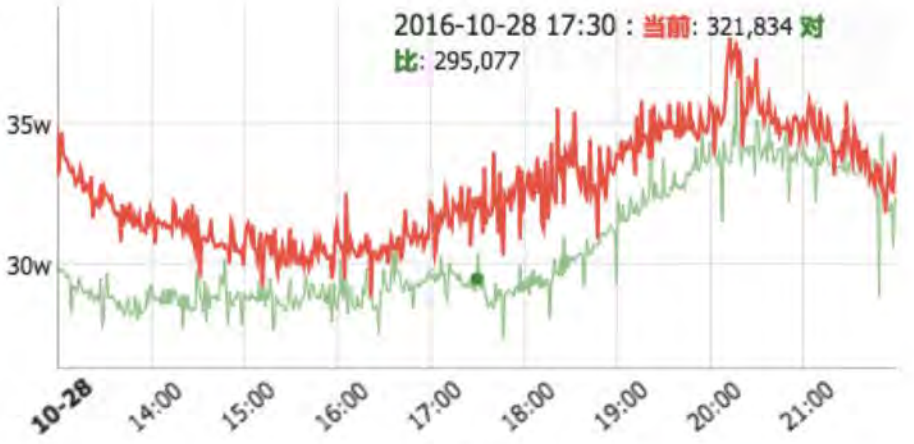
效果展示 – 顺序消费

以某个读多写少的业务为例

- 旧架构乱序率达2.4%
- 新架构将乱序率降低至0.006%

(因重试队列导致乱序)

- 如业务需要绝对顺序消费,可关闭重试队列



总请求数

乱序数