极光博客

JiGuang – Powering Mobile Apps

Redis 存储分片之代理服务Twemproxy 测试

概述

实际业务场景中单点 Redis 容量、并发都是有限的，所以有 Redis Cluster 的需求。

但是官方的 Redis Cluster 一再跳票，还不可用。

只好先使用最简单的方式：Proxy。有很多可选，但在大范围生产使用的， Twitter 开源的 Twemproxy 看起来是个理想的选择 – https://github.com/twitter/twemproxy 。

我们期望的目标：

- tag/alias 缓存集群（现在单点容量支持越来越不够）
- 数据统计时高并发缓存

下面的文章内容也是基于实际生产需要而进行的一系列测试.

测试环境说明

本次测试的机器都是虚拟机，对应的母机配置为 Dell R710 Intel(R) Xeon(R) CPU E5606 @ 2.13GHz 2CPU(单CPU 4核) 32G内存，上面安装了4台虚拟机，配置分别如下：

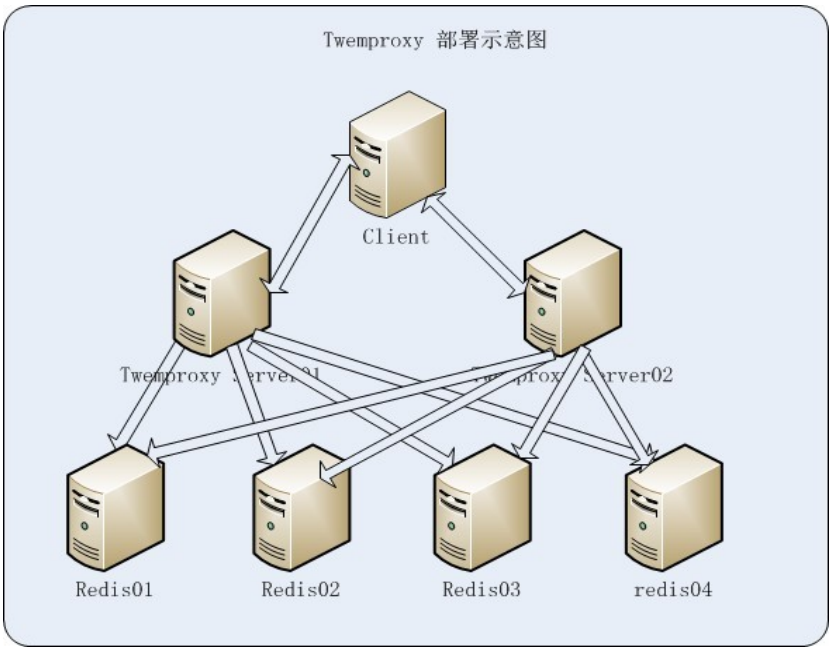
序号	机器IP	配置	部署服务	redis服务数量
1	192.168.2.65	4cpu,8GRam	redis,twemproxy	4个端口分别为(10000,10002,10003,10004)
2	192.168.2.66	4cpu,8GRam	redis	2个端口分别为(10000,10002)
3	192.168.2.67	4cpu,8GRam	redis	2个端口分别为(10000,10002)
4	192.168.2.68	4cpu,4GRam	redis,twemproxy	2个端口分别为(10000,10002)

虚拟机系统： **CentOS release 6.3 (Final)**

Redis版本：2.6.16

Twemproxy版本：nutcracker-0.2.4

部署示意图



下面的测试都是根据上面的配置不同组合来进行测试得出对应的结论。

测试工具：Redis Benchmark.

测试结论

功能

1. 前端使用 Twemproxy 做代理，后端的 Redis 数据能基本上根据 key 来进行比较均衡的分布。
2. 后端一台 Redis 挂掉后，Twemproxy 能够自动摘除。恢复后，Twemproxy 能够自动识别、恢复并重新加入到 Redis 组中重新使用。
3. Redis 挂掉后，后端数据是否丢失依据 Redis 本身的策略配置，与 Twemproxy 基本无关。
4. 如果要新增加一台 Redis，Twemproxy 需要重启才能生效；并且数据不会自动重新 Reblance，需要人工单独写脚本来实现。
5. 如同时部署多个 Twemproxy，配置文件一致（测试配置为**distribution**：ketama,modula），则可以从任意一个读取，都可以正确读取 key对应的值。
6. 多台 Twemproxy 配置一样，客户端分别连接多台 Twemproxy可以在一定条件下提高性能。根据 Server 数量，提高比例在 110-150%之间。
7. 如原来已经有 2 个节点 Redis，后续有增加 2 个 Redis，则数据分布计算与原来的 Redis 分布无关，现有数据如果需要分布均匀的话，需要人工单独处理。
8. 如果 Twemproxy 的后端节点数量发生变化，Twemproxy 相同算法的前提下，原来的数据必须重新处理分布，否则会存在找不到**key**值的情况。

性能

不管 Twemproxy 后端有几台 Redis，前端的单个 Twemproxy 的性能最大也只能和单台 Redis 性能差不多。

Twemproxy介绍

Twemproxy 也叫 nutcraker。是 Twtter 开源的一个 Redis 和 Memcache 代理服务器，主要用于管理 Redis 和 Memcached 集群，减少与Cache 服务器直接连接的数量。

Twemproxy特性：

- 轻量级、快速
- 保持长连接
- 减少了直接与缓存服务器连接的连接数量
- 使用 pipelining 处理请求和响应
- 支持代理到多台服务器上
- 同时支持多个服务器池
- 自动分片数据到多个服务器上
- 实现完整的 memcached 的 ASCII 和再分配协议
- 通过 yaml 文件配置服务器池
- 支持多个哈希模式，包括一致性哈希和分布
- 能够配置删除故障节点
- 可以通过端口监控状态
- 支持 linux, *bsd,os x 和 solaris

Twemproxy安装配置参考官网：https://github.com/twitter/twemproxy 或 附后的 Reference。

启动命令：

```
$/usr/local/twemproxy/sbin/nutcracker -d -c /usr/local/twemproxy/etc/test.yml -i 2000 -o logs/nutcracker.log
```

或者修改源码的 scripts 中的 ini 代码以 service 方式启动。

运行中如果需要查看运行状态使用下面方式，结果为json格式，需要自己格式化。

1.web界面运行启动服务的http://ip:22222查看，如本次测试查看url：http://192.168.2.68:22222/

2.使用nc命令查看 Twemproxy 状态语句：

```
$nc 192.168.2.68 22222|python -mjson.tool
```

Twemproxy支持命令测试

源码的scripts中包含一些脚本可以进行基本功能测试，如下：

scripts目录脚本

```
[root@test66 scripts]$ ls -tlh
total 76K
-rwxr-xr-x 1 root root 496 Oct 23 10:16 pipelined_read.sh
-rwxr-xr-x 1 root root 639 Oct 23 10:16 pipelined_write.sh
-rwxr-xr-x 1 root root 495 Oct 23 10:16 populate_memcached.sh
-rw-r–r– 1 root root 665 Oct 23 10:16 redis-check.py
-rwxr-xr-x 1 root root 48K Oct 23 10:16 redis-check.sh  #检测redis基本命令是否可以使用
-rwxr-xr-x 1 root root 526 Oct 23 10:16 multi_get.sh
-rw-r–r– 1 root root 1.2K Oct 23 10:16 nutcracker.init
-rw-r–r– 1 root root 1.3K Oct 23 10:16 nutcracker.spec
```

从执行结果看，下面命令都可以使用(结合我们常用的命令)

```
del psetex linsert smove zscore dump set llen spop zunionstore exists setbit lpop srandmember eval expire psetex
lpush srem persist setnx lpushx sunion expireat setrange lrange sunionstore expire hdel lrem zadd pttlhexists ltrim zcard
ttl hget rpop zcount type hgetall append hincrby rpush zinterstore bitcount hincrbyfloat

rpushx zrange get hkeys sadd zrangebyscore getbit hlen scard zrank getrange zrem

getset hmset sdiffstore zremrangebyrank incr hset sinter zremrangebyscore incrby

hsetnx sinterstore zrevrange incrbyfloat hvals sismember zrevrangebyscore mget

lindex smembers zrevrank rpoplpush zincrby hmget sdiff
```

测试不支持的几个命令： restore decr decrby

如果生产环境中使用的话，建议先检查是否有无法使用的命令后在决定使用。

Twemproxy 和单机 **Redis** 对比测试

测试方法：使用 Redis 自带压力测试工具 redis-benchmark 测试2-3次，取其中比较好的结果。

注：测试本机上面的 Redis 都是在其他客户端上面执行，避免由于不通过网络导致测试不准确.另外每次运行结果都有不同差距。

执行命令类似如下：

$/usr/local/bin/redis-benchmark -h 192.168.2.68 -p 10000 -c 100 -q

Twmemproxy 配置的后端 Redis 信息如下：

…

distribution： ketama

…

servers:

– 192.168.2.67:10000:1

– 192.168.2.66:10000:1

– 192.168.2.65:10000:1

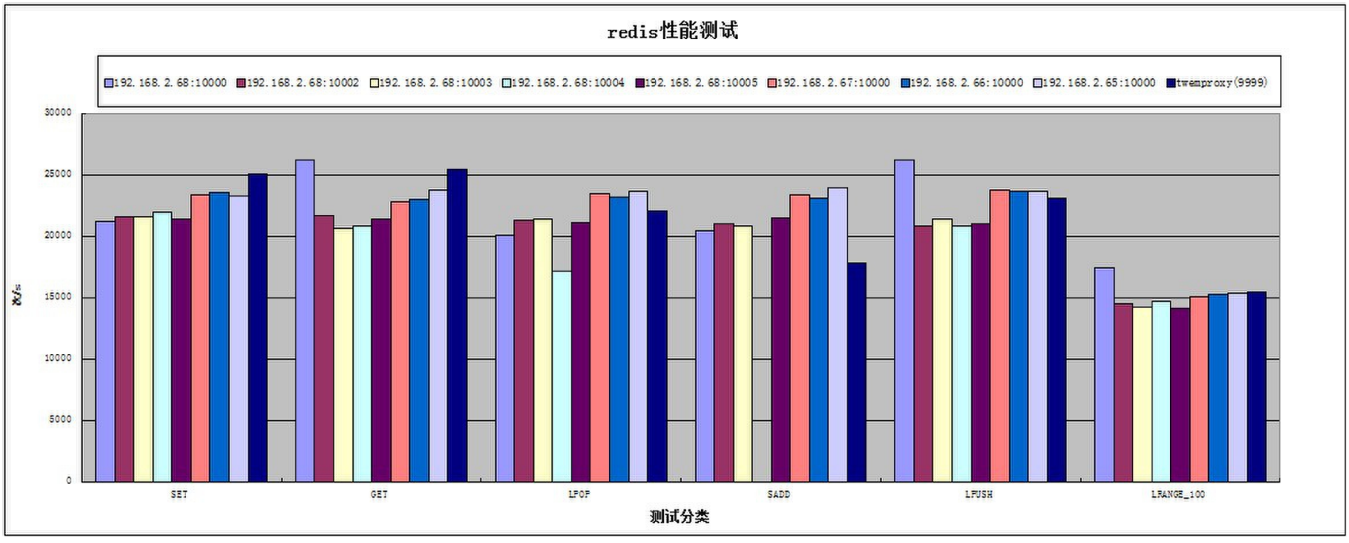
– 192.168.2.68:10000:1

– 192.168.2.68:10002:1

– 192.168.2.68:10003:1

从自带的压力测试工具看，基本上 Twemproxy 的性能和单台 Redis 服务基本差不多。后端自带 Redis 只是为数据更好的 split(由于从实际统计信息看看，压力测试实际上只访问了后端的其中某一台redis服务，故这个结果也属于正常)

命令\IP	2.68:10000	2.68:10002	2.68:10003	2.68:10004	2.68:10005	2.67:10000	2.66:10000	2.65:10000	TWEMPROXY(9999)
SET	21231.42	21598.27	21645.02	22026.43	21413.28	23364.49	23584.91	23310.02	25125.63
GET	26246.72	21739.13	20661.16	20876.83	21367.52	22831.05	23041.47	23809.53	25510.21
LPOP	20120.72	21276.60	21367.52	17152.66	21097.05	23474.18	23201.86	23696.68	22075.05
SADD	20449.90	21052.63	20833.33	20876.83	21551.72	23364.49	23094.69	23923.44	17825.31
LPUSH	26178.01	20833.33	21413.28	20876.83	21008.40	23809.53	23696.68	23696.68	23148.15
LRANGE_100	17452.01	14534.88	14245.01	14684.29	14144.27	15060.24	15267.18	15408.32	15503.88



Twemproxy后端接入不同 **Redis** 数量测试对比

本测试主要验证 Twemproxy 后端接入不同 Redis 数量后，测试的ops比较，结果只能作为参考意义。

设置测试方式，Twemproxy 的分布方式设置为 hash，测试对比如上图。基本上和单台redis性能差不多。实际看后端也只访问一台redis。

分布参数设置为random模式。可以比较均匀分布。

测试类似命令：

/usr/local/redis/bin/redis-benchmark -h 192.168.2.65 -p 9999 -t SET -n 1000000 -c 100 -q

目前环境： 2台机器都搭建 Twemproxy Server，每台端口为9999，19999，29999

目前6台 Redis 分别测试的结果为:

(31438.63+32406.51+37087.86+26886.78+34291.20+32544.67)/6=32442.6083

可以近似看成是 Redis 的单台性能。

测试方式：

1.后端 Redis 节点数量不变，不同 Twemproxy server 测试及多个同时运行测试结果如下：

twemproxy server运行数量(port)	1(A server)	1(B Server)	2	4	6
测试结果(/s)	30278.26	32867.71	35143.28	40176.777	52345.5152

从上面数据可以看出，单台最多也只能达到单个 Redis 的性能；2个节点运行性能增加大概110%左右。4个 server 运行，性能大概增加了123%，6个 server 接入运行160%。

2.前端使用1个 Twemproxy server，后端 Redis 数量分别为2，3，4，5，6来进行压力测试，看测试结果,测试数据如下：

redis节点数	2	3	4	5	6
测试结果(/s)	34882.1	34749.97	32296.61	32438.04	32867.71

从数据可以看出，后端节点数量与 Twemproxy 的性能基本无关，最大性能也就是单个 Redis 的性能。

Twemproxy功能测试

1.Twemproxy 正常访问，后端 Redis 挂掉一台，前端访问是否正常;后端 Redis 挂掉的恢复，不重启 Twemproxy，观察恢复的数据是否有继续增加

从测试结果看，基本正常，由于使用命令/usr/local/bin/redis-cli，无法看到错误信息，不能确认在中断瞬间是否有报错。

从各 Redis 的 keys 数量看，基本可以满足.测试过程中 Redis 中断不到1分钟。从实际数据看只丢失了15个key(总key数量为26W)，可以认为Twemproxy 能够自动摘掉故障的 Redis 及自动恢复。

#初始化redis，各redis中的keys为空

[root@test_2_68 bin]$ sh getkeys.sh

192.168.2.68:10000 keys: 0 or not exists

192.168.2.68:10002 keys: 0 or not exists

192.168.2.68:10003 keys: 0 or not exists

192.168.2.65:10000 keys: 0 or not exists

192.168.2.66:10000 keys: 0 or not exists

192.168.2.67:10000 keys: 0 or not exists

#运行测试程序一段时间后查看

[root@test_2_68 bin]$ sh getkeys.sh

192.168.2.68:10000 keys: 5055

192.168.2.68:10002 keys: 5619

192.168.2.68:10003 keys: 6031

192.168.2.65:10000 keys: 5708

192.168.2.66:10000 keys: 4646

192.168.2.67:10000 keys: 4453

#kill掉一台redis后查看

[root@test_2_68 bin]$ sh getkeys.sh

192.168.2.68:10000 keys: 9045

192.168.2.68:10002 keys: 8860

192.168.2.68:10003 keys: 9552

192.168.2.65:10000 keys: 12047

192.168.2.66:10000 keys: 8920

Could not connect to Redis at 192.168.2.67:10000: Connection refused

192.168.2.67:10000 keys: 0 or not exists

#恢复启动redis后查看

[root@test_2_68 bin]$

[root@test_2_68 bin]$ sh getkeys.sh

192.168.2.68:10000 keys: 14170

192.168.2.68:10002 keys: 11525

192.168.2.68:10003 keys: 13349

192.168.2.65:10000 keys: 15750

192.168.2.66:10000 keys: 12206

192.168.2.67:10000 keys: 9327

#测试程序运行完毕后查看

[root@test_2_68 bin]$ sh getkeys.sh

192.168.2.68:10000 keys: 43186

192.168.2.68:10002 keys: 38090

192.168.2.68:10003 keys: 43069

192.168.2.65:10000 keys: 49291

192.168.2.66:10000 keys: 46428

192.168.2.67:10000 keys: 39921

#测试程序总共插入26W keys。从测试中看基本差15key，可以认为是redis中断期间未插入的。生产上如有容错机制，应可以接受

[root@test_2_68 bin]$ echo “43186+38090+43069+49291+46428+39921″|bc

259985

2.正常装载一部分数据，计算后端各 Redis 的 key 分布情况是否均匀

#总共插入26W个key,通过twemproxy的端口操作

$sh getkeys.sh

192.168.2.68:10000 keys: 42881

192.168.2.68:10002 keys: 37990

192.168.2.68:10003 keys: 42600

192.168.2.65:10000 keys: 48144

192.168.2.66:10000 keys: 45905

192.168.2.67:10000 keys: 42480

#从操作结束后查看各redis的keys看，基本上能够差不多一致，每个redis分布相对比较均匀

---

3. Twemproxy 默认使用(**distribution: ketama**)先使用后端3个节点装载key数量：300708 ，然后后端节点增加到6个(**distribution: ketama**)，在装载300708个 key 值，对比分布趋势：

#插入300708个key，后端节点为3个redis

[root@test_2_68 bin]$ sh getkeys.sh

192.168.2.68:10000 keys: 95673

192.168.2.68:10002 keys: 0 or not exists

192.168.2.65:10002 keys: 0 or not exists

192.168.2.65:10000 keys: 94225

192.168.2.66:10000 keys: 110810

192.168.2.66:10002 keys: 0 or not exists

#继续插入300708个key，后端节点为6个redis

```
[root@test_2_68 bin]$ sh getkeys.sh
192.168.2.68:10000 keys: 140883 #新增45210
192.168.2.68:10002 keys: 51135
192.168.2.65:10002 keys: 49022
192.168.2.65:10000 keys: 144687 #新增50462
192.168.2.66:10000 keys: 167928 #新增57118
192.168.2.66:10002 keys: 47761
#可以看出，新增加后，数据继续增加还是根据key比较均匀分布,与已经存在的数据无关.现有的数据需要自己使用脚本重新分割
```

相同算法下，后续的数据都可以正常提取查找，但是原来已经在 Redis 的数据信息，部分找不到。具体数据如下：

算法规则	总**KEYS**数	未找到的**KEYS**	找到的**KEYS**	找到**KEYS**的比例	备注
distribution: ketama	300708	147757	152951	51.86%	缺省默认的算法
distribution: random	300708	250731	49977	16.62%	
distribution: modula	300708	250408	50300	16.72%	
distribution: ketama	300708	147757	152951	50.86%	缺省默认的算法

从上面可以看出，增加后端 Redis 后，Twemproxy 使用计算新的算法 key 保存的值。从缺省算法的成功率上可以看出，找不到的比例和增加的新的 Redis 点有一定关系(刚好大概一半找不到。我们增加了1倍的节点)

数据一致性测试

测试方法，分别开3个 Twemproxy server(port)，分布格式为 ketama，ketama，random。从其中一个 ketama 分布的插入。然后分别从其他2个不同类型的 server 读取，判断读取值是否正确。

装载数据共300708记录，然后使用 get 方式读取，对应的 key 还是从源文件读取 key。可以看到，类型为 ketama 的2台 server 都可以全部获取到对应的key值，而 random 有2/3获取不到(总记录300708，能获取到key的记录:100347).

Reference

在测试中参考的网络部分资料，本文章中部分内容也有引用，在此感谢！

- 官网：https://github.com/twitter/twemproxy
- 翻译介绍： http://1.breakwang.sinaapp.com/?p=78
- 安装测试：http://blog.mkfree.com/posts/515bce9d975a30cc561dc360#
- Redis : http://redis.io/

本条目发布于2013 年 11 月 12 日 [http://blog.jiguang.cn/redis-twemproxy-benchmark/] 。属于技术文章分类，被贴了Cluster、Redis、Twemproxy 标签。作者是feipeng。

《Redis 存储分片之代理服务Twemproxy 测试》上有11条评论

inetfuture
2013 年 11 月 14 日下午 12:06

好奇你们用什么结构来存储tag/alias呢？

javen
2013 年 11 月 19 日下午 1:11

你的问题非常好！

JPush tag/alias 量非常地大，并且更新频繁，查找也需要快。有点挑战。

前端 Cache 部分就用这个 Redis + Twemproxy。

hello
2014 年 6 月 5 日下午 3:08

没有说到点子上？
后端用数据库存，前端用redis缓存tag结果？

pumpkin zhang
2014 年 5 月 6 日下午 3:21

我只关注“Twemproxy后端接入不同 Redis 数量测试对比”
你这个测试是有问题的，benchmark 是测并发和读写性能的，”-t SET” 时，你开个monitor 看下就知道了，key 都是一个，只会打到后端1 台redis 上，没有测到压力分摊效果。

白凡
2014 年 5 月 8 日下午 12:17

你好,测试数据的redis节点和twemproxy节点,在文章中的描述和table的数据似乎对不上.能给出真实的数据么?

zhangfp
2014 年 5 月 27 日上午 9:36

文章的数据是真实的。后端不同的redis的单个数据测试及汇总，因为有部分数据可能测试好几次去其中的一次，故twemproxy的总数据和列表单独redis的数据汇总可能有稍微一点点差距

mathew
2015 年 1 月 10 日下午 6:49

通过 proxy set 之后,会自动分片,没有数据同步的情况下,如果某特定 key 所在的 redis 挂了,我岂不是取不到对应的 value 了, 是有这个情况吧, 这个情况下就需要从数据库获取了是吧?谢谢

javen
2015 年 3 月 30 日上午 9:33

如果你的 Redis 集群仅用作 Cache，一般是你说的逻辑了。

但为了提高 Redis 集群的可靠性，也可以每个 Redis 节点做 master-slave。这个点出故障了，slave 自动切换为 master。

zhangyicou
2015 年 7 月 31 日上午 11:05

“Redis 节点做 master-slave。这个点出故障了，slave 自动切换为 master。”
这里的master挂掉后，slave被切换为master后，twemproxy代理如何自动更新配置文件的redis.master的ip和端口病重启的呢？如果有资料的话，请给些资料。谢谢

月读

2015 年 1 月 27 日下午 12:00

你好：我们在使用twemproxy的时候出现了一些问题。想问问你们有没有碰到类似的问题。

我们是redis2.7 + nutcracker-0.3.0版本。twemproxy下挂了4个redis实例。
但使用过程中发现会有key在exists和hgetall命令上超时的情况，而且每次总是那固定的几个key超时（socket超时时间放到1分钟还是报超时，排除是网络原因造成的）。
最后通过跟日志发现是twemproxy那边的问题，直连redis各种畅通无阻。无奈最后换成ShardedJedis做客户端分片。

请问你们有碰到过类似的问题吗？是怎么解决的？

javen

2015 年 3 月 30 日上午 9:31

我们貌似没有这种问题。

某特定的几个 Key，背后的原因是什么？

评论已关闭。