作者 萧田国 发布于 2015年4月6日 | 9 讨论

专栏介绍

“高效运维最佳实践”是InfoQ在2015年推出的精品专栏，由触控科技运维总监萧田国撰写，InfoQ总编辑崔康策划。

前言

诚如开篇文章所言，高效运维包括管理的专业化和技术的专业化。前两篇我们主要在说些管理相关的内容，本篇说一下技术专业化。希望读者朋友们能适应这个转换，谢谢。

互联网早在几年前就已进入Web 2.0时代，对后台支撑能力的要求，提高了几十倍甚至几百倍。在这个演化过程中，缓存系统扮演了举足轻重的角色。

运维进化到今天，已经不是重复造轮子的时代。所以，我们在架构优化和自动化运维中，可以尽可能地选用优秀的开源产品，而不是自己完全从头再来（各种技术geek除外）。

本文主要讨论Redis集群相关技术及新发展，关于Redis运维等内容，以后另开主题讨论。

本文重点推荐Codis——豌豆荚开源的Redis分布式中间件（该项目于4个月前在GitHub开源，目前star已超过2100）。其和Twemproxy相比，有诸多激动人心的新特性，并支持从Twemproxy无缝迁移至Codis。

本文主要目录如下，对Redis比较了解的朋友，可跳过前两部分，直接欣赏Codis相关内容。

1. Redis常见集群技术
   1.1 客户端分片
   1.2 代理分片
   1.3 Redis Cluster
2. Twemproxy及不足之处

3. Codis实践

3.1 体系架构

3.2 性能对比测试

3.3 使用技巧、注意事项

好吧我们正式开始。

1. Redis常见集群技术

长期以来，Redis本身仅支持单实例，内存一般最多10~20GB。这无法支撑大型线上业务系统的需求。而且也造成资源的利用率过低——毕竟现在服务器内存动辄100~200GB。

为解决单机承载能力不足的问题，各大互联网企业纷纷出手，“自助式”地实现了集群机制。在这些非官方集群解决方案中，物理上把数据“分片”（sharding）存储在多个Redis实例，一般情况下，每一“片”是一个Redis实例。

包括官方近期推出的Redis Cluster，Redis集群有三种实现机制，分别介绍如下，希望对大家选型有所帮助。

1.1 客户端分片

这种方案将分片工作放在业务程序端，程序代码根据预先设置的路由规则，直接对多个Redis实例进行分布式访问。这样的好处是，不依赖于第三方分布式中间件，实现方法和代码都自己掌控，可随时调整，不用担心踩到坑。

这实际上是一种静态分片技术。Redis实例的增减，都得手工调整分片程序。基于此分片机制的开源产品，现在仍不多见。

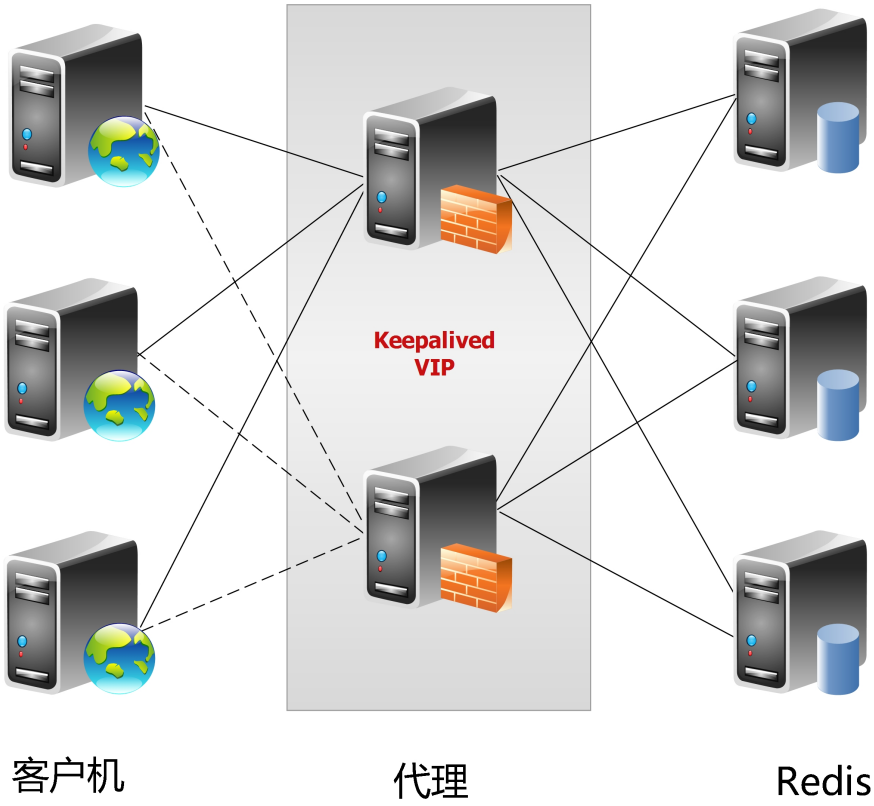
这种分片机制的性能比代理式更好（少了一个中间分发环节）。但缺点是升级麻烦，对研发人员的个人依赖性强——需要有较强的程序开发能力做后盾。如果主力程序员离职，可能新的负责人，会选择重写一遍。

所以，这种方式下，可运维性较差。出现故障，定位和解决都得研发和运维配合着解决，故障时间变长。

这种方案，难以进行标准化运维，不太适合中小公司（除非有足够的DevOPS）。

1.2 代理分片

这种方案，将分片工作交给专门的代理程序来做。代理程序接收到来自业务程序的数据请求，根据路由规则，将这些请求分发给正确的Redis实例并返回给业务程序。



这种机制下，一般会选用第三方代理程序（而不是自己研发），因为后端有多个Redis实例，所以这类程序又称为分布式中间件。

这样的好处是，业务程序不用关心后端Redis实例，运维起来也方便。虽然会因此带来些性能损耗，但对于Redis这种内存读写型应用，相对而言是能容忍的。

这是我们推荐的集群实现方案。像基于该机制的开源产品Twemproxy，便是其中代表之一，应用非常广泛。

1.3 Redis Cluster

在这种机制下，没有中心节点（和代理模式的重要不同之处）。所以，一切开心和不开心的事情，都将基于此而展开。

Redis Cluster将所有Key映射到16384个Slot中，集群中每个Redis实例负责一部分，业务程序通过集成的Redis Cluster客户端进行操作。客户端可以向任一实例发出请求，如果所需数据不在该实例中，则该实例引导客户端自动去对应实例读写数据。

Redis Cluster的成员管理（节点名称、IP、端口、状态、角色）等，都通过节点之间两两通讯，定期交换并更新。

由此可见，这是一种非常“重”的方案。已经不是Redis单实例的“简单、可依赖”了。

可能这也是延期多年之后，才近期发布的原因之一。
这令人想起一段历史。因为Memcache不支持持久化，所以有人写了一个Membase，后来改名叫Couchbase，说是支持Auto Rebalance，好几年了，至今都没多少家公司在使用。
这是个令人忧心忡忡的方案。为解决仲裁等集群管理的问题，Oracle RAC还会使用存储设备的一块空间。而Redis Cluster，是一种完全的去中心化……
本方案目前不推荐使用，从了解的情况来看，线上业务的实际应用也并不多见。

2. Twemproxy及不足之处

Twemproxy是一种代理分片机制，由Twitter开源。Twemproxy作为代理，可接受来自多个程序的访问，按照路由规则，转发给后台的各个Redis服务器，再原路返回。
这个方案顺理成章地解决了单个Redis实例承载能力的问题。当然，Twemproxy本身也是单点，需要用Keepalived做高可用方案。
我想很多人都应该感谢Twemproxy，这么些年来，应用范围最广、稳定性最高、最久经考验的分布式中间件，应该就是它了。只是，他还有诸多不方便之处。
Twemproxy最大的痛点在于，无法平滑地扩容/缩容。
这样导致运维同学非常痛苦：业务量突增，需增加Redis服务器；业务量萎缩，需要减少Redis服务器。但对Twemproxy而言，基本上都很难操作（那是一种锥心的、纠结的痛……）。
或者说，Twemproxy更加像服务器端静态sharding。有时为了规避业务量突增导致的扩容需求，甚至被迫新开一个基于Twemproxy的Redis集群。
Twemproxy另一个痛点是，运维不友好，甚至没有控制面板。
Codis刚好击中Twemproxy的这两大痛点，并且提供诸多其他令人激赏的特性。

3. Codis实践

Codis由豌豆荚于2014年11月开源，基于Go和C开发，是近期涌现的、国人开发的优秀开源软件之一。现已广泛用于豌豆荚的各种Redis业务场景（已得到豌豆荚@刘奇同学的确认，呵呵）。
从3个月的各种压力测试来看，稳定性符合高效运维的要求。性能更是改善很多，最初比Twemproxy慢20%；现在比Twemproxy快近100%（条件：多实例，一般Value长度）。

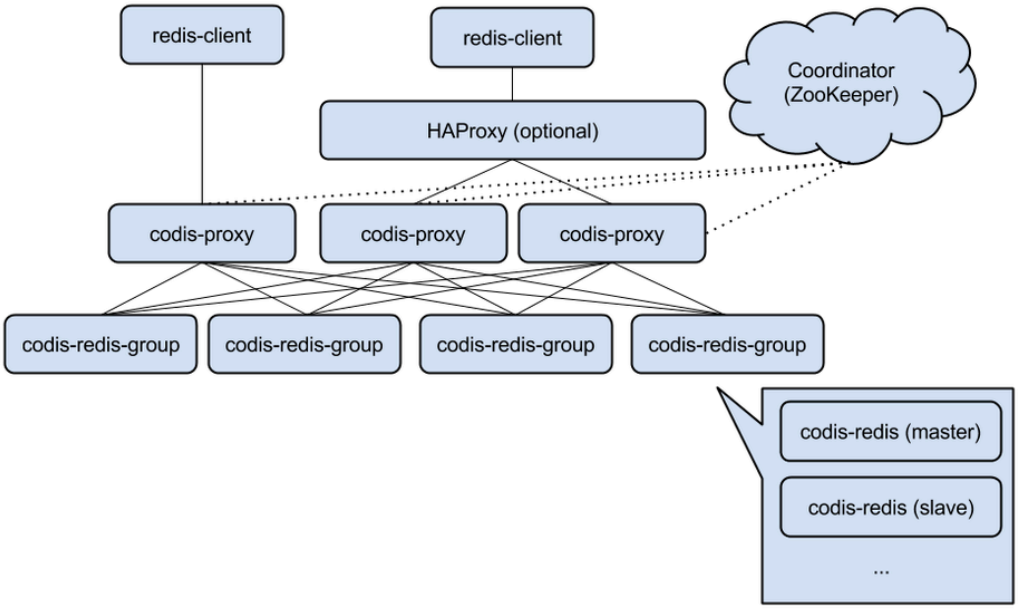
3.1 体系架构

Codis引入了Group的概念，每个Group包括1个Redis Master及至少1个Redis

Slave，这是和Twemproxy的区别之一。这样做的好处是，如果当前Master有问题，则运维人员可通过Dashboard“自助式”切换到Slave，而不需要小心翼翼地修改程序配置文件。

为支持数据热迁移（Auto Rebalance），出品方修改了Redis Server源码，并称之为Codis Server。

Codis采用预先分片（Pre-Sharding）机制，事先规定好了，分成1024个slots（也就是说，最多能支持后端1024个Codis Server），这些路由信息保存在ZooKeeper中。



ZooKeeper还维护Codis Server Group信息，并提供分布式锁等服务。

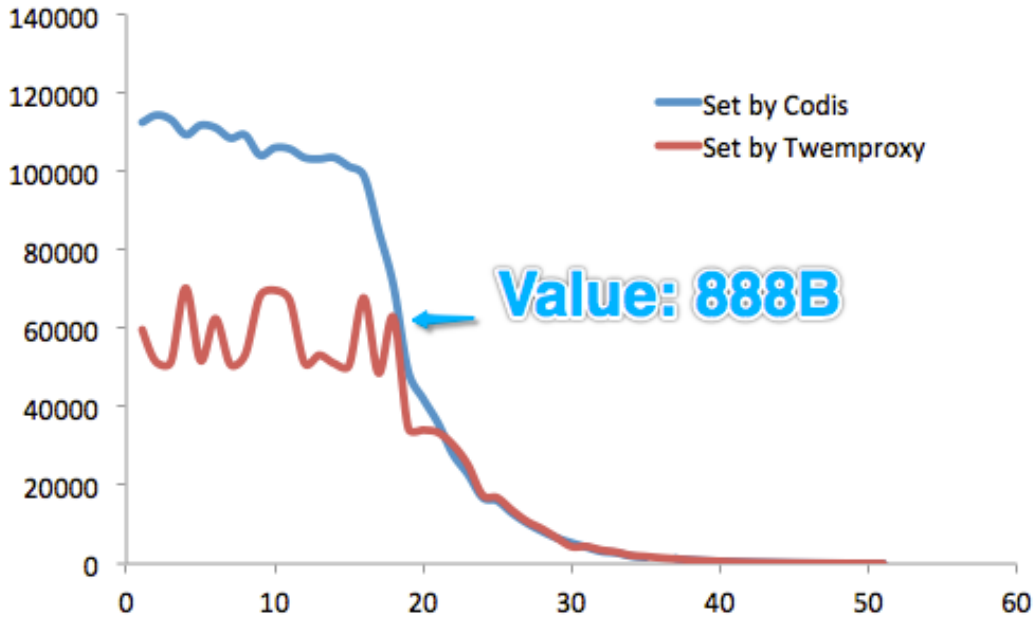
3.2 性能对比测试

Codis目前仍被精益求精地改进中。其性能，从最初的比Twemproxy慢20%（虽然这对于内存型应用而言，并不明显），到现在远远超过Twemproxy性能（一定条件下）。

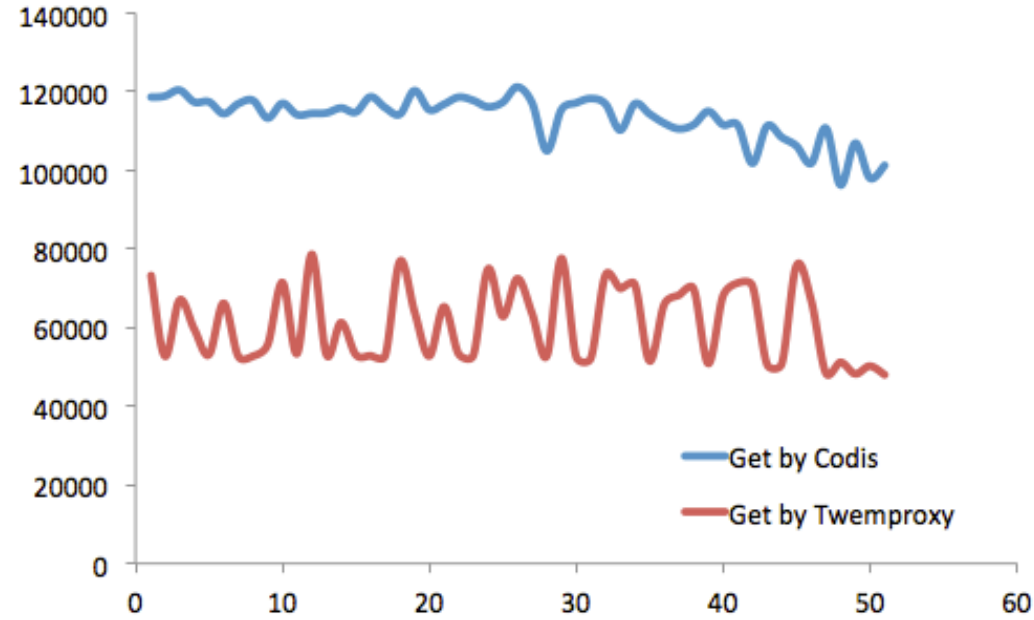
我们进行了长达3个月的测试。测试基于redis-benchmark，分别针对Codis和Twemproxy，测试Value长度从16B~10MB时的性能和稳定性，并进行多轮测试。一共有4台物理服务器参与测试，其中一台分别部署codis和twemproxy，另外三台分别部署codis server和redis server，以形成两个集群。

从测试结果来看，就Set操作而言，在Value长度<888B时，Codis性能优越优于

Twemproxy（这在一般业务的Value长度范围之内）。



就Get操作而言，Codis性能一直优于Twemproxy。



3.3 使用技巧、注意事项

Codis还有很多好玩的东东，从实际使用来看，有些地方也值得注意。

1）无缝迁移Twemproxy

出品方贴心地准备了Codis-port工具。通过它，可以实时地同步 Twemproxy 底下的 Redis 数据到你的 Codis 集群。同步完成后，只需修改一下程序配置文件，将

Twemproxy 的地址改成 Codis 的地址即可。是的，只需要做这么多。

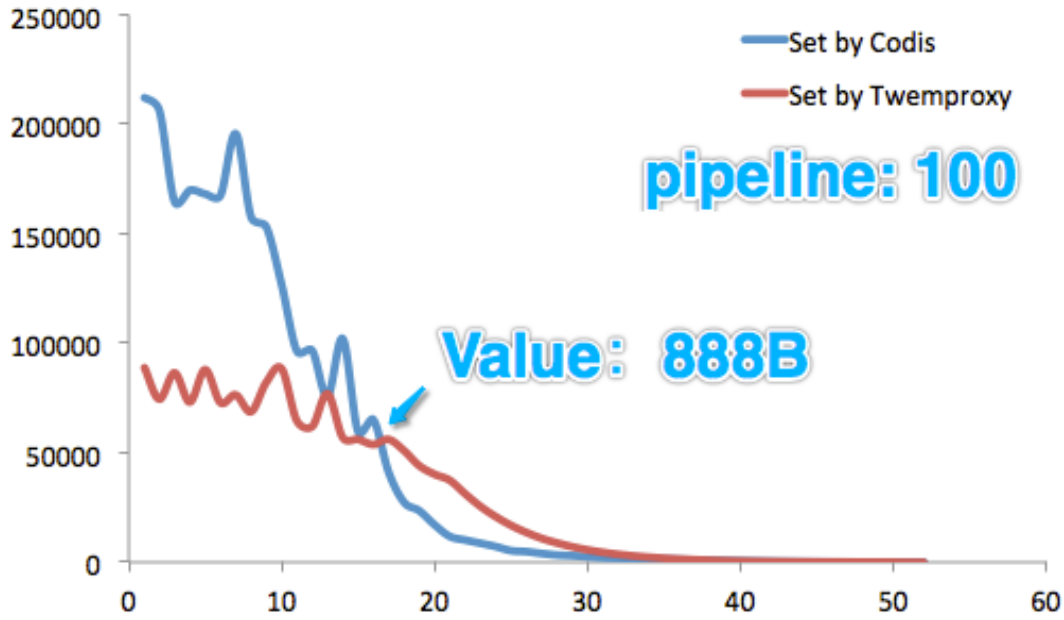
2）支持Java程序的HA

Codis提供一个Java客户端，并称之为Jodis（名字很酷，是吧？）。这样，如果单个Codis Proxy宕掉，Jodis自动发现，并自动规避之，使得业务不受影响（真的很酷！）。

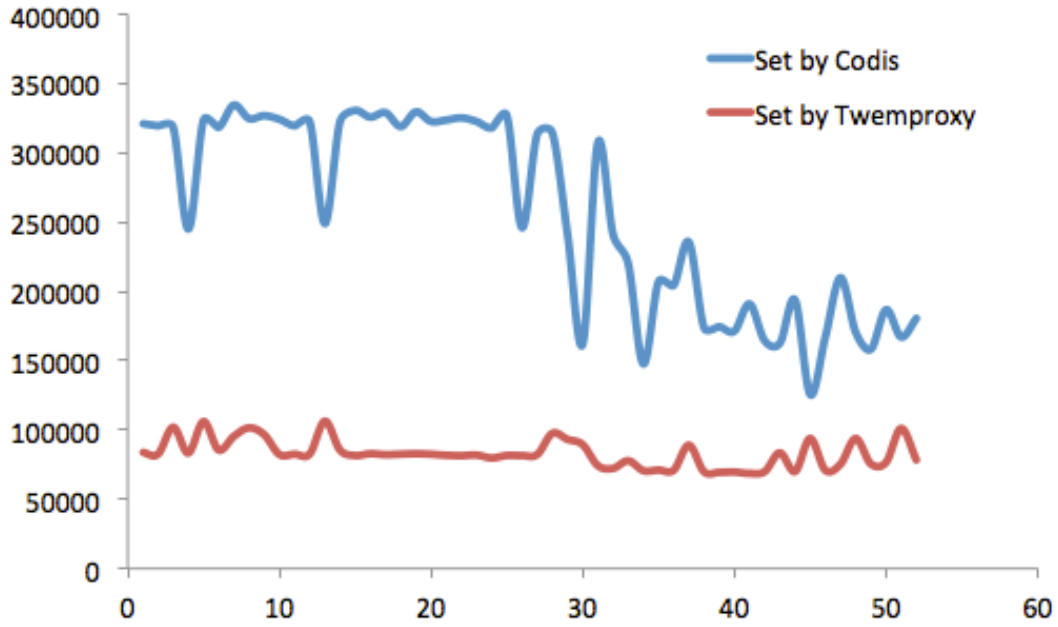
3）支持Pipeline

Pipeline使得客户端可以发出一批请求，并一次性获得这批请求的返回结果。这提升了Codis的想象空间。

从实际测试来看，在Value长度小于888B字节时，Set性能迅猛提升；



Get性能亦复如是。



4）Codis不负责主从同步

也就是说， Codis仅负责维护当前Redis Server列表，由运维人员自己去保证主从数据的一致性。

这是我最赞赏的地方之一。这样的好处是，没把Codis搞得那么重。也是我们敢于放手在线上环境中上线的原因之一。

5）对Codis的后续期待?

好吧，粗浅地说两个。希望Codis不要变得太重。另外，加pipeline参数后，Value长度如果较大，性能反而比Twemproxy要低一些，希望能有改善（我们多轮压测结果都如此）。

因篇幅有限，源码分析不在此展开。另外Codis源码、体系结构及FAQ，参见如下链接：https://github.com/wandoulabs/codis

PS：线上文档的可读性，也是相当值得称赞的地方。一句话：很走心，赞！

最后，Redis初学者请参考这个链接：http://www.gamecbg.com/bc/db/redis/13852.html，文字浅显易懂，而且比较全面。

本文得到Codis开发团队刘奇和黄东旭同学的大力协助，并得到Tim Yang老师等朋友们在内容把控方面的指导。本文共同作者为赵文华同学，他主要负责Codis及Twemproxy的对比测试。在此一并谢过。

关于作者

萧田国，男，硕士毕业于北京科技大学，ACMUG核心成员，目前为触控科技运维

负责人。拥有十多年运维及团队管理经验。先后就职于联想集团（Oracle数据库主管）、搜狐畅游（数据库主管）、智明星通及世纪互联等。从1999年开始，折腾各种数据库如Oracle/MySQL/MS SQL Server/NoSQL等，兼任数据库培训讲师若干年。

曾经的云计算行业从业者，现在喜欢琢磨云计算及评测、云端数据库，及新技术在运维中的应用。主张管理学科和运维体系的融合、人性化运维管理，打造高效、专业运维团队。

近来有时参加一些大小技术会议，做做演讲嘉宾或主持人（有空找我来玩呀：）

我的个人微信号：xiaotianguo。如需更多

另外，我也有微信公众号叻，微信搜索“开心南瓜by萧田国”或扫描如下二维码，和我进行微信互动。公众号里将有我原创的各类技术和非技术文章，及我所喜欢的文章/帖子。一起来吧~