TRedis在腾讯游戏的演进和实践

汪清平

腾讯游戏DBA团队

议程

- Redis的介绍
- Redis@腾讯游戏
- TRedis的介绍
- TRedis@腾讯游戏
- 经验和教训
- Q&A

Redis is an open source, BSD licensed, advanced **key-value cache** and **store**. It is often referred to as a **data structure server** since keys can contain **strings, hashes, lists, sets, sorted sets, bitmaps** and **hyperloglogs**.

— http://redis.io

Redis的特性

❖ 高性能	get/set 10w+
❖ 单线程	单线程：收包、发包、解析
❖ Key/Value存储	所有数据按“key”访问
❖ 纯内存+持久化（V2.8）	数据全内存，高性能，RDB/AOF落地
❖ 支持多种数据结构	string,hash,list,set,sorted set, geo
❖ 协议简单	各种语言的api支持
❖ 生产者/消费者消息队列	高性能，异步处理请求队列
❖ 过期时间	到期数据自动删除

Redis@腾讯游戏

- string: openid映射/用户信息
- hash: 角色信息/装备道具
- list: 消息队列/评论
- set: 属性记录/资格
- zset: 排行榜
- ttl: 活动礼包，Cache过期等

Redis@腾讯游戏

3500+实例

5T+内存

300W+请求峰值

1500亿+日请求量

Redis痛点

- 访问密度趋低
- 服务拉起缓慢
- 存在数据安全问题
- 单机存储量受限
- 运维不方便

我们需要的Redis

- 超大存储容量
- 快速启动
- 数据安全
- 易于运维
- 集群友好
- 高性能

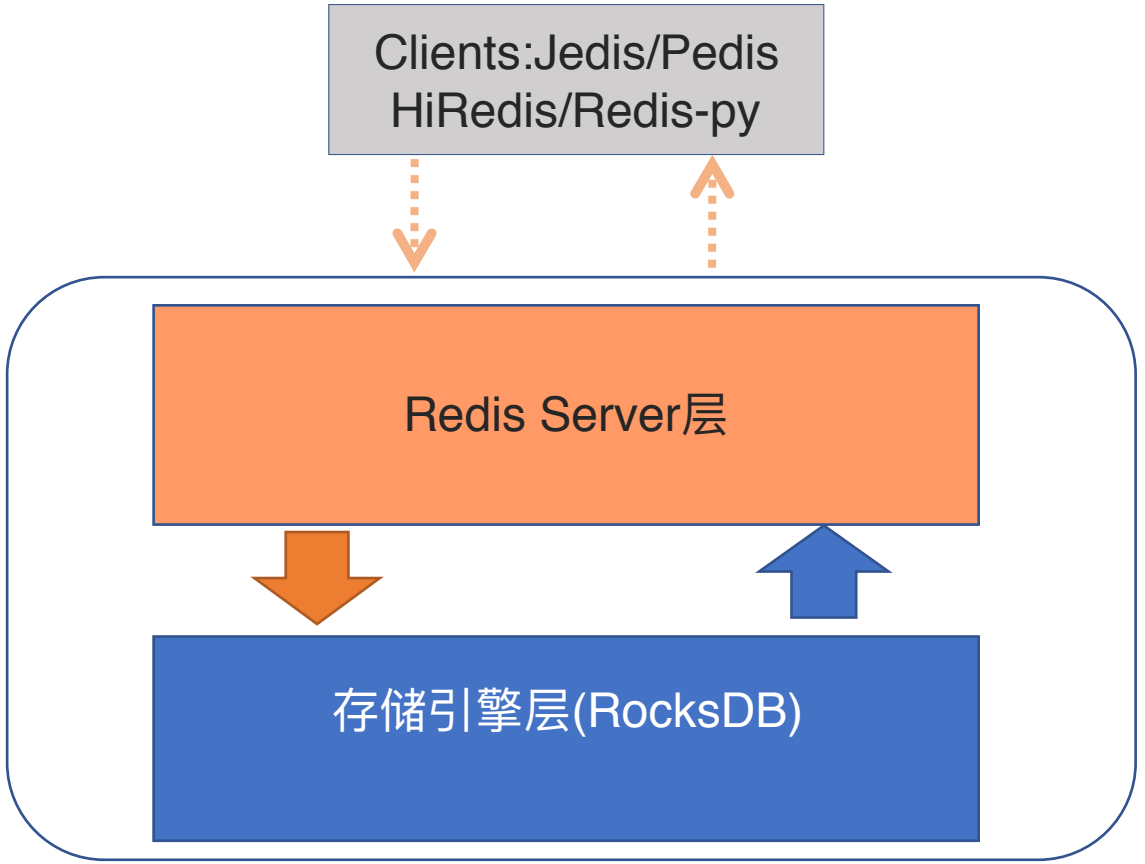
TRedis介绍

- 以RocksDB为存储引擎的高性能持久化Redis存储
- 完全兼容Redis协议，高度兼容Redis功能
- 以SSD为存储介质，实现单实例亿级key存储，容量突破内存限制
- 数据实时写入RocksDB，无数据丢失风险
- 启动速度快，无须加载全量数据到内存

TRedis介绍

- 基于binlog的主从同步，主从同步关系更加健壮
- 全量数据同步时的断点续传
- 基于物理备份的快速重建热备
- 基于binlog的定点数据构建
- 内置的实例拆分

TRedis架构


Clients:Jedis/Pedis
HiRedis/Redis-py
Redis Server层
存储引擎层(RocksDB)

TRedis设计

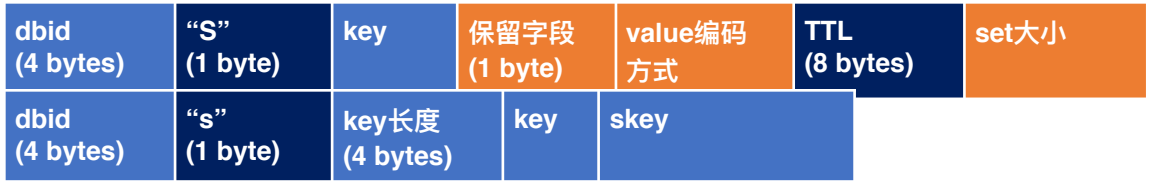
- 基于key/value构建Redis的数据结构
  - RocksDB所有key/value按照key的字典序存储
  - RocksDB提供高效的迭代器功能
  - RocksDB提供原子更新功能
- 基于binlog的可靠而高效的主从同步方案
  - RocksDB有高效的快照机制
  - RocksDB提供原子更新功能
  - RocksDB提供方便的一致性备份功能

TRedis设计 – 存储设计

- String

- Hash

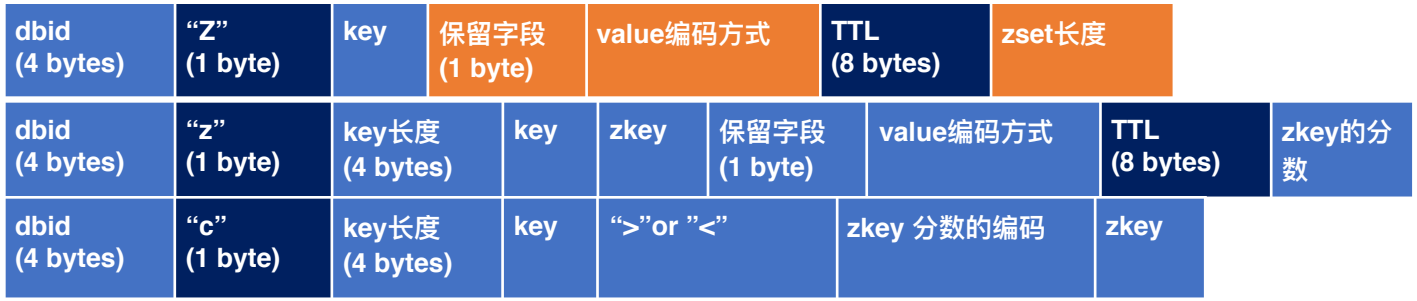
- Set



TRedis设计 – 存储设计

- List

- ZSet

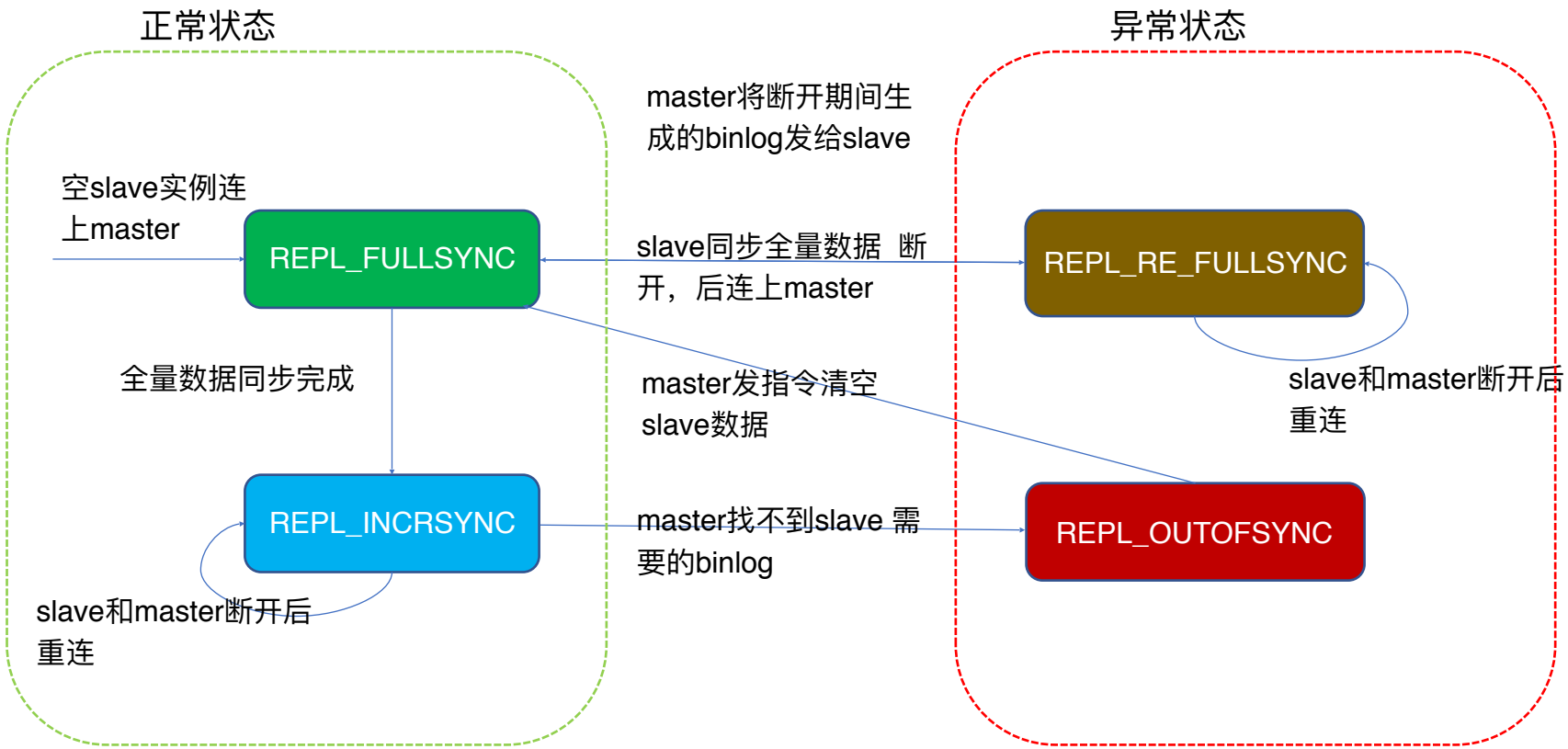


TRedis设计 - 存储设计

- ./redis-benchmark -p 9988 -n 1000000 -c 24 -r 100000000 rpush TRedis_list __rand_int__
  - RPUSH: 33705.21 requests per second
  - LPOP: 24636.00 requests per second

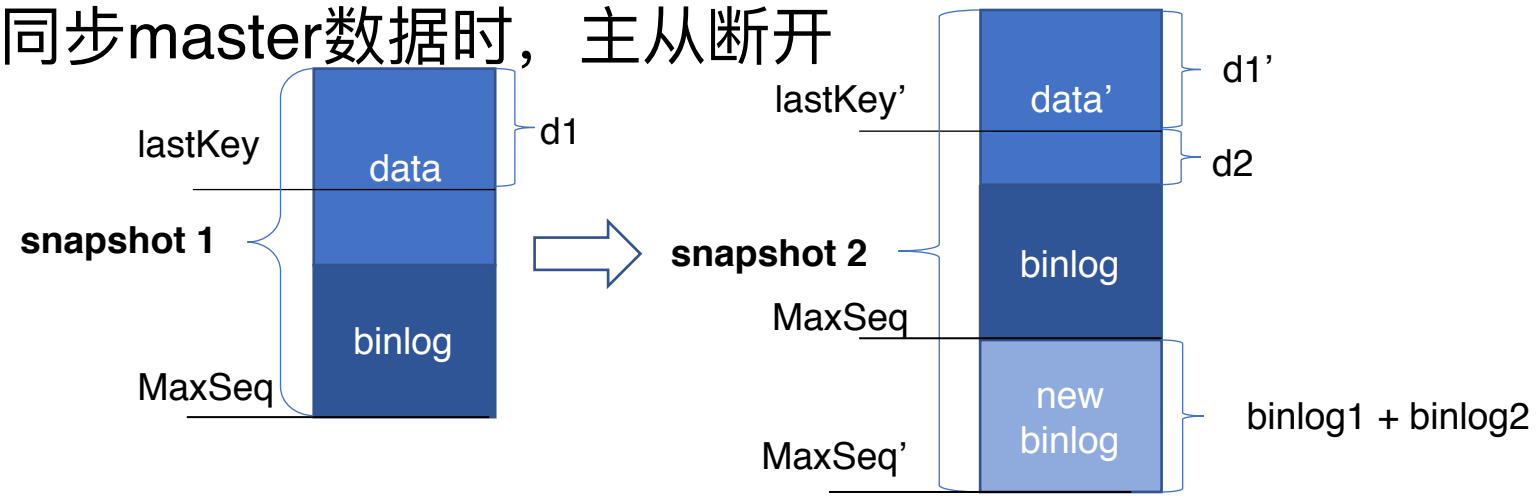
- ./redis-benchmark -p 9988 -n 1000000 -c 24 -t set,get -r 100000000
  - SET: 40278.73 requests per second
  - GET: 44120.89 requests per second

TRedis设计 – 基于binlog的同步



TRedis设计 – 基于binlog的主从同步

- 主从同步异常处理
  - 增量同步binlog时，主从断开
  - 全量同步master数据时，主从断开



snapshot 2 = snapshot 1 + binlog[MaxSeq + 1 ... MaxSeq']

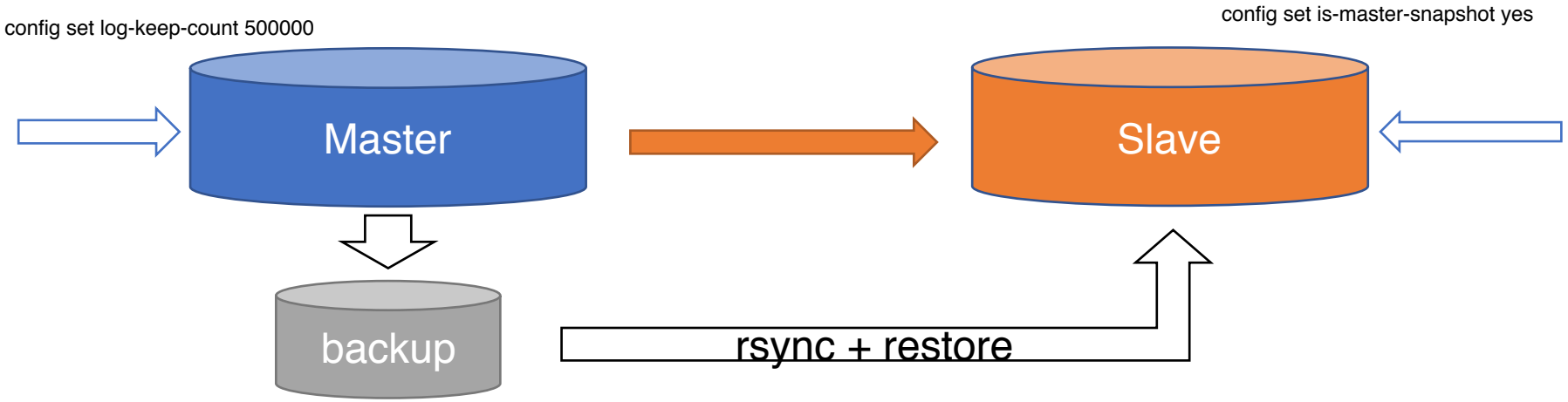
data` = d1'+ d2 = d1 + binlog1 + d2

其中，binlog1应用于存放在lastKey以前的KV，binlog2应用于存放在lastKey以后的KV

(前置条件： 一条binlog不能修改两组key/value)

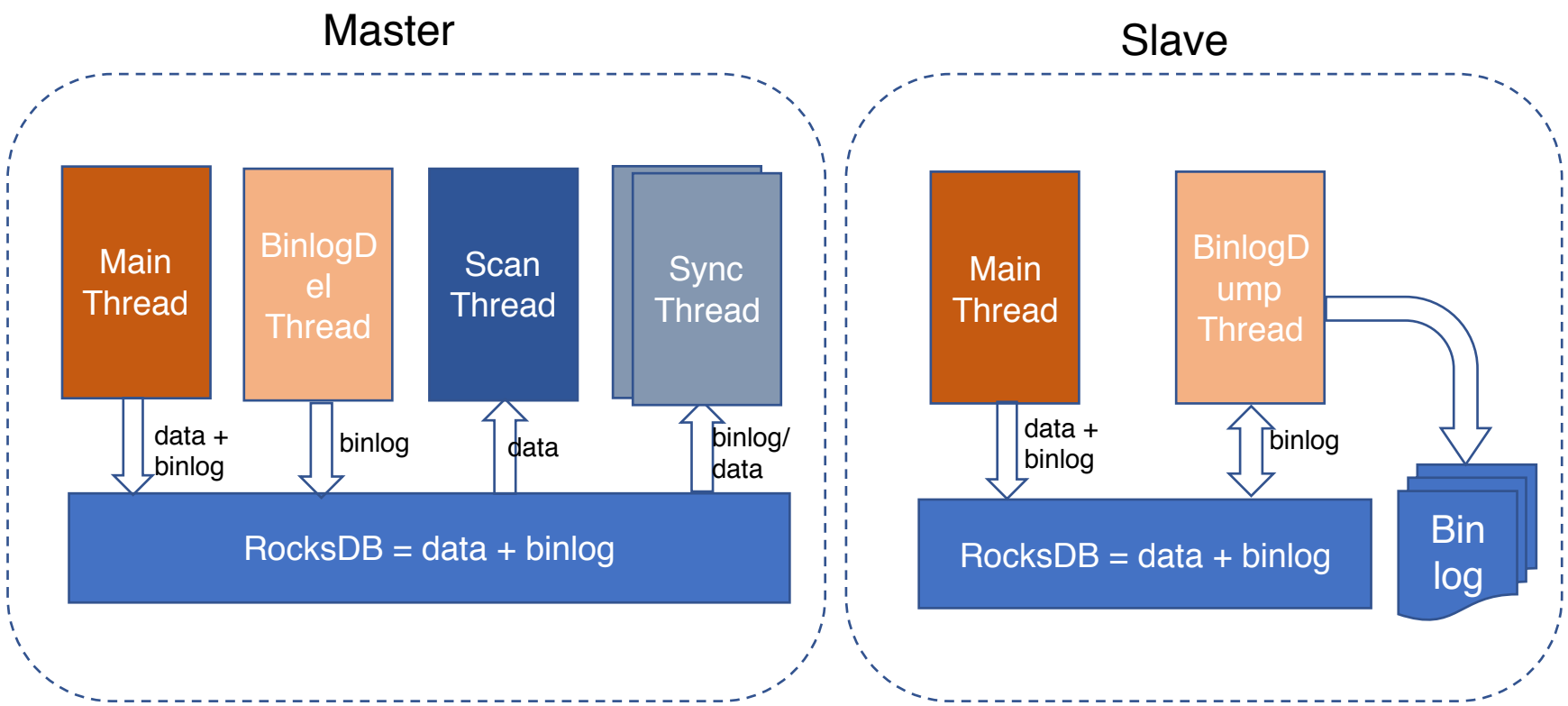
TRedis设计 – 基于binlog的主从同步

- 全量数据同步耗时太长
  - Master需要保留足够数量的binlog
  - 拉起slave后，到Slave与Master建立同步关系前，Slave的数据不能变化

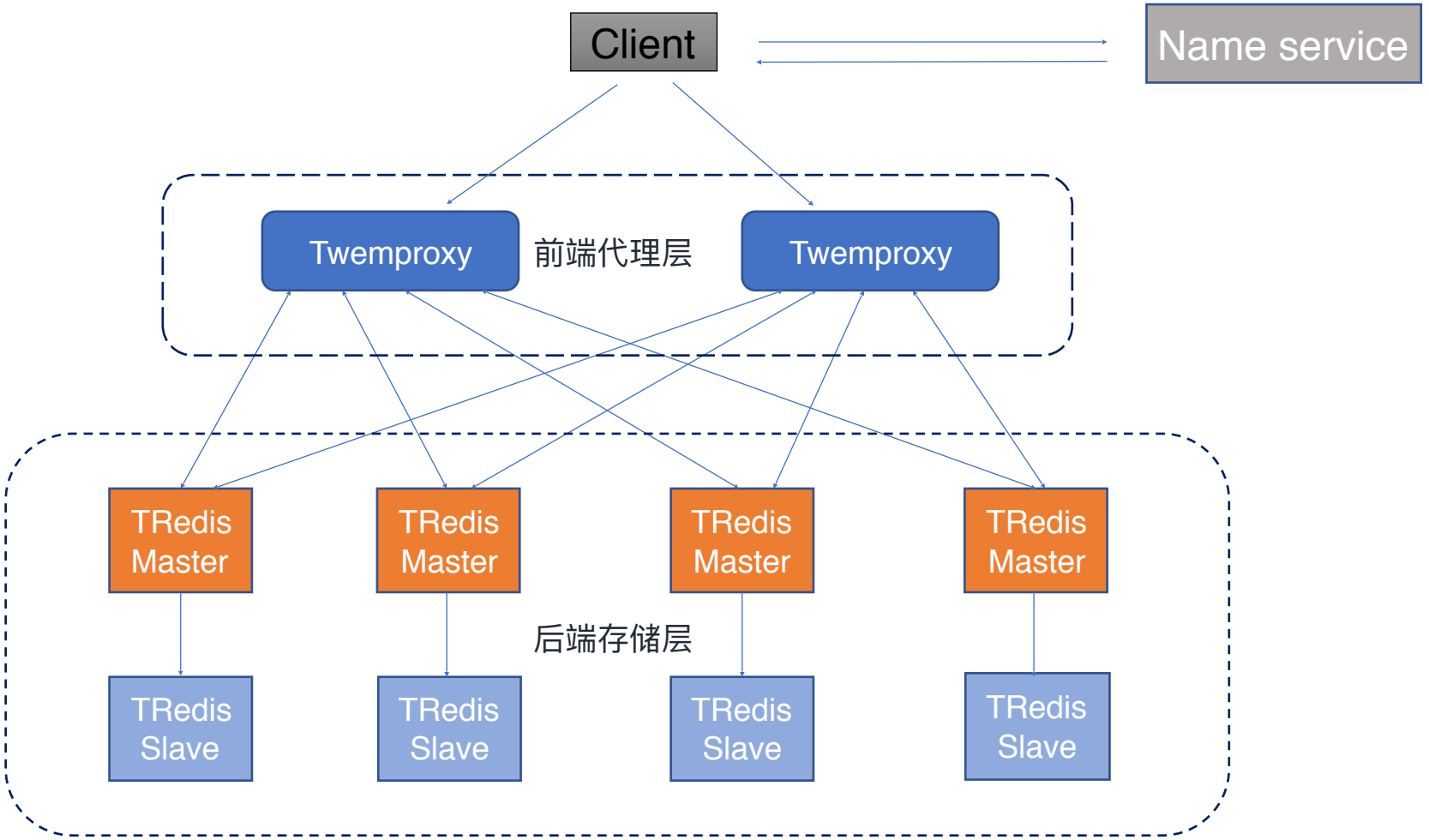


单实例(~30G)重建slave时间缩小到 **10 ~ 20分钟**

TRedis架构

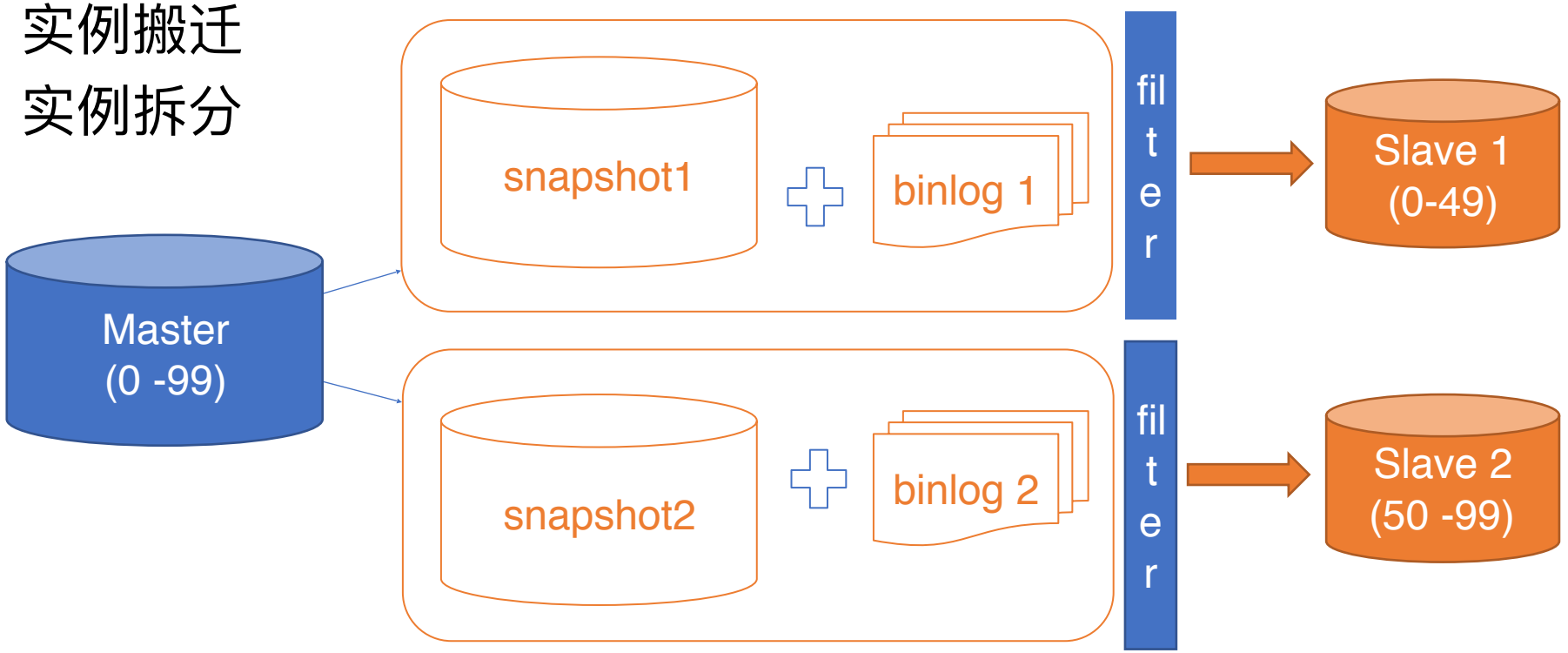

Master
Main Thread
BinlogDel Thread
Scan Thread
Sync Thread
data + binlog
binlog
data
binlog/ data
RocksDB = data + binlog
Slave
Main Thread
BinlogDump Thread
data + binlog
binlog
RocksDB = data + binlog
Bin log

集群下的TRedis


Client
Name service
Twemproxy
前端代理层
Twemproxy
TRedis Master
TRedis Master
TRedis Master
TRedis Master
后端存储层
TRedis Slave
TRedis Slave
TRedis Slave
TRedis Slave

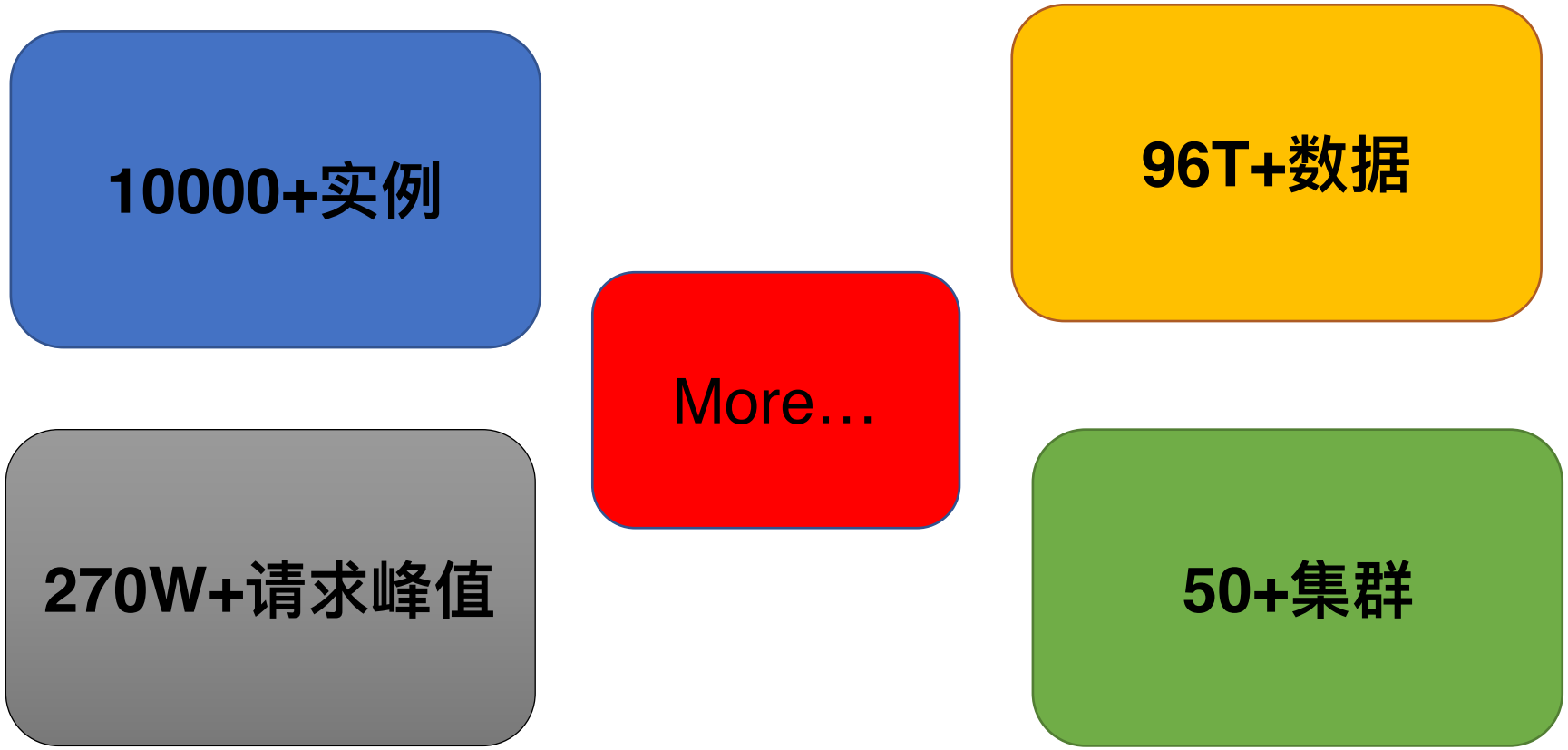
集群下的TRedis

- 集群下的挑战：快速扩容
  - 实例搬迁
  - 实例拆分



实例拆分方案

TRedis@腾讯游戏



经验和教训

- 功能的取舍
  - 选择不实现lrem和linsert
  - Set和ZSet的实现方式
- 方案的选择
  - binlog存放于RocksDB还是写到独立文件
  - 使用rsync同步全备还是逐key同步
- 造还是不造
  - 能驾驭的方案才是靠谱的方案
  - 不需要完美的方案，满足业务需求的方案就是好方案

Q&A