



GOPS 2016
Shanghai



全球运维大会

2016

重新定义运维

上海站

会议时间： 9月23日-9月24日

会议地点： 上海·雅悦新天地大酒店

主办单位：



指导单位：



唯品会大规模Redis存储架构演进

陈群 唯品会



个人简介

- 2014.3加入唯品会DBA团队，资深数据库工程师
- 主要负责Redis/Hbase/Kafka集群运维和开发支持
- 关注MySQL、NoSQL、大数据以及分布式存储等技术
- 个人博客：DBA的罗浮宫 (www.mdba.cn)



目录



1 应用场景

2 架构演进

3 运维实践

4 服务治理



Redis

KV存储

数据结构

全内存

持久化

主从复制

集群模式



<http://redis.io>

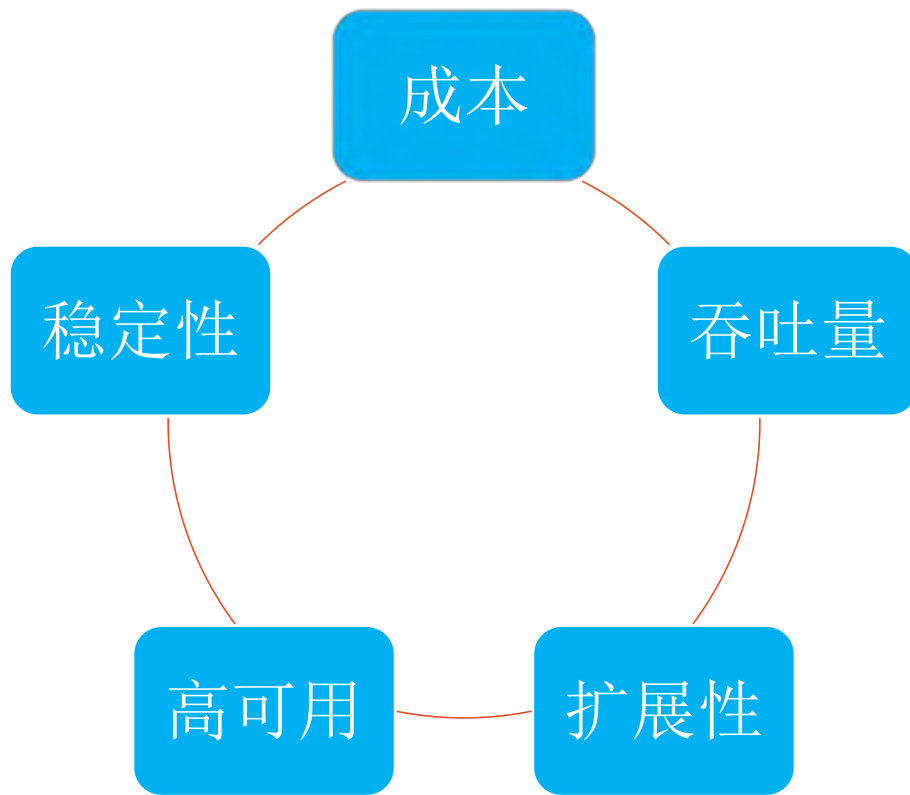


redis在线应用

缓存	专题	商品	档期	收藏	评论
队列	活动	红包	会员	订单	支付
存储	风控	营销	客服	物流	仓储
	运营	报表	模型	分流	推荐



挑战



目录

1 应用场景



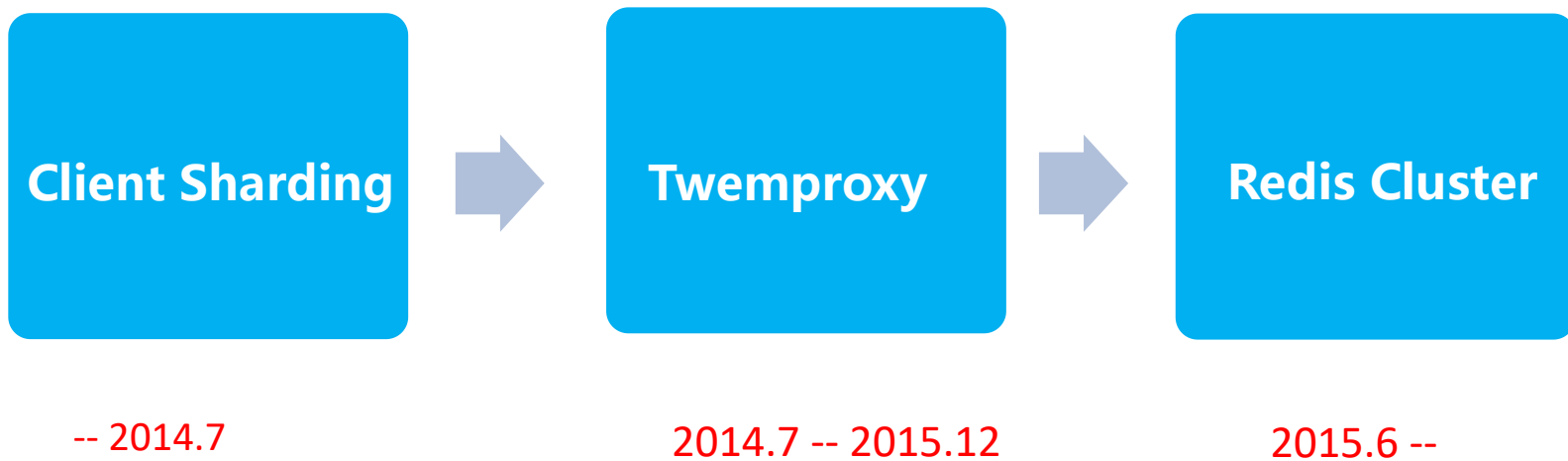
2 架构演进

3 运维实践

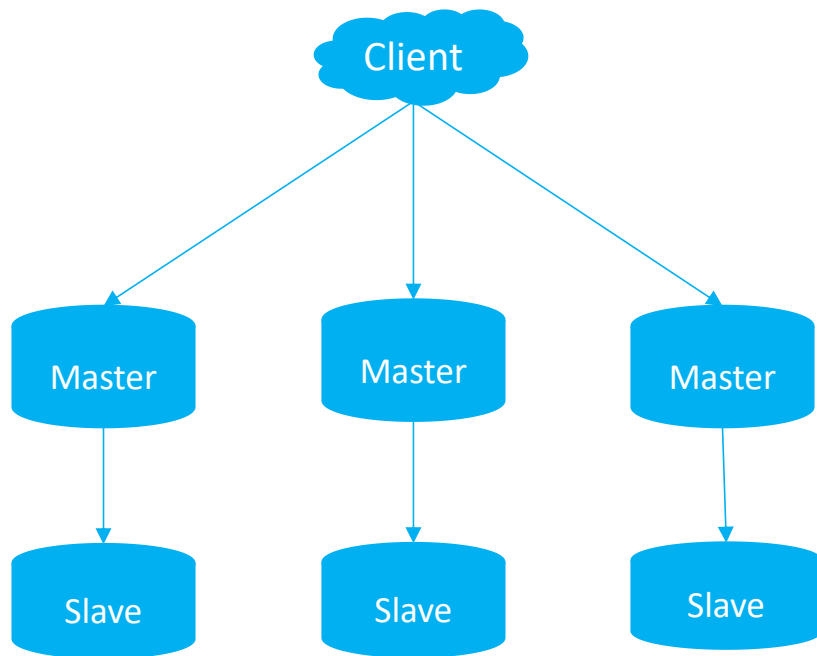
4 服务治理



演进之路



Client Sharding

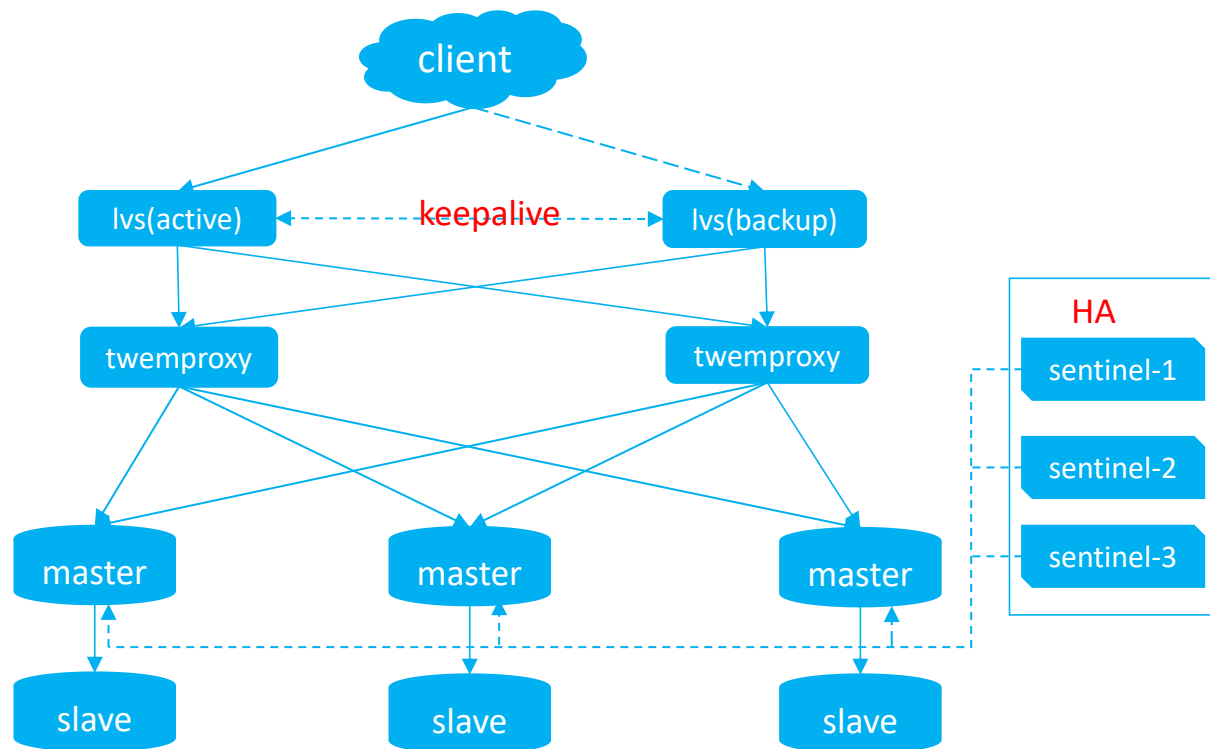


Client Sharding

- 自定义数据分布规则
- 无在线扩容能力
- 扩容过程对应用不透明
- 需要开发高可用方案



Twemproxy



Twemproxy

- 提供数据分片算法，包括一致性哈希
- 兼容redis/mc协议和大部分redis命令
- 支持pipeline
- 支持mset/mget操作
- 架构复杂，成本较高
- Redis层无法在线扩容
- twemproxy缺陷
- lvs层瓶颈，改用ospf模式？



三层架构

1、为什么使用LB/LVS?

- 方便管理，部署和监控
- 依赖lvs做健康检查，剔除问题节点
- 权重调整，用于压测

2、如何解决Twemproxy的扩容能力不足？

- 预分配充足的节点
- 改造Twemproxy，支持节点替换



去三层架构

- 主推redis cluster架构
- 架构简化
- 节约大量机器成本
- 运维管理简单化
- 系统瓶颈更少
- 大幅度性能提升



Tips

- 注意twemproxy不支持redis操作
- 合理设置twemproxy请求redis的timeout参数
- 正对缓存和存储服务，分别设置redis eject策略
- 根据数据大小设置mbuf的大小
- pipeline请求不宜过大，过大导致twemproxy申请大量的内存空间
- 本地化部署策略，和应用部署在一起

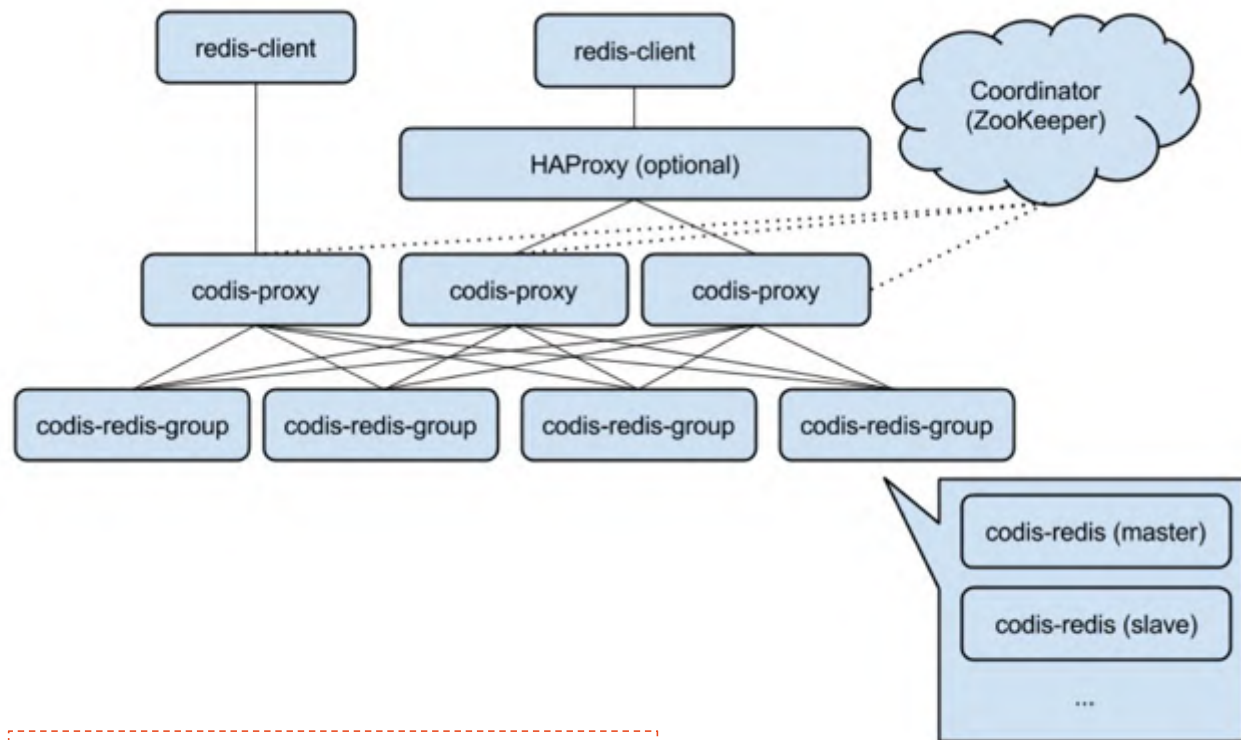


二次开发

- 支持在线替换redis节点，完成redis迁移。
- 增加连接keepalived，解决tcp连接不释放问题。
- 多线程版本twemproxy，节约了千万的机器成本。
- 开源：<https://github.com/vipshop/twemproxy-vip>
- Twemproxy与异构集群之间的数据迁移。
- <https://github.com/vipshop/redis-migrate-tool>



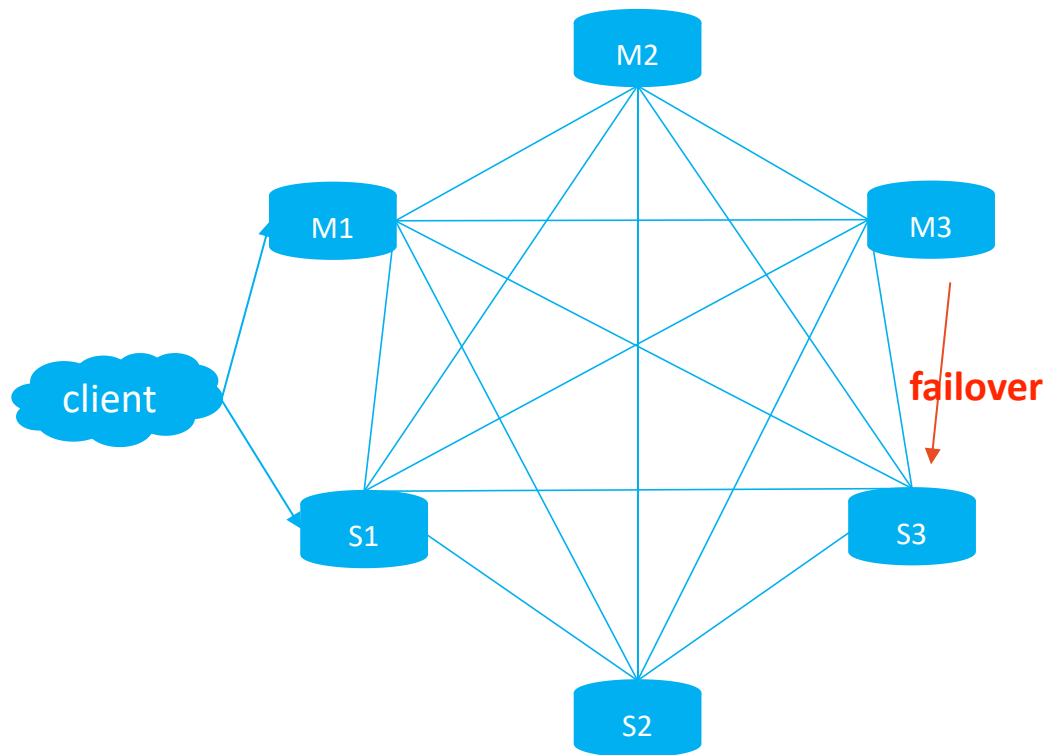
为什么不实用Codis ?



来源: <https://github.com/CodisLabs/codis>



Redis Cluster



Redis Cluster

- 无中心架构
- 数据按照slot存储分布在多个redis实例上
- 增加slave做standby数据副本，用于failover，集群快速恢复
- 实现故障auto failover
- 亦可manual failover，为升级和迁移提供可操作方案
- 降低硬件成本和运维成本，提高系统的扩展性和可用性



架构对比

	Client Sharding	Twemproxy	Redis Cluster
架构	• 多个独立的实例 • 主从复制 • 客户端实现分片	• twemproxy完成分片 • twemproxy仅转发请求和处理响应 • 可以部署在应用服务器，或者dns轮询 • 或者Twemproxy上加负载均衡，简化开发 • 层次多	• 无中心架构 • server端实现分片(crc32) • 主从复制 • 强依赖smart client
兼容性	• 依赖具体客户端实现	• 兼容大部分redis命令 • 支持mget/mset • 支持pipeline	• 依赖具体客户端实现
成本	• 低 • 增加开发复杂度	• 机器数量多 • 管理和维护成本高	• 低
扩展性	• 扩展性差 • 修改配置和代码，发布变更 • 简单扩容方式：倍增节点	• redis层扩容能力弱	• 扩展能力强 • 可以在线增加/缩容节点
HA	• 需要自己开发HA模块 • Sentinel + Zookeeper/DNS	• 需要自己开发HA模块 • Sentinel + Zookeeper/DNS	• slave 热备 • auto-failover/switchover



目录

1 应用场景

2 架构演进



3 运维实践

4 服务治理



Redis监控

- 进程cpu利用率, why? 单进程
- QPS、命中率
- 内存
- 复制状态
- 客户端连接数



参数优化

- *timeout 180*
- *tcp-keepalive 300*
- *repl-backlog-size 32M* #默认1M, 导致无法pysnc
- *client-output-buffer-limit normal 512mb 256mb 60*
- *client-output-buffer-limit slave 1024mb 256mb 120*



数据持久化

- rdb 还是aof ?
- 主库持久化还是从库 ?
- Fork子线程时，前端出现请求超时。
- 磁盘配置
- 缓存是否需要持久化数据 ?



高性能

- slow query严重影响服务的稳定性。
- 单个请求执行很长时间，其他客户端大量请求超时
- 比如常见的 $O(N)$ 操作，hmget/lrang/keys等
- 管理操作命令，bgrewriteaof/bgsave时fork子线程时造成短暂的写入阻塞
- 隐藏的操作，big-key expired



Redis Cluster

- cluster-node-timeout , 默认15s , 过小造成集群不稳定
- cluster-require-full-coverage , 集群是否可以部分可用 ?
- 建议使用成熟稳定的redis-3.0.7版本 , 3.2.x新增代码较多
- 3.0.x后期版本数据迁移更快 , migrate操作实现单次迁移多个key
- 提醒开发注意客户端驱动参数默认值 , 比如JedisCluster里面
DEFAULT_MAX_REDIRECTIONS = 5



目录

1 应用场景

2 架构演进

3 运维实践



4 服务治理



Cache和Storage

不同地方：

- 使用场景和目的
- 数据读写逻辑和异常处理策略
- 数据一致性和安全性
- 数据持久化、备份、恢复需求

混用或者误用带来的问题：

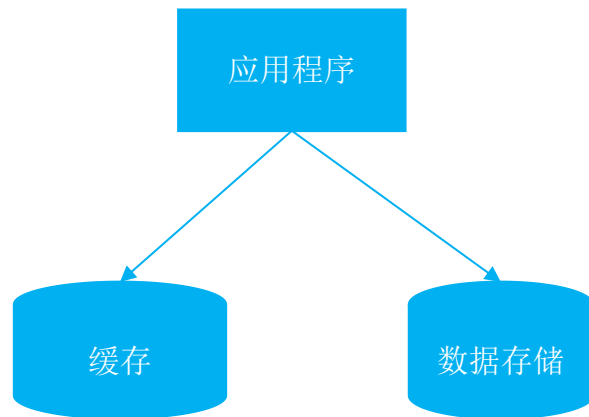
- 1) 数据异常或者丢失
- 2) 资源管理复杂，缓存和存储对硬件配置需求不同



请求回源

常见问题：

- 1) 数据miss或者异常求处理无后续处理逻辑
- 2) db同步cache数据更新策略
- 3) 重试机制
- 4) 回源惊群问题和过载保护机制



多级缓存与数据一致性问题

目的：

- 缓存热点数据
- 减少穿透到DB的请求
- 数据闭环

常见问题：

- 数据不一致
- 数据生命周期
- 数据更新策略
- JVM GC问题



探索

- 探索内存+磁盘存储模式，降低生产成本



Thanks !



GOPS2016 全球运维大会更多精彩

GOPS2016 全球运维大会·北京站

2016年12月16日-17日
北京国际会议中心

