Codis 2.0：从 Cache 到 DB

黄东旭

DTCC

2015中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2015

大数据技术探索和价值发现

关于我

- 黄东旭
- @Dongxu_Huang
- Gopher，脑残谷粉，分布式系统信徒，开源运动实践者

Codis

- 分布式 Redis
- Stateless Proxy-based
- 完全兼容 Twemproxy
- 目前已经有很多知名和不知命的公司用于生产环境
  - 猎豹移动，科大讯飞，豌豆荚，汽车之家...
- 开源, github star 2200+

传统单点缓存

Redis, Memcached

1. 单机内存有限
2. 带宽压力
3. 单点问题
4. 不能动态扩容
5. 磁盘损坏时数据抢救

分布式缓存

1. Twemproxy
2. Redis Cluster (official)
3. Tair, Coachbase, Aerospike
4. Codis

Twemproxy

1. 最早/使用最广泛的解决方案
2. Proxy based
3. 静态的拓扑
4. 运维基本靠手
5. 最大痛点：无法平滑的扩/缩容
    - 甚至修改个配置都需要重启服务 Orz

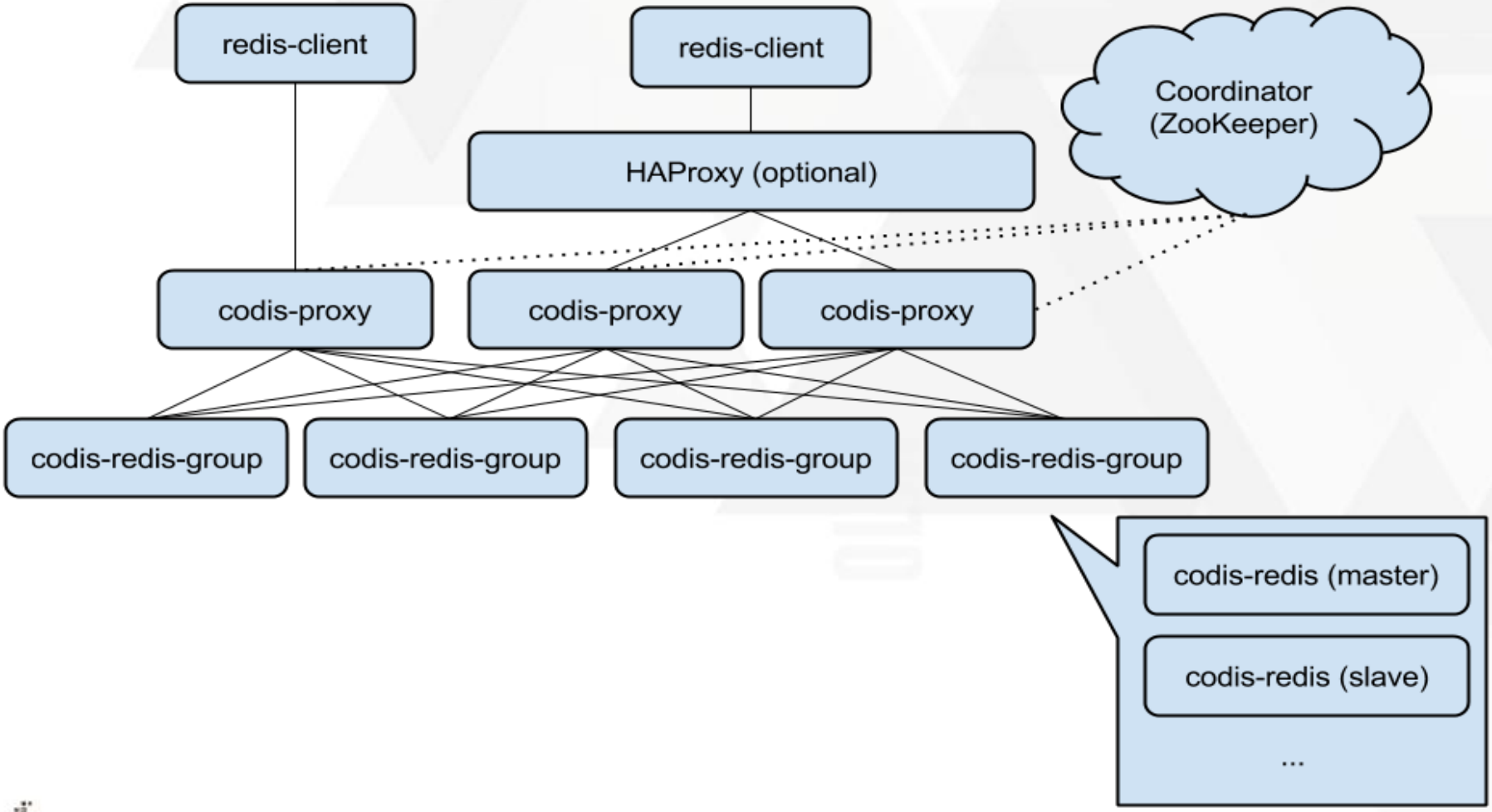
Redis Cluster

1. 官方出品
2. 去中心化设计
3. 客户端需要修改
4. 目前还缺乏 Best Practice，还没有人写一个 Redis Cluster 若干条注意事项
5. 整个系统高度耦合，升级比较困难

Tair, Couchbase, Aerospike…

1. 技术选型
2. 数据结构支持
3. 社区

Codis 1.x


redis-client
redis-client
Coordinator
(ZooKeeper)
HAProxy (optional)
codis-proxy
codis-proxy
codis-proxy
codis-redis-group
codis-redis-group
codis-redis-group
codis-redis-group
codis-redis (master)
codis-redis (slave)
...

Codis 1.x

- 业务不停机，平滑扩/缩容
- 无状态 Proxy，负载均衡，无单点
- 充分利用多核
- 运维工具齐全
  - dashboard
  - redis-port
  - codis-ha

Codis 1.x

缺点(maybe :P) :

- 强依赖 zookeeper
- 修改了官方 redis
- 性能
- MULTI / EXEC 等指令不支持

Codis 整体设计

- Pre-sharding
  - Slot => [0, 1023]
- Zookeeper
- Proxy 无状态
- 平滑扩容/缩容
- 扩容对用户透明

设计考量

- 分布式系统是复杂的
- 开发人员不足
- 尽量拆分，简化每个模块，同时易于升级
- 每个组件只负责自己的事情
- Redis 只作为存储引擎
- Proxy 状态

设计考量

- Redis 是否挂掉的判定放到外部，因为分布式系统存活的判定是复杂的
- 提供 API 让外部调用，当 Redis master 挂掉的时候，提升 slave 为 master
- 我们不喜欢读写分离

设计考量

- graph everything
  - slot status
  - proxy status
  - group status
  - lock
  - action

设计考量

proxy vs smart client

proxy:

更好的监控，控制

后端信息不暴露，易于升级

smart client:

更好的性能

更低的延迟，升级比较麻烦

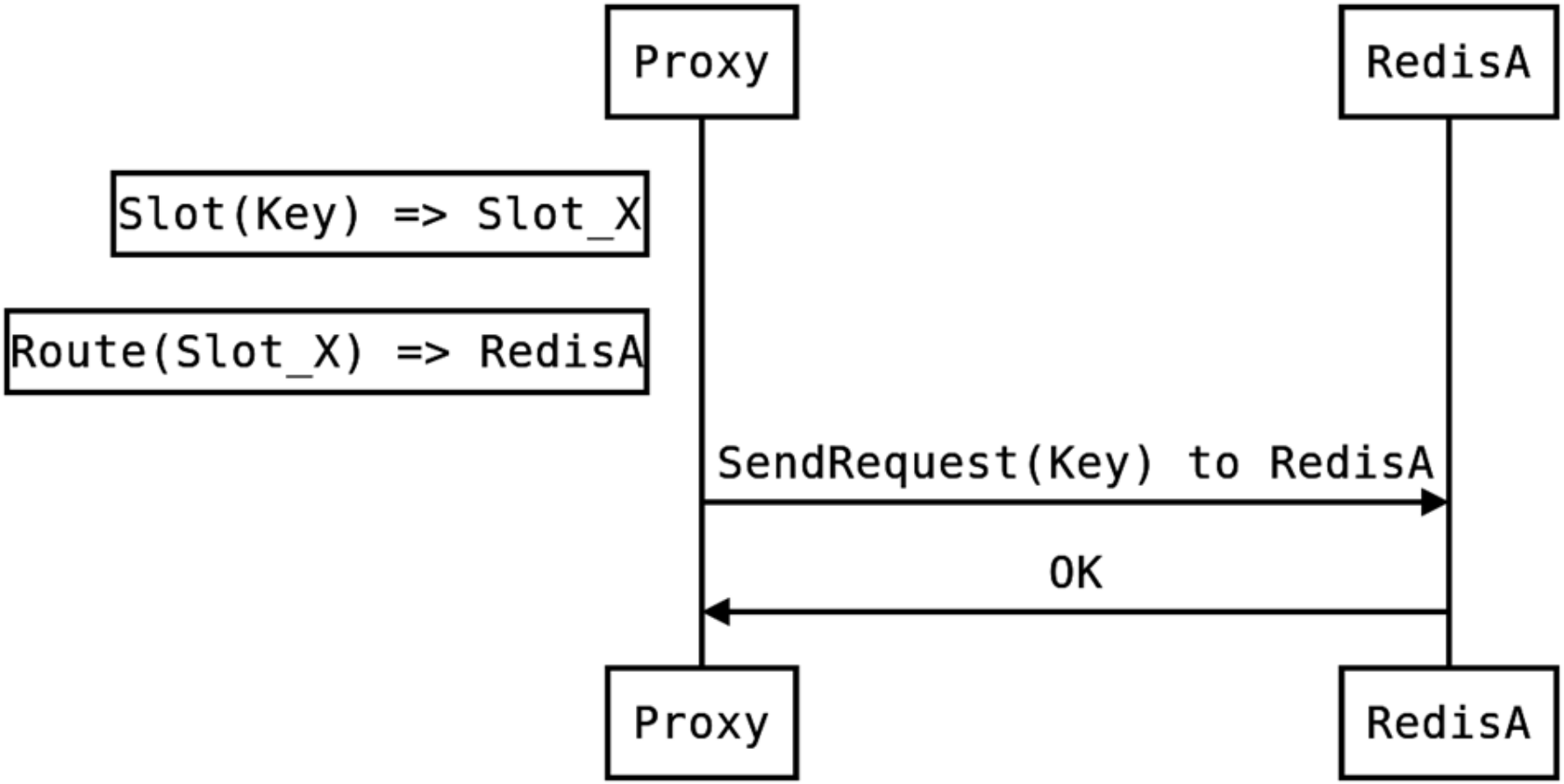
无状态 Proxy

1. 路由表统一存储在 Coordinator 中
2. 连接任意一个 proxy 发起请求的效果是一样的
3. 负载均衡变得非常简单，proxy 可以平滑的水平扩展

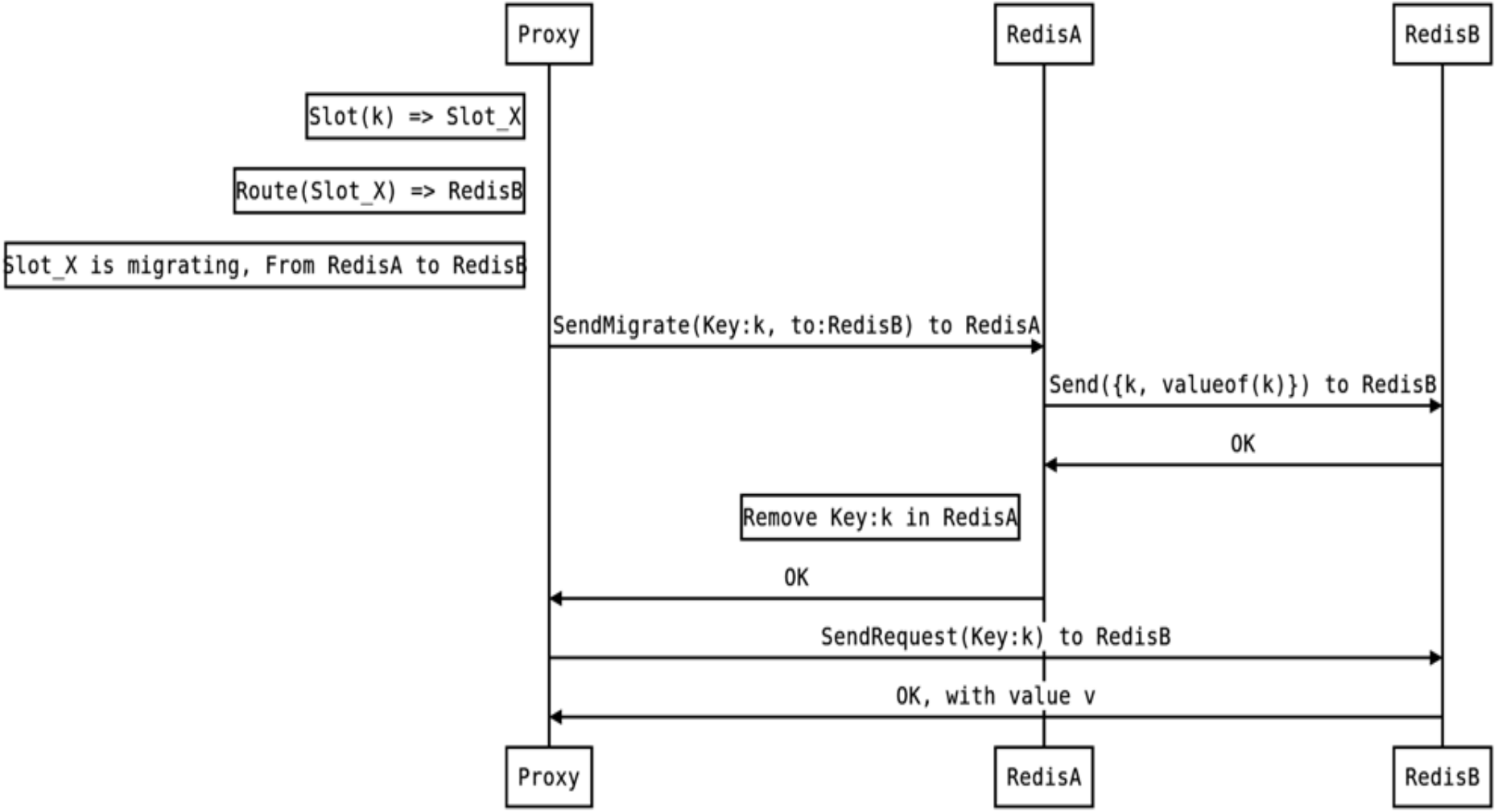
路由信息一致性保证

- 在 cluster-admin 发起集群状态变更时，所有的 proxy 必须达到信息的一致后，才能重新对外提供服务
- cluster-admin 和 proxy 之间通过二阶段提交保证一致性

Read/Write Flow (Normal)



Read/Write Flow (Migrating)



如何保证安全迁移

1. 将 slot_X 状态标记为 ‘pre_migrate’
2. 等待所有的 proxy 确认
3. 将 slot_X 状态标记为‘migrating’
4. admin 不断发送 SLOTSMGRT 命令给 source redis instance 直到 slot_X 所有的 key 迁移完成
5. 将 slot_X 状态标记为 ‘online’

Result

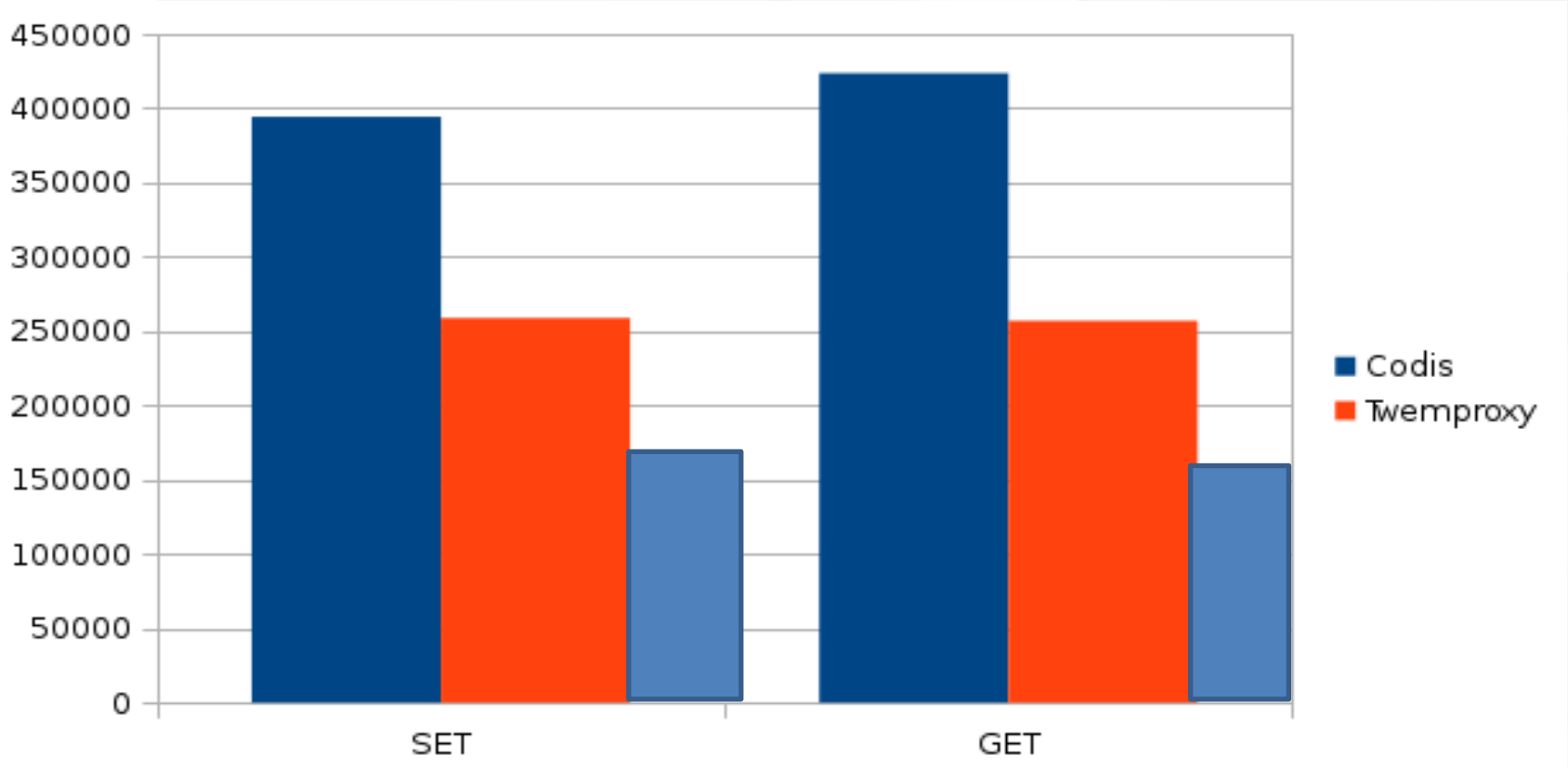
1. 平滑水平扩展
2. 可插拔的存储引擎（各个模块之间的纽带是 Redis Protocol）
3. 由于不同组件之间解耦, 都可以独立升级
   a. proxy
   b. cluster admin
   c. storage engine

RebornDB

RebornDB (Codis 2.0)

1. Support both zookeeper and etcd
2. Plugglable storage engine
3. Backend pipeline
4. Simple HA tools
5. New Dashboard
6. Sync Replication
7. **Distributed Redis Protocol Framework**

性能



Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz

redis-benchmark -p 22121 -c 500 -n 5000000 -P 100 -r 10000 -t get,set

性能

1. Pipeline 带来的性能提升
2. Golang 利用多核的能力
3. 除此之外，多节点和可以平行扩展的 proxy 让整个集群的性能也同样可以 scale

抽象 Coordinator

接口化

- 使用 zookeeper 和 etcd 的最小子集实现现有功能
  - https://github.com/ngaut/zkhelper

无状态的存储引擎

1. 存储引擎不负责具体的路由信息，只关心数据的存储，及点对点的迁移
2. Redis 协议

持久化存储

RocksDB <3 Redis Protocol

- 完全兼容 Redis 协议，包括 SYNC 相关协议
- 实现了 Codis 的 SLOT 相关指令

持久化存储

典型业务

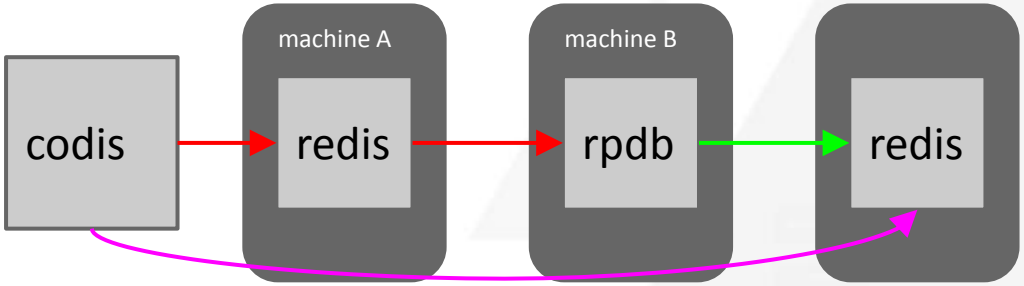
- 作为 NoSQL 使用
- 数据量大 118G 无热备（备份靠手，周期 bgsave)
- 重做数据困难
- 读写速度小 (QPS 500~2000, max=5000)

持久化存储

- as redis-server
  - 读写 QPS set=3w, get=4w 链接
  - 指令集：
    - 大部分数据相关指令 链接
      - 不包括 set 有关的多key 指令
      - 不包括 zset.{rank,range} 操作
      - 目前业务不需要
  - 支持 slaveof, sync, bgsave 等
    - bgsave 相对较慢

持久化存储

- as redis-{master, slave}



Codis-HA

- 读取集群 server groups 信息，对 master 探活，调用 RESTful API 提升 slave

Memcached
Redis

Twemproxy

Codis

RebornDB

MySQL
PostgreSQL
SQLServer

MongoDB
Cassandra
HBase

MegaStore

Spanner
?

RebornDB

http://github.com/reborndb/reborndb/reborn