计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(10):91−98 [doi: 10.15888/j.cnki.csa.006576]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

一种高效的 Redis Cluster 的分布式缓存系统①

李　翀[1], 刘利娜[1,2], 刘学敏[1], 张士波[1]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学, 北京 100049)
通讯作者: 刘利娜, E-mail: liulina@cnic.cn

摘　要: 为了满足大型互联网应用对高并发访问、快速响应、动态扩展、易维护性等需求, 本文基于 Redis 4.0 设计并实现了一种 Redis Cluster 分布式缓存系统, 集成了可视化开源工具 CacheCloud 对该系统进行实时监控和高效管理, 基于官方 Redis-Bechmark 进行了 QPS 性能测试, 并与 Codis 分布式缓存系统进行了对比. 实验结果表明 Redis Cluster 各功能高效运作, 性能优越, 在并发访问数 10 000 以上时响应时间明显优于 Codis.

关键词: 分布式缓存; 分片; Redis Cluster; CacheCloud; 每秒查询率 (QPS)

引用格式: 李翀,刘利娜,刘学敏,张士波.一种高效的 Redis Cluster 的分布式缓存系统.计算机系统应用,2018,27(10):91–98. http://www.c-s-a.org.cn/1003-3254/6576.html

High Efficient Distributed Cache System Based on Redis Cluster

LI Chong[1], LIU Li-Na[1,2], LIU Xue-Min[1], ZHANG Shi-Bo[1]

[1](Computer Network Information Center, University of Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: In order to meet the growing demands for high concurrency, quick response, dynamical scaling, and maintainability in larger-scale internet applications, in this research work, a distributed cluster cache system based on Redis 4.0 has been implemented, which the data can be distributed and scaled into different nodes so the system's linear scalability, load balancing, concurrency, data throughput, and responsiveness can be optimized. The open source tool called CacheCloud is integrated into it so the cluster can be efficiently managed and monitored in real time. The results show that the system reaches high performance and response time of using Query Per Second (QPS) on Redis Cluster is much faster than that on Codis after 10 000 concurrent accesses.

Key words: distributed cache system; sharding; Redis Cluster; CacheCloud; Query Per Second (QPS)

随着 Web2.0 和大数据时代到来, 单机缓存系统已无法承受当下大型互联网应用高并发、大数据、快速响应的需求. 分布式缓存[1]是学术界和工业界公认的解决方案, 通过将数据和请求分布到不同节点, 实现水平扩展和负载均衡, 进而提高并发数、数据吞吐量和快速响应能力. 目前分布式缓存系统的主流实现方式主要有两种: 基于 Memcached[2]和基于 Redis[3]的分布式缓存系统. Memcached 是一种开源高性能分布式内存对象缓存系统, 通过减少数据库负载来实现对动态 Web 应用加速. 但由于其支持数据类型单一、客户端实现分片方式导致集群扩容困难、一致性维护成本、不支持数据持久化和内存管理方式导致内存浪费等问题. Redis 是一种开源基于内存键值对存储数据库, 具有分布式缓存、高并发快速访问、丰富的数据类型、

① 基金项目: 中国科学院信息化专项 (Y647021224)
Foundation item: Informatization Special Project of Chinese Academy of Sciences (Y647021224)
收稿时间: 2018-03-09; 修改时间: 2018-03-28; 采用时间: 2018-04-02; csa 在线出版时间: 2018-09-28

数据持久化及备份机制、消息队列机制、高扩展性和可维护性等优点, 使得基于 Redis 构建分布式缓存系统开始成为热点, 其分布式主要通过分片实现, 具体方式主要有三种: 基于客户端、基于 Proxy 和基于 Redis Cluster[4,5]的分片方式.

基于客户端的分片方式, 因其水平扩展困难、业务变更导致不可控等原因, 实际应用较少. 而基于 Proxy 的分片方式在开发简单, 运维便捷, 对应用几乎透明, 相对比较成熟, 例如 Twemproxy[6]和 Codis[7]. 但是随着系统规模增大, 逐渐暴露出代理影响性能、组件较多复杂, 对资源消耗较大、扩展复杂、代码升级困难、硬件环境要求高等缺点.

Redis Cluster 成为研究热点. Redis3.0 以后开始支持 Redis Cluster, 目前最新稳定版本 Redis4.0 于 2017 年 7 月发布, 因刚发布不久, 可借鉴解决案例不多. 同时, Redis4.0 Cluster 官方给出的 Redis Cluster 部署方案比较单一, 集群数量很大时部署不便, 也没有较好的可视化监控和管理工具, 当业务扩大, 集群节点成百上千的时候, 命令行运维工具将会十分困难. 基于以上原因, 本文基于 Redis4.0 研究并实现了一种 Redis Cluster 分布式缓存系统[8,9], 集成了 CacheCloud[10]开源可视化工具进行实时监控和管理, 并进行了功能和性能验证. 实验表明, 该系统各功能高效运转, 管理便捷, 可扩展性强, 体现了高可用、高性能和高扩展的特点.

1 Redis Cluster 技术研究

Redis Cluster 是基于 Redis 的一个分布式实现, 由 Redis 官网推出; 它引入了哈希槽的概念; 支持动态添加或删除节点, 可线性扩展至 1000 个节点; 多个 Redis 节点间数据共享, 部分节点不可达时, 集群仍然可用; 数据通过异步复制, 不保证数据的强一致性; 具备自动 Failover 的能力; 客户端直接连接 Redis Server, 免去了 Proxy 的性能损耗.

1.1 工作原理

Redis Cluster 是一种去中心化结构, 如图 1 所示, 所有节点之间互相连接, 通过 Gossip 协议来发布广播消息, 每间隔时间内互发 PING/PONG 心跳包来检测其他节点状态, 来保持信息同步. 客户端直接连接任意 Redis Server, 并由 Redis Cluster 路由转发客户端请求到真正请求数据的节点.

Redis Cluster 中每个 Redis 节点需要打开两个 TCP 连接, 一个是为客户端提供服务的常规端口, 另一个是节点间互相通信所需要的端口. 这两种端口之间偏移量是 10 000. 对于分片, Redis Cluster 并没有使用一致性哈希来实现, 而是引入了哈希槽的概念. Redis Cluster 共有 16 384 个哈希槽, 每个 Master 节点只负责一部分哈希槽, 每个 Key 通过公式 (1) 计算出属于哪个节点.

$$hash\_slot = crc16(key)\%16384 \quad (1)$$

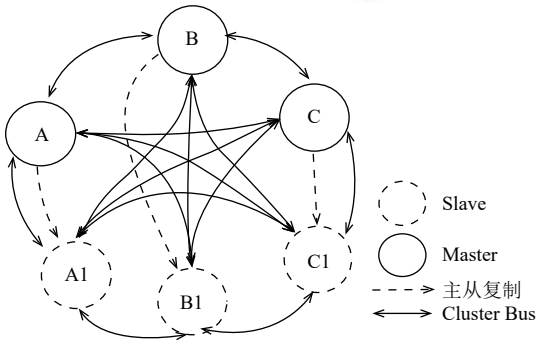

图 1 Redis Cluster 拓扑图

1.2 设计目标

CAP 即 Consistency(一致性)、Availability(可用性)、Partition tolerance(分区容错性)[11,12], 在分布式系统中不可兼得. Redis Cluster 的设计目标主要是高性能、高可用和高扩展, 但牺牲了部分的数据一致性.

(1) 一致性

是指对所有节点数据访问一致性. Redis Cluster 无法保证强一致性. 由于集群使用异步复制, 在某些情况下, Redis Cluster 可能会丢失系统向客户确认的写入. 性能和一致性之间需要权衡, Redis 集群在绝对需要时支持同步写入的时候, 可以通过 WAIT 命令实现, 但是这使得丢失写入的可能性大大降低.

(2) 可用性

可用性是对集群整体是否可用的衡量, 即当集群中一部分节点故障后, 集群整体是否还能响应客户端的读写请求. Redis 集群在分区的少数节点那边不可用. 一个由 N 个主节点组成的集群, 每个主节点都只有一个从节点. 当有一个节点被分割出去后, 集群的多数节点这边仍然是可访问的. 当有两个节点被分割出去后集群仍可用的概率是 $1–(1/(N×2–1))$. 如一个拥有 6 个节点的集群, 每个节点有一个从节点, 那么在两个节点从多数节点这边分割出去后集群不再可用的概率是 $1/(6×2–1) = 0.090909$, 即有大约 9% 的概率.

(3) 高性能和线性扩展

当操作某个 Key 时, Redis Cluster 节点不会像代理一样直接找到负责这个 Key 的节点并执行命令, 而是将客户端重定向到负责这个 Key 的节点. 通过对 Key 的操作, 客户端会记录路由信息, 最终客户端获得每个节点负责的 Key 最新信息, 所以一般情况下, 对于给定的操作, 客户端会直接连接正确的节点并执行命令. Redis Cluster 不支持同时操作多个键值, 避免了数据在节点间来回移动. 但普通操作是可以被处理得跟在单一 Redis 上一样的, 这意味着在一个拥有 N 个主节点的 Redis 集群中, 由于 Redis 的设计是支持线性扩展的, 所以同样的操作在集群上的表现会跟在单一 Redis 上的表现乘以 N 一样.

节点定时向其他节点发送 ping 命令, 它会随机选择存储的其他集群节点的其中三个进行信息“广播”, 广播的信息包含一项是节点是否被标记为 PFAIL/FAIL. 如图 2 所示, 超过一半的 Master 节点也就是 A 和 B 节点同时判定 C 节点失效, 那么 C 节点将会被标为 FAIL, 这个节点已失效的信息会被广播至整个集群, 所有集群中的节点都会将失效的节点标志为 FAIL, 即失效. 又如图中 B 节点收到 A 节点判定失效, 此时被标为 PFAIL 状态, 等待其主他节点共同判定后才能决定是否失效.

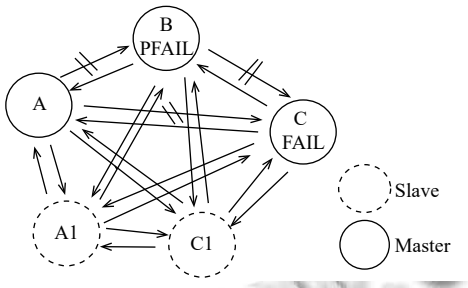

图 2 Redis Cluster 容错图

如果集群任意 Master 挂掉, 且当前 Master 没有 Slave, 该 Master 对应哈希槽无法使用, 集群不可用. 如果有 Slave 节点, 将会进行 Slave 选举, 最终选择 Slave 成为主节点. 如果集群超过半数以上 Master 挂掉, 无论是否有 Slave 集群进入 fail 状态.

2 系统设计与实现

2.1 集群架构设计

Redis Cluster 搭建于 4 台虚拟机上, 虚拟机具体信息见 2.2 运行环境中的表 2. 由于 Redis 是单线程模型, 一个 Redis 服务进程只会使用一个内核, 为了减少线程切换的开销, 提升 Redis 的吞吐量和优化 Redis 性能, 同时考虑到虚拟机内核数和内存大小, 因此集群设计为 12 个主节点, 每个节点对应 Maxmemory 为 512 MB, 每个主节点对应两个从节点, 集群共 36 个节点, 任意节点之间通过 Cluster Bus, 即集群总线, 互相连接. 集群主从节点详细信息见表 1, 图 3 是集群架构缩略图, 其中数字, 如: 8000, 为虚拟机端口.

表 1 Redis Cluster 节点信息

主从节点	IP 地址	端口号
Master(4)	172.17.0.212	8000/8001/8002/8003
Slave(4)	172.17.2.22	8002/8003/8004/8005
Slave(4)	172.17.2.23	8002/8003/8004/8005
Master(3)	172.17.0.213	8000/8001/8002
Slave(3)	172.17.2.22/23	8006/8007/8008
Master(1)	172.17.0.213	8003
Slave(2)	172.17.0.212/213	8008
Master(2)	172.17.2.22	8000/8001
Slave(4)	172.17.0.212/213	8004/8005
Master(2)	172.17.2.23	8000/8001
Slave(4)	172.17.0.212/213	8006/8007

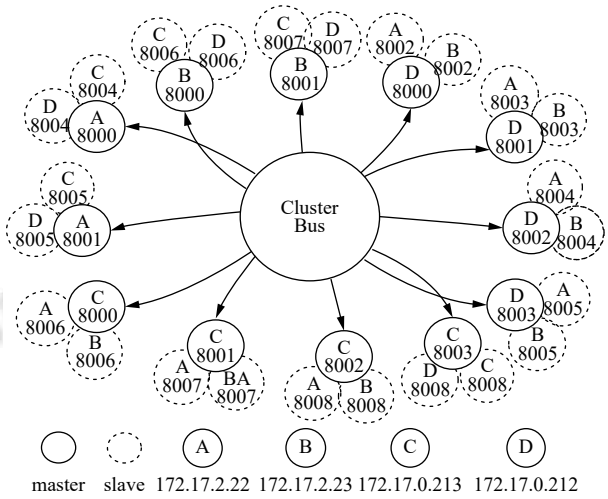

图 3 Redis Cluster 架构缩略图

搭建 Redis Cluster 是基于 Redis 最新版本 4.0, 具体版本为 Redis 4.0.1, 由于 Redis Cluster 搭建依赖 Ruby 环境, 所以使用了 Ruby2.5.0 版本.

2.2 运行环境

系统运行环境通过 4 台服务器模拟, 虚拟机信息如表 2 所示.

2.3 配置信息

(1) Linux 系统性能配置

1) sysctlvm.overcommit_memory=1, 该设置用来防止内存申请不到发生卡死, 造成 fork 失败的情况.

表 2 服务器集群环境

服务器名称	IP 地址	操作系统	硬件环境
服务器 1	172.17.0.212	CentOS release7.4 x86_64	4CPU、8 GB 内存、150 GB 硬盘
服务器 2	172.17.0.213	CentOS release7.4 x86_64	4CPU、8 GB 内存、150 GB 硬盘
服务器 3	172.17.2.22	CentOS release6.8 x86_64	2CPU、8 GB 内存、100 GB 硬盘
服务器 4	172.17.2.23	CentOS release6.8 x86_64	2CPU、8 GB 内存、100 GB 硬盘

2) sysctlvm.swappiness=20, 当 swappiness 为 20 时表示内存在使用到 80% 的时候, 就开始出现有交换分区的使用. Redis 运行时根据此数据来选择是否将内存中数据 swap 到硬盘.

3) Transparent Huge Pages (THP)[13] Linux kernel 在 2.6.38 内核增加了 THP 特性, 支持大内存页 (2 MB) 分配, 默认开启. 当开启时可以降低 fork 子进程的速度, 但 fork 操作之后, 每个内存页从原来 4 KB 变为 2 MB, 会大幅增加重写期间父进程内存消耗. 操作命令如下:

echo never >/sys/kernel/mm/

redhat_transparent_hugepage/enabled

4) sysctl -w net.ipv4.tcp_timestamps=1, 表示开启对于 TCP 时间戳的支持, 若该项设置为 0, 则下面一项 net.ipv4.tcp_tw_recycle=1 设置不起作用.

5) sysctl -w net.ipv4.tcp_tw_recycle=1, 表示开启 TCP 连接中 TIME-WAIT sockets 的快速回收. 对于 Redis 客户端连接数比较多时, 释放连接数.

6) sysctlnet.core.somaxconn=65535, 结合 Redis 配置参数 tcp-backlog, 当系统并发量大并且客户端速度缓慢的时候, 设置这二个参数. 主要为了提高 Redis 的并发数.

(2) CPU 绑定

Redis 是单线程模型, 一个 Redis 服务进程只会使用一个内核, 为了减少线程切换的开销, 提升 Redis 的吞吐量和优化 Redis 性能, 因此, 选择将 Redis 进程绑定到固定 CPU 上. 以下是具体绑定步骤:

1) 从获取 Redis 节点 PID: ps -ef | grep redis;

2) 查看 CPU 信息: cat/proc/cpuinfo;

3) 绑定 CPU: taskset-cp x PID

(3) Redis 配置信息

以 172.17.0.213:8000 节点为例, 其他节点配置除了端口号、bind 绑定的 IP 和 PID、RDB、AOF、log 等文件名称不一样外, 其他信息一致.

######基本参数#########

daemonize yes#以守护进程方式运行

pidfile/var/run/redis-8000.pid#节点唯一标识 PID

cluster-enabled yes #开启 Redis Cluster 模式

cluster-config-file nodes-8000.conf

save 900 1 #900 秒 (15 分钟) 内有 1 个更改

save 300 10 #300 秒 (5 分钟) 内有 10 个更改

save 60 10000#60 秒内有 10000 个更改

dbfilename8000-dump.rdb#RDB 持久化存储文件

appendonly yes#打开 AOF 持久化

appendfilename "8000-appendonly.aof"#AOF 存储文件

################优化配置参数#############

#考虑安全性问题

port 8000#节点监听端口, 安全起见, 不使用默认 6379

bind 172.17.0.213 #绑定内网 IP, 防止外网访问, 防攻击

protected-mode yes #当开启后, 禁止公网访问 Redis

tcp-backlog 65535#确定 TCP 连接中已完成队列长度

tcp-keepalive 300#周期性检测客户端是否还处于健康状###态, 避免服务器一直阻塞

maxclients 1000000 #最大客户端连接数, 提高并发量

cluster-node-timeout 15000

##集群节点的不可用时间 (超时时间), 超过这个时间就会##被认为下线, 单位是毫秒

cluster-slave-validity-factor 1

#集群主从切换的控制因素之一, 若大于 0, 主从间的最大#断线时间通过 node timeout*cluster-node-timeout 来计算

cluster-migration-barrier 1

##与同一主节点连接的从节点最少个数

cluster-require-full-coverage no

#默认值为 yes, 全部 hash slots 都正常集群才可用, 若为##no, 那么可以允许部分哈希槽下线的情况下继续使用

maxmemory512M#节点最大使用内存

maxmemory-policy volatile-lru#超出最大内存后替换策略

requirepass, ./z#2kknsl;aksKm-0q8%^&#提升安全性

##

2.4 集群管理

CacheCloud 是用来监控和管理 Redis Cluster 的一个开源工具. 在 172.17.0.212 虚拟机上搭建 CacheCloud, 并将 Redis Cluster36 个节点作为一个应用导入至 CacheCloud, 开始收集 Redis Cluster 节点数据, 进行实时监控 Redis Cluster 整个集群状态. 图 4、图 5 和图 6 展示了 Redis Cluster 接入 CacheCloud 之后的部分功能图, 可以实时在图形化界面查看 Redis Cluster 拓扑图、节点流量监控、客户端连接统计等信息, 还可以进行相关管理.

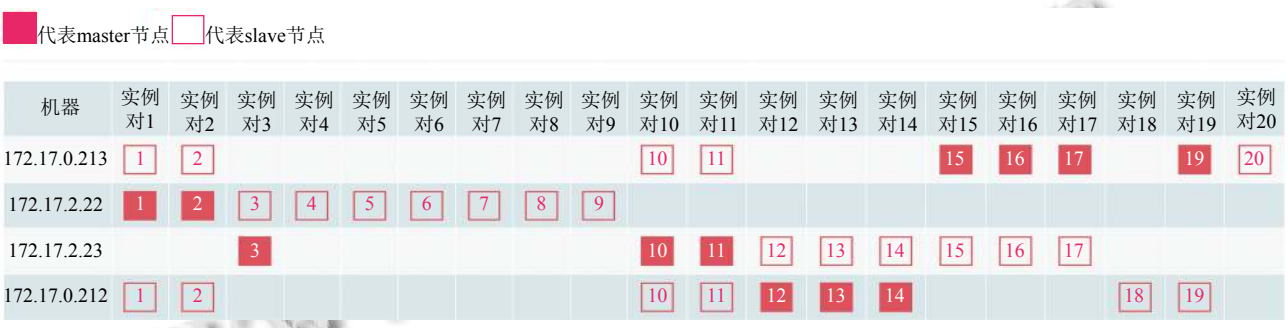

图 4 Redis Cluster – CacheCloud 应用节点拓扑图

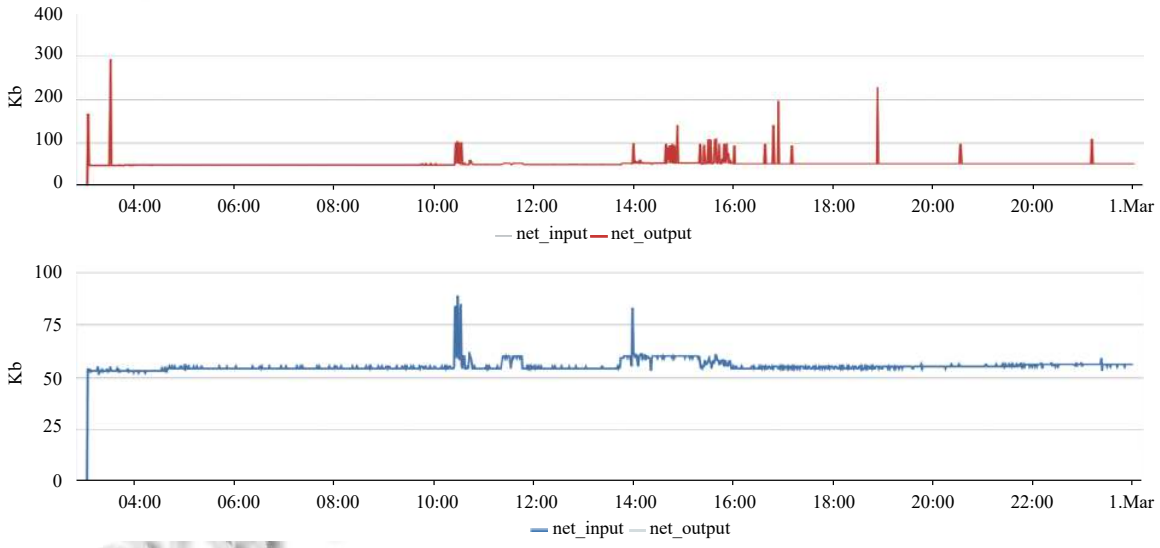

图 5 Redis Cluster – CacheCloud 应用网络流量图

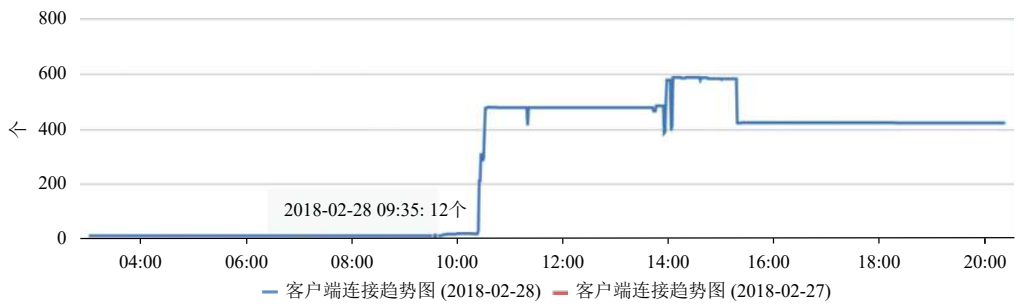

图 6 Redis Cluster – CacheCloud 客户端连接统计图

3 测试与分析

本部分工作分别以 Redis Cluster 功能验证和性能测试展开. 功能测试主要针对系统可用性、故障自诊断和智能恢复、动态扩展、配置管理等管理功能进行验证, 验证工具采用管理工具 CacheCloud. 性能测试分两个部分: 基于 Redis-Bechmark 对 Redis Cluster 本身进行不同命令请求数的 QPS 测试; 和业界较成熟的 Codis 分布式缓存系统进行并发响应时间对比测试实验对比.

3.1 Redis Cluster 功能验证

CacheCloud 可以查看 Redis Cluster 实时全局信息, 包括应用主从节点数, 运行状态等, 如图 7. 在基准测试时, 整个集群的命令分布及命中率都实时反映在了页面上, 说明了系统的可用性和高效性. 图 8 展示了 CacheCloud 对 Redis Cluster 具体节点的动态扩展、Master/Slave 动态切换、添加和删除节点、实例的启动和下线、以及故障转移相关操作等, 实验表明, 各指标状态稳定、功能正常.

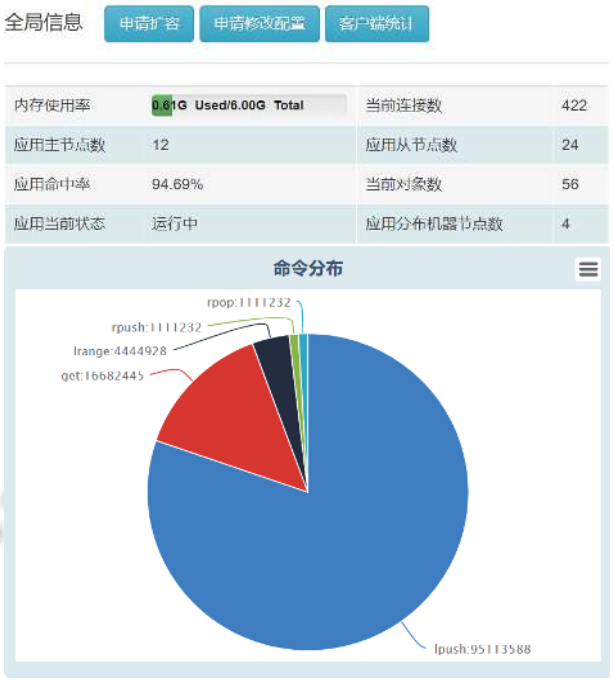

图 7 Redis Clsuter 全局信息图

图 8 RedisCluster-CacheCloud 应用实例管理

3.2 Redis Cluster 性能测试

Redis-benchmark 是官方自带的 Redis 性能测试工具, 可以有效的测试 Redis 服务的性能. 图 9、图 10 和图 11 分别为使用 redis-benchmark 对 Redis Cluster 节点主要命令的 QPS 统计信息, 当请求数 Requests 数分别为 10^0、10^1、10^2、10^3、10^4、10^5、10^6、10^7 和 10^8 时, 基准测试命令 PING_INLINE、PING_BULK、GET、SET、INCR、LPUSH、RPUSH、LPOP、RPOP、SADD、HSET、MSET 等对应的 QPS 趋势图, 表 3 是各命令 QPS 具体值, 从而从实验验证了 Redis 读和写的高效响应速度.

3.3 性能对比 (Redis Cluster vs Codis)

为和业界较成熟的 Codis 作性能对比, 在 4 台虚拟机上同时搭建了 Codis 集群. Codis 成员详细信息见表 4, Codis-group 中主从节点详细信息见表 5.

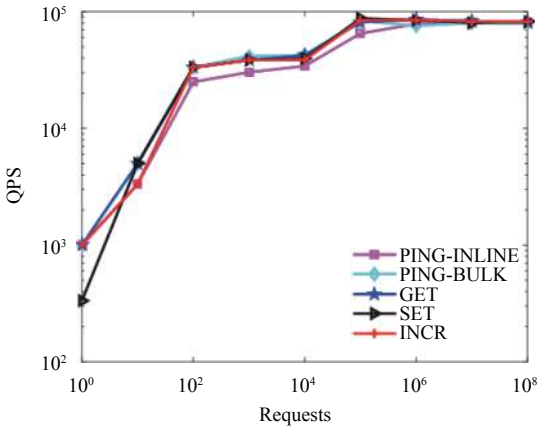

图 9 Redis Cluster 各命令 QPS (a)

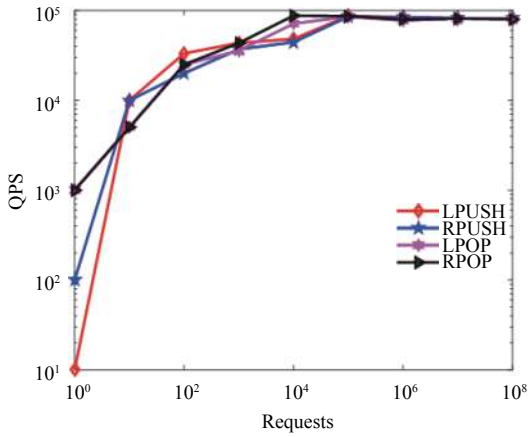

图 10 Redis Cluster 各命令 QPS (b)

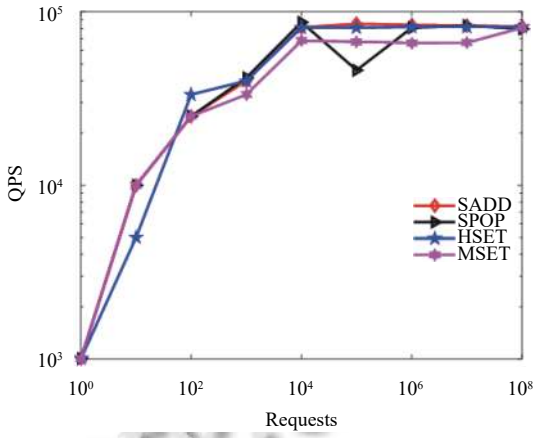

图 11 Redis Cluster 各命令 QPS (c)

Codis 集群成员为:36 个 redisserver(其中 12 个 Master 节点, 24 个 Slave 节点, 分 12 组, 每组 1 主 2 从), 3 个 codis-proxy, 3 个 Zookeeper, 1 个 codis-dashboard, 1 个 codis-fe, 1 个 codis-ha.

对 Redis Cluster 和 Codis 分别以客户端并发数为 10^0、10^1、10^2、10^3、10^4、10^5、10^6 和 10^7 时, 测量 Redis Cluster 和 Codis 集群响应时间. 图 12 为 Redis Cluster 和 Codis 响应时间对比图. 实验结果表明, 当并发请求数为 10 000 及以上时, Redis Cluster 的响应时间明显优于 Codis, 这是因为 Codis 通过代理连接, 性能有所损耗. 因此, 基于 Redis Cluster 的分布式实现方式简单且高效.

表 3 Redis 各命令 QPS

Requests	PING_INLINE	PING_BULK	GET	SET	INCR	LPUSH	RPUSH	LPOP	RPOP	SADD	HSET	MSET
10^0	1000.0	1000.0	1000.0	333.3	1000.0	10.0	100.0	1000.0	1000.0	1000.0	1000.0	1000.0
10^1	3333.3	5000.0	5000.0	5000.0	3333.3	10 000.0	10 000.0	5000.0	5000.0	10 000.0	5000.0	10 000.0
10^2	25 000.0	33 333.3	33 333.3	33 333.3	33 333.3	33 333.3	20 000.0	25 000.0	25 000.0	25 000.0	33 333.3	25 000.0
10^3	30 303.0	41 666.7	38 461.5	38 461.5	38 461.5	43 478.3	37 037.0	35 714.3	43 478.3	40 000.0	40 000.0	33 333.3
10^4	34 246.6	42 372.9	42 372.9	39 525.7	38 610.0	48 309.2	44 444.5	71 428.6	87 719.3	81 300.8	81 300.8	68 027.2
10^5	65 104.2	83 612.0	81 367.0	87 642.4	84 459.5	86 881.0	85 324.2	85 689.8	86 505.2	85 251.5	80 906.2	67 204.3
10^6	78 591.6	75 815.0	85 814.8	84 495.1	84 997.9	84 182.2	83 626.0	77 417.4	79 352.5	84 026.6	81 866.6	66 141.9
10^7	80 690.7	80 008.3	83 127.6	80 425.3	82 598.9	81 716.7	82 298.4	82 027.7	81 402.1	83 236.9	81 853.2	66 277.8
10^8	80 786.7	79 760.6	81 005.4	81 851.9	82 044.0	81 164.6	82 102.5	81 447.0	81 419.4	82 019.4	81 712.6	65335.2

4 结论与展望

本文主要做了三个方面工作: 研究了分布式缓存系统并基于最新版 Redis4.0 构建了 Redis Cluster 分布式缓存系统, 对其可用性、水平扩展、数据一致性及故障转移做了验证; 基于 Redis 自身特点, 从 Linux 层面、Redis Cluster 本身做了配置优化; 集成了开源工具 CacheCloud 对 Redis Cluster 进行监控和管理, 弥补了其本身在可视化管理方面的欠缺. 实验表明, Redis Cluster 在高并发时响应速度要明显优于 Codis, 这一定程度上牺牲了数据强一致性. 同时, 其丰富的数据类

型、数据持久化及备份、消息队列、高扩展性等特性, 使得 Redis Cluster 分布式缓存更加先进和完善.

Redis 4.0 新增了模块机制、部分复制 (PSYNC2.0)、混合 RDB-AOF 持久化策略、更优的缓存驱逐策略, 兼容 NAT 和 Docker, 但在有些方面还有待验证和提升. 下一步将结合实际应用, 在数据一致性、消息队列和 Docker 相结合方面做进一步研究和提升.

表 4 Codis 集群组成成员信息

角色	IP 地址	端口
Zookeeper1/2/3	172.17.0.212/213, 172.17.2.23	2181
Codis-proxy 1	172.17.0.213	19000
Codis-proxy 2	172.17.2.23	19000
Codis-proxy 3	172.17.0.212	19000
Codis-dashboard	172.17.0.213	8080
Codis-fe	172.17.0.213	-
Codis-ha	172.17.0.213	-
Codis-group	Redis Server(36)	Master(12)|Slave(24)

表 5 Codis-group 主从节点信息

CodisGroup	主从节点	IP 地址	端口号
Group1–4	Master(1/2/3/4)	172.17.0.212	7000/7001/7002/7003
	Slave(1.1/1.2/2.1/2.2)	172.17.2.22/23	7002/7003
	Slave(3.1/3.2/4.1/4.2)	172.17.2.22/23	7004/7005
Group 5–8	Master(5/6/7/8)	172.17.0.213	7000/7001/7002/7003
	Slave(5.1/5.2/6.1/6.2/7.1/7.2)	172.17.2.22/23	7006/7007/7008
	Slave(8.1/8.2)	172.17.0.212/213	7008
Group 9–12	Master(9/10/11/12)	172.17.2.22/23	7000/7001
	Slave(9.1/9.2/10.1/10.2) Slave(/11.1/11.2/12.1/12.2)	172.17.0.212/213	7004/7005/7006/7007

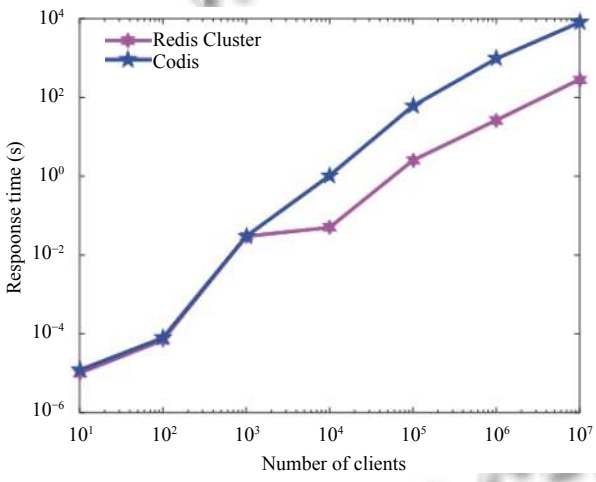

图 12 并发请求响应时间对比图

参考文献

1 于君泽, 曹洪伟, 邱硕. 深入分布式缓存: 从原理到实践. 北京: 机械工业出版社, 2018.

2 Memcached. High-performance, distributed memory object caching system. http://memcached.org/. [2018-01-16].

3 Redis. Partitioning: How to split data among multiple Redis instances. https://redis.io/topics/partitioning. [2018-02-12].

4 Redis. Redis cluster tutorial. https://redis.io/topics/cluster-tutorial. [2017-07-28].

5 Redis. Redis cluster specification. https://redis.io/topics/cluster-spec. [2017-07-28].

6 Twemproxy. Proxy for memcached and Redis. https://github.com/twitter/twemproxy. [2018-01-10].

7 CodisLabs/codis. Proxy based Redis cluster solution. https://github.com/CodisLabs/codis. [2018-02-12].

8 Ji ZL, Ganchev I, O'Droma M, *et al*. A distributed redis framework for use in the UCWW. Proceedings of 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. Shanghai, China. 2014. 241–244. [doi: 10.1109/CyberC.2014.50]

9 王绍东. 基于 Redis Cluster 的分布式内存数据库研究与应用[硕士学位论文]. 广州: 华南理工大学, 2016.

10 搜狐视频 (sohu tv)Redis 私有云平台. https://github.com/sohutv/cachecloud. [2018-02-01].

11 Kimm H, Li ZQ, Kimm H. SCADIS: Supporting reliable scalability in redis replication on demand. Proceedings of 2017 IEEE International Conference on Smart Cloud. New York, NY, USA. 2017. 7–12.

12 Chen SS, Tang XX, Wang HW, *et al*. Towards scalable and reliable in-memory storage system: A case study with redis. Proceedings of 2016 IEEE Trustcom/BigDataSE/ISPA. Tianjin, China. 2016. 1660–1667.

13 付磊, 张益军. Redis 开发与运维. 北京: 机械工业出版社, 2017.