09-切片集群：数据增多了，是该加内存还是加实例？

你好，我是蒋德钧。今天我们来学习切片集群。

我曾遇到过这么一个需求：要用Redis保存5000万个键值对，每个键值对大约是512B，为了能快速部署并对外提供服务，我们采用云主机来运行Redis实例，那么，该如何选择云主机的内存容量呢？

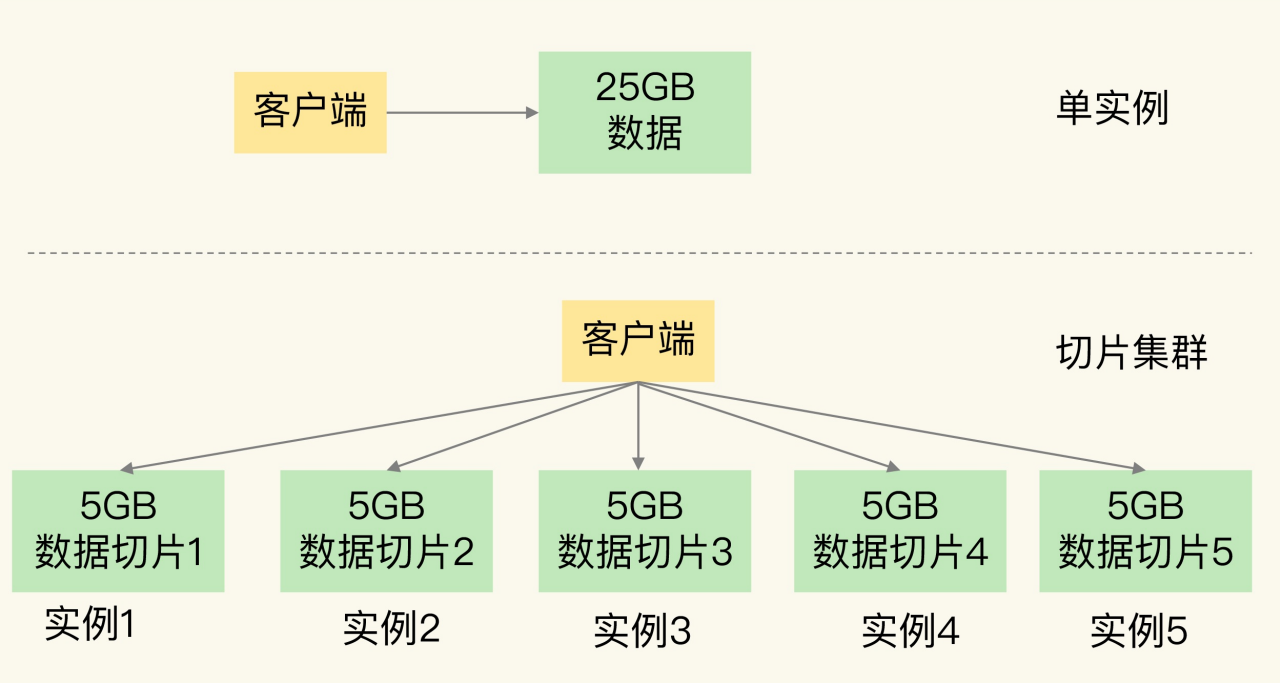
我粗略地计算了一下，这些键值对所占的内存空间大约是25GB（5000万*512B）。所以，当时，我想到的第一个方案就是：选择一台32GB内存的云主机来部署Redis。因为32GB的内存能保存所有数据，而且还留有7GB，可以保证系统的正常运行。同时，我还采用RDB对数据做持久化，以确保Redis实例故障后，还能从RDB恢复数据。

但是，在使用的过程中，我发现，Redis的响应有时会非常慢。后来，我们使用INFO命令查看Redis的latest_fork_usec指标值（表示最近一次fork的耗时），结果显示这个指标值特别高，快到秒级别了。

这跟Redis的持久化机制有关系。在使用RDB进行持久化时，Redis会fork子进程来完成，fork操作的用时和Redis的数据量是正相关的，而fork在执行时会阻塞主线程。数据量越大，fork操作造成的主线程阻塞的时间越长。所以，在使用RDB对25GB的数据进行持久化时，数据量较大，后台运行的子进程在fork创建时阻塞了主线程，于是就导致Redis响应变慢了。

看来，第一个方案显然是不可行的，我们必须要寻找其他的方案。这个时候，我们注意到了Redis的切片集群。虽然组建切片集群比较麻烦，但是它可以保存大量数据，而且对Redis主线程的阻塞影响较小。

切片集群，也叫分片集群，就是指启动多个Redis实例组成一个集群，然后按照一定的规则，把收到的数据划分成多份，每一份用一个实例来保存。回到我们刚刚的场景中，如果把25GB的数据平均分成5份（当然，也可以不做均分），使用5个实例来保存，每个实例只需要保存5GB数据。如下图所示：



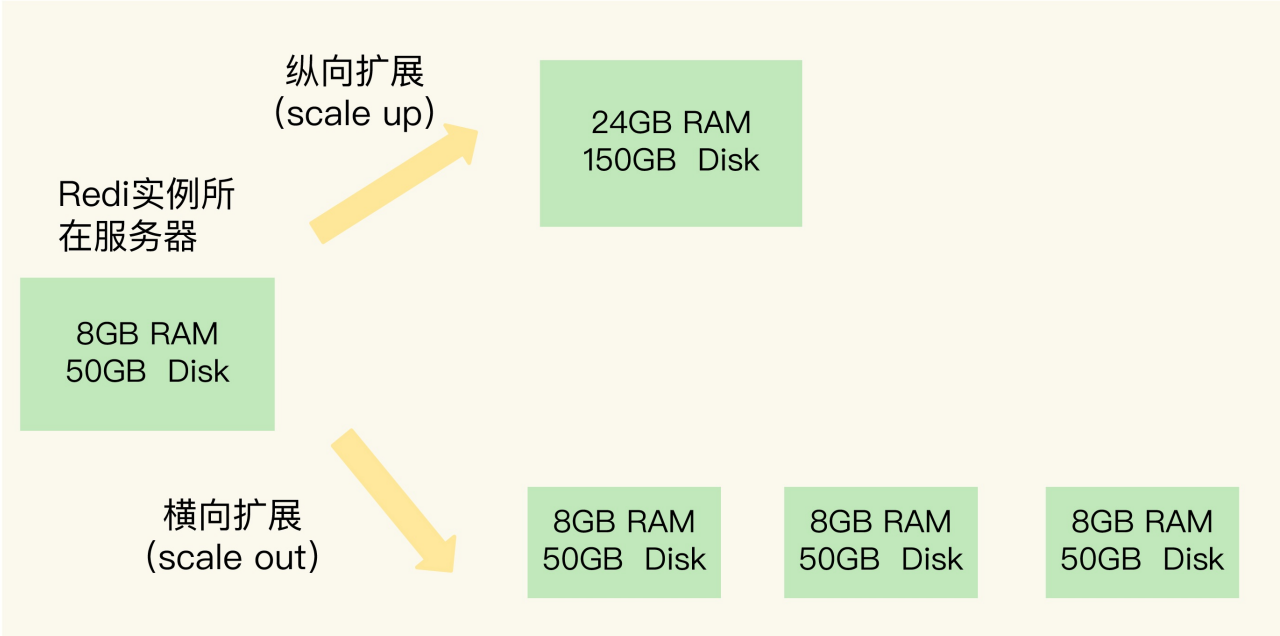
那么，在切片集群中，实例在为5GB数据生成RDB时，数据量就小了很多，fork子进程一般不会给主线程带来较长时间的阻塞。采用多个实例保存数据切片后，我们既能保存25GB数据，又避免了fork子进程阻塞主线程而导致的响应突然变慢。

在实际应用Redis时，随着用户或业务规模的扩展，保存大量数据的情况通常是无法避免的。而切片集群，就是一个非常好的解决方案。这节课，我们就来学习一下。

如何保存更多数据？

在刚刚的案例里，为了保存大量数据，我们使用了大内存云主机和切片集群两种方法。实际上，这两种方法分别对应着Redis应对数据量增多的两种方案：纵向扩展（scale up）和横向扩展（scale out）。

- **纵向扩展**：升级单个Redis实例的资源配置，包括增加内存容量、增加磁盘容量、使用更高配置的CPU。就像下图中，原来的实例内存是8GB，硬盘是50GB，纵向扩展后，内存增加到24GB，磁盘增加到150GB。
- **横向扩展**：横向增加当前Redis实例的个数，就像下图中，原来使用1个8GB内存、50GB磁盘的实例，现在使用三个相同配置的实例。



那么，这两种方式的优缺点分别是什么呢？

首先，纵向扩展的好处是，**实施起来简单、直接**。不过，这个方案也面临两个潜在的问题。

第一个问题是，当使用RDB对数据进行持久化时，如果数据量增加，需要的内存也会增加，主线程fork子进程时就可能会阻塞（比如刚刚的例子中的情况）。不过，如果你不要求持久化保存Redis数据，那么，纵向扩展会是一个不错的选择。

不过，这时，你还要面对第二个问题：**纵向扩展会受到硬件和成本的限制**。这很容易理解，毕竟，把内存从32GB扩展到64GB还算容易，但是，要想扩充到1TB，就会面临硬件容量和成本上的限制了。

与纵向扩展相比，横向扩展是一个扩展性更好的方案。这是因为，要想保存更多的数据，采用这种方案的话，只用增加Redis的实例个数就行了，不用担心单个实例的硬件和成本限制。**在面向百万、千万级别的用户规模时，横向扩展的Redis切片集群会是一个非常好的选择**。

不过，在只使用单个实例的时候，数据存在哪儿，客户端访问哪儿，都是非常明确的，但是，切片集群不可避免地涉及到多个实例的分布式管理问题。要想把切片集群用起来，我们就需要解决两大问题：

- 数据切片后，在多个实例之间如何分布？
- 客户端怎么确定想要访问的数据在哪个实例上？

接下来，我们就一个个地解决。

数据切片和实例的对应分布关系

在切片集群中，数据需要分布在不同实例上，那么，数据和实例之间如何对应呢？这就和接下来我要讲的Redis Cluster方案有关了。不过，我们要先弄明白切片集群和Redis Cluster的联系与区别。

实际上，切片集群是一种保存大量数据的通用机制，这个机制可以有不同的实现方案。在Redis 3.0之前，官方并没有针对切片集群提供具体的方案。从3.0开始，官方提供了一个名为Redis Cluster的方案，用于实现切片集群。Redis Cluster方案中就规定了数据和实例的对应规则。

具体来说，Redis Cluster方案采用哈希槽（Hash Slot，接下来我会直接称之为Slot），来处理数据和实例之间的映射关系。在Redis Cluster方案中，一个切片集群共有16384个哈希槽，这些哈希槽类似于数据分区，每个键值对都会根据它的key，被映射到一个哈希槽中。

具体的映射过程分为两大步：首先根据键值对的key，按照CRC16算法计算一个16 bit的值；然后，再用这个16bit值对16384取模，得到0~16383范围内的模数，每个模数代表一个相应编号的哈希槽。关于CRC16算法，不是这节课的重点，你简单看下链接中的资料就可以了。

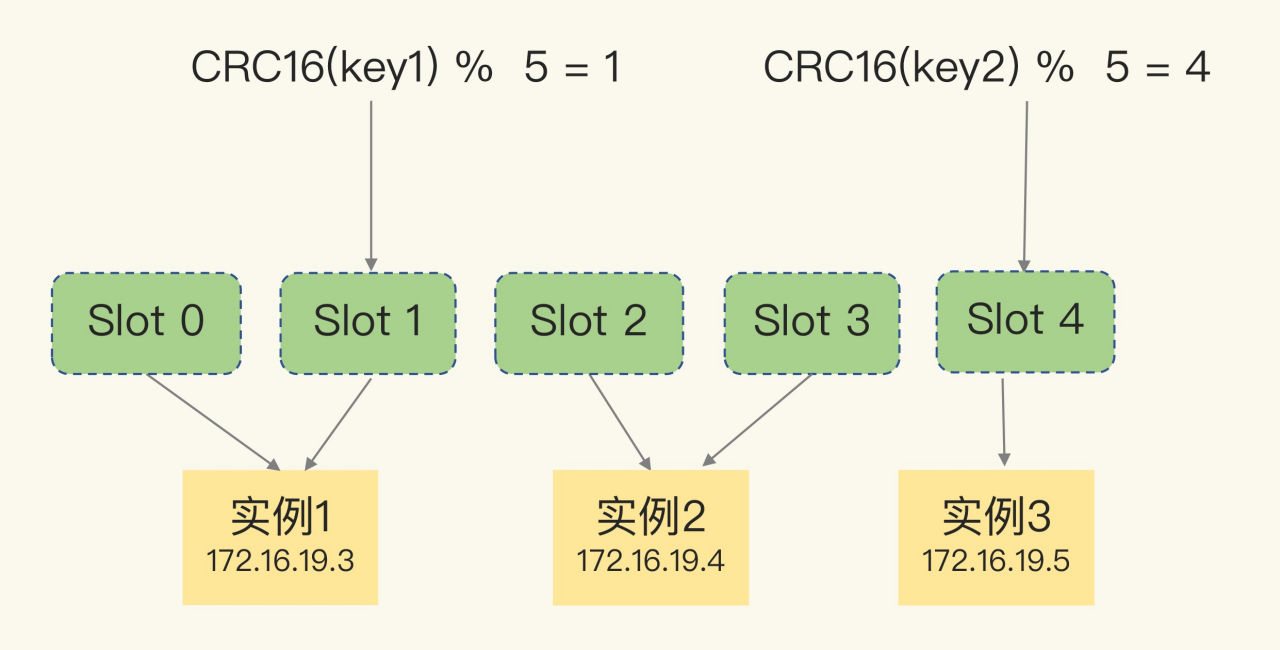
那么，这些哈希槽又是如何被映射到具体的Redis实例上的呢？

我们在部署Redis Cluster方案时，可以使用cluster create命令创建集群，此时，Redis会自动把这些槽平均分布在集群实例上。例如，如果集群中有N个实例，那么，每个实例上的槽个数为16384/N个。

当然， 我们也可以使用cluster meet命令手动建立实例间的连接，形成集群，再使用cluster addslots命令，指定每个实例上的哈希槽个数。

举个例子，假设集群中不同Redis实例的内存大小配置不一，如果把哈希槽均分在各个实例上，在保存相同数量的键值对时，和内存大的实例相比，内存小的实例就会有更大的容量压力。遇到这种情况时，你可以根据不同实例的资源配置情况，使用cluster addslots命令手动分配哈希槽。

为了便于你理解，我画一张示意图来解释一下，数据、哈希槽、实例这三者的映射分布情况。



示意图中的切片集群一共有3个实例，同时假设有5个哈希槽，我们首先可以通过下面的命令手动分配哈希槽：实例1保存哈希槽0和1，实例2保存哈希槽2和3，实例3保存哈希槽4。

```
redis-cli -h 172.16.19.3 –p 6379 cluster addslots 0,1
redis-cli -h 172.16.19.4 –p 6379 cluster addslots 2,3
redis-cli -h 172.16.19.5 –p 6379 cluster addslots 4
```

在集群运行的过程中，key1和key2计算完CRC16值后，对哈希槽总个数5取模，再根据各自的模数结果，就可以被映射到对应的实例1和实例3上了。

另外，我再给你一个小提醒，**在手动分配哈希槽时，需要把16384个槽都分配完，否则Redis集群无法正常工作**。

好了，通过哈希槽，切片集群就实现了数据到哈希槽、哈希槽再到实例的分配。但是，即使实例有了哈希槽的映射信息，客户端又是怎么知道要访问的数据在哪个实例上呢？接下来，我就来和你聊聊。

客户端如何定位数据?

在定位键值对数据时，它所处的哈希槽是可以通过计算得到的，这个计算可以在客户端发送请求时来执行。但是，要进一步定位到实例，还需要知道哈希槽分布在哪个实例上。

一般来说，客户端和集群实例建立连接后，实例就会把哈希槽的分配信息发给客户端。但是，在集群刚刚创建的时候，每个实例只知道自己被分配了哪些哈希槽，是不知道其他实例拥有的哈希槽信息的。

那么，客户端为什么可以在访问任何一个实例时，都能获得所有的哈希槽信息呢？这是因为，Redis实例会把自己的哈希槽信息发给和它相连接的其它实例，来完成哈希槽分配信息的扩散。当实例之间相互连接后，每个实例就有所有哈希槽的映射关系了。

客户端收到哈希槽信息后，会把哈希槽信息缓存在本地。当客户端请求键值对时，会先计算键所对应的哈希槽，然后就可以给相应的实例发送请求了。

但是，在集群中，实例和哈希槽的对应关系并不是一成不变的，最常见的变化有两个：

- 在集群中，实例有新增或删除，Redis需要重新分配哈希槽；
- 为了负载均衡，Redis需要把哈希槽在所有实例上重新分布一遍。

此时，实例之间还可以通过相互传递消息，获得最新的哈希槽分配信息，但是，客户端是无法主动感知这些变化的。这就会导致，它缓存的分配信息和最新的分配信息就不一致了，那该怎么办呢？

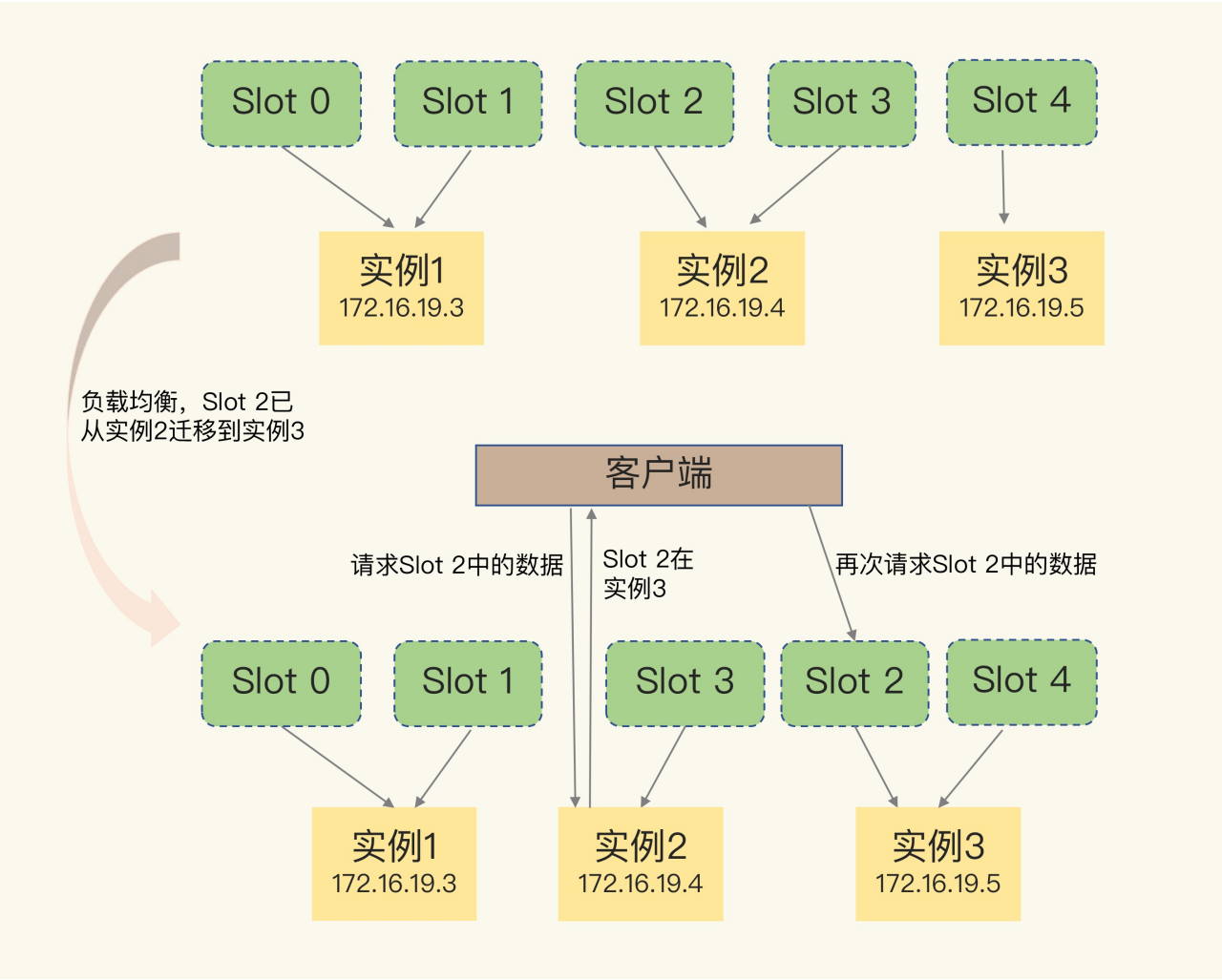
Redis Cluster方案提供了一种**重定向机制，**所谓的“重定向”，就是指，客户端给一个实例发送数据读写操作时，这个实例上并没有相应的数据，客户端要再给一个新实例发送操作命令。

那客户端又是怎么知道重定向时的新实例的访问地址呢？当客户端把一个键值对的操作请求发给一个实例时，如果这个实例上并没有这个键值对映射的哈希槽，那么，这个实例就会给客户端返回下面的MOVED命令响应结果，这个结果中就包含了新实例的访问地址。

```
GET hello:key
(error) MOVED 13320 172.16.19.5:6379
```

其中，MOVED命令表示，客户端请求的键值对所在的哈希槽13320，实际是在172.16.19.5这个实例上。通过返回的MOVED命令，就相当于把哈希槽所在的新实例的信息告诉给客户端了。这样一来，客户端就可以直接和172.16.19.5连接，并发送操作请求了。

我画一张图来说明一下，MOVED重定向命令的使用方法。可以看到，由于负载均衡，Slot 2中的数据已经从实例2迁移到了实例3，但是，客户端缓存仍然记录着“Slot 2在实例2”的信息，所以会给实例2发送命令。实例2给客户端返回一条MOVED命令，把Slot 2的最新位置（也就是在实例3上），返回给客户端，客户端就会再次向实例3发送请求，同时还会更新本地缓存，把Slot 2与实例的对应关系更新过来。



需要注意的是，在上图中，当客户端给实例2发送命令时，Slot 2中的数据已经全部迁移到了实例3。在实际应用时，如果Slot 2中的数据比较多，就可能会出现一种情况：客户端向实例2发送请求，但此时，Slot 2中的数据只有一部分迁移到了实例3，还有部分数据没有迁移。在这种迁移部分完成的情况下，客户端就会收到一条ASK报错信息，如下所示：

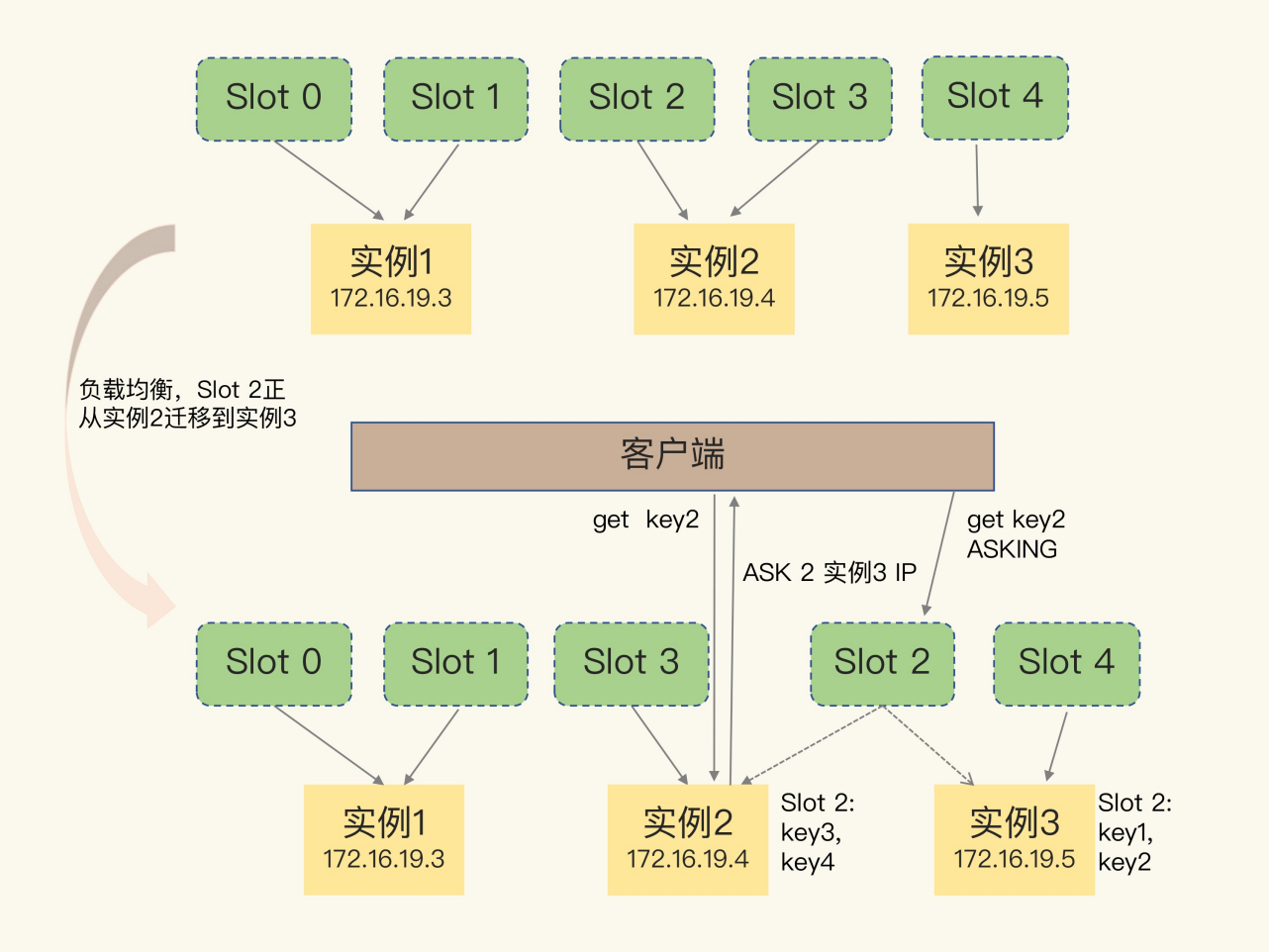
```
GET hello:key
(error) ASK 13320 172.16.19.5:6379
```

这个结果中的ASK命令就表示，客户端请求的键值对所在的哈希槽13320，在172.16.19.5这个实例上，但是这个哈希槽正在迁移。此时，客户端需要先给172.16.19.5这个实例发送一个ASKING命令。这个命令的意思是，让这个实例允许执行客户端接下来发送的命令。然后，客户端再向这个实例发送GET命令，以读取数据。

看起来好像有点复杂，我再借助图片来解释一下。

在下图中，Slot 2正在从实例2往实例3迁移，key1和key2已经迁移过去，key3和key4还在实例2。客户端向实例2请求key2后，就会收到实例2返回的ASK命令。

ASK命令表示两层含义：第一，表明Slot数据还在迁移中；第二，ASK命令把客户端所请求数据的最新实例地址返回给客户端，此时，客户端需要给实例3发送ASKING命令，然后再发送操作命令。



和MOVED命令不同，**ASK命令并不会更新客户端缓存的哈希槽分配信息**。所以，在上图中，如果客户端再次请求Slot 2中的数据，它还是会给实例2发送请求。这也就是说，ASK命令的作用只是让客户端能给新实例发送一次请求，而不像MOVED命令那样，会更改本地缓存，让后续所有命令都发往新实例。

小结

这节课，我们学习了切片集群在保存大量数据方面的优势，以及基于哈希槽的数据分布机制和客户端定位键值对的方法。

在应对数据量扩容时，虽然增加内存这种纵向扩展的方法简单直接，但是会造成数据库的内存过大，导致性能变慢。Redis切片集群提供了横向扩展的模式，也就是使用多个实例，并给每个实例配置一定数量的哈希槽，数据可以通过键的哈希值映射到哈希槽，再通过哈希槽分散保存到不同的实例上。这样做的好处是扩展性好，不管有多少数据，切片集群都能应对。

另外，集群的实例增减，或者是为了实现负载均衡而进行的数据重新分布，会导致哈希槽和实例的映射关系发生变化，客户端发送请求时，会收到命令执行报错信息。了解了MOVED和ASK命令，你就不会为这类报错而头疼了。

我刚刚说过，在Redis 3.0 之前，Redis官方并没有提供切片集群方案，但是，其实当时业界已经有了一些切片集群的方案，例如基于客户端分区的ShardedJedis，基于代理的Codis、Twemproxy等。这些方案的应用早于Redis Cluster方案，在支撑的集群实例规模、集群稳定性、客户端友好性方面也都有着各自的优势，我会在后面的课程中，专门和你聊聊这些方案的实现机制，以及实践经验。这样一来，当你再碰到业务发展带来的数据量巨大的难题时，就可以根据这些方案的特点，选择合适的方案实现切片集群，以应对业务需求了。

每课一问

按照惯例，给你提一个小问题：Redis Cluster方案通过哈希槽的方式把键值对分配到不同的实例上，这个过程需要对键值对的key做CRC计算，然后再和哈希槽做映射，这样做有什么好处吗？如果用一个表直接把键值对和实例的对应关系记录下来（例如键值对1在实例2上，键值对2在实例1上），这样就不用计算key和哈希槽的对应关系了，只用查表就行了，Redis为什么不这么做呢？

欢迎你在留言区畅所欲言，如果你觉得有收获，也希望你能帮我把今天的内容分享给你的朋友，帮助更多人解决切片集群的问题。

精选留言：

- Kaito 2020-08-24 13:56:49

  Redis Cluster不采用把key直接映射到实例的方式，而采用哈希槽的方式原因：

  1、整个集群存储key的数量是无法预估的，key的数量非常多时，直接记录每个key对应的实例映射关系，这个映射表会非常庞大，这个映射表无论是存储在服务端还是客户端都占用了非常大的内存空间。

  2、Redis Cluster采用无中心化的模式（无proxy，客户端与服务端直连），客户端在某个节点访问一个key，如果这个key不在这个节点上，这个节点需要有纠正客户端路由到正确节点的能力（MOVED响应），这就需要节点之间互相交换路由表，每个节点拥有整个集群完整的路由关系。如果存储的都是key与实例的对应关系，节点之间交换信息也会变得非常庞大，消耗过多的网络资源，而且就算交换完成，相当于每个节点都需要额外存储其他节点的路由表，内存占用过大造成资源浪费。

  3、当集群在扩容、缩容、数据均衡时，节点之间会发生数据迁移，迁移时需要修改每个key的映射关系，维护成本高。

  4、而在中间增加一层哈希槽，可以把数据和节点解耦，key通过Hash计算，只需要关心映射到了哪个哈希槽，然后再通过哈希槽和节点的映射表找到节点，相当于消耗了很少的CPU资源，不但让数据分布更均匀，还可以让这个映射表变得很小，利于客户端和服务端保存，节点之间交换信息时也变得轻量。

  5、当集群在扩容、缩容、数据均衡时，节点之间的操作例如数据迁移，都以哈希槽为基本单位进行操作，简化了节点扩容、缩容的难度，便于集群的维护和管理。

  另外，我想补充一下Redis集群相关的知识，以及我的理解：

  Redis使用集群方案就是为了解决单个节点数据量大、写入量大产生的性能瓶颈的问题。多个节点组成一个集群，可以提高集群的性能和可靠性，但随之而来的就是集群的管理问题，最核心问题有2个：请求路由、数据迁移（扩容/缩容/数据平衡）。

  1、请求路由：一般都是采用哈希槽的映射关系表找到指定节点，然后在这个节点上操作的方案。

  Redis Cluster在每个节点记录完整的映射关系(便于纠正客户端的错误路由请求)，同时也发给客户端让客户端缓存一份，便于客户端直接找到指定节点，客户端与服务端配合完成数据的路由，这需要业务在使用Redis Cluster时，必须升级为集群版的SDK才支持客户端和服务端的协议交互。

  其他Redis集群化方案例如Twemproxy、Codis都是中心化模式（增加Proxy层），客户端通过Proxy对整个集群进行操作，Proxy后面可以挂N多个Redis实例，Proxy层维护了路由的转发逻辑。操作Proxy就像是操作一个普通Redis一样，客户端也不需要更换SDK，而Redis Cluster是把这些路由逻辑做在了SDK中。当然，增加一层Proxy也会带来一定的性能损耗。

2、数据迁移：当集群节点不足以支撑业务需求时，就需要扩容节点，扩容就意味着节点之间的数据需要做迁移，而迁移过程中是否会影响到业务，这也是判定一个集群方案是否成熟的标准。

Twemproxy不支持在线扩容，它只解决了请求路由的问题，扩容时需要停机做数据重新分配。而Redis Cluster和Codis都做到了在线扩容（不影响业务或对业务的影响非常小），重点就是在数据迁移过程中，客户端对于正在迁移的key进行操作时，集群如何处理？还要保证响应正确的结果？

Redis Cluster和Codis都需要服务端和客户端/Proxy层互相配合，迁移过程中，服务端针对正在迁移的key，需要让客户端或Proxy去新节点访问（重定向），这个过程就是为了保证业务在访问这些key时依旧不受影响，而且可以得到正确的结果。由于重定向的存在，所以这个期间的访问延迟会变大。等迁移完成之后，Redis Cluster每个节点会更新路由映射表，同时也会让客户端感知到，更新客户端缓存。Codis会在Proxy层更新路由表，客户端在整个过程中无感知。

除了访问正确的节点之外，数据迁移过程中还需要解决异常情况（迁移超时、迁移失败）、性能问题（如何让数据迁移更快、bigkey如何处理），这个过程中的细节也很多。

Redis Cluster的数据迁移是同步的，迁移一个key会同时阻塞源节点和目标节点，迁移过程中会有性能问题。而Codis提供了异步迁移数据的方案，迁移速度更快，对性能影响最小，当然，实现方案也比较复杂。 [49赞]

- Darren 2020-08-24 09:51:00
  我认为有以下几点：
  1、存在表的话，存在单点问题，即使部署多份，存在数据一致性问题，提高了复杂度；
  2、即使解决了第一个问题，但是Redis主打的是快，表的读写并发问题处理；
  3、key对应的是实例，对应关系粒度太大；

  4、用key做hash避免依赖别的功能或者服务，提供了整体的内聚性；
  5、在做Redis集群，为了数据分配均匀，进行一致性哈希的时候，虚拟节点和真实节点之间还有对应关系，存在多级映射关系，增加了耗时，影响Redis主线程的执行速度。 [7赞]

- 小宇子2B 2020-08-24 07:57:33
  1.让key在多个实例上分布更均匀
  2.需要rehash的时候，还要去修改这个对应关系表，代价有点大
  3.存在表里，key的数量太大，表的维护是个问题 [4赞]

- Monday 2020-08-24 23:34:38
  思考题：
  1、使用CRC这个hash函数原因
  1）hash从查询slot的时间复杂度上讲，CRC为O(1)；存表（理解为有序数组或跳表），再快也就是O(Log n)
  2）hash从存储映射关系的空间复杂度上讲，CRC为O(1)；存表，至少也得O(n)，若是跳表还得存额外的索引

  另外我有两个问题咨询下老师，望答复，谢谢！
  1、Redis切片集群使用CRC这个hash函数先获取到具体的slot，然后在具体的slot中，是不是再通过另一个hash函数访问Key对应的值？类似于Java结构：HashMap<String, HashMap<String,Object>>
  2、Redis的slot数量为什么是16384=2^14个，如果用2B来存长度也是2^16=65536个啊？

  [1赞]

- 扩散性百万咸面包 2020-08-24 21:55:20
  隔壁分布式数据库也讲到了分片，但是它里面提到现代的分布式数据库实现分片基本都是Range-based的，能够实现分片的动态调度，适合互联网的场景。那为什么Redis依旧要用Hash-based的设计方式呢？是为了更高并发的写入性能吗？ [1赞]

  作者回复2020-08-24 23:28:33
  如果是根据某个字段的取值范围进行range-based分片，有可能的一个问题是：某个range内的记录数量很多，这就会导致相应的数据分片比较大，一般也叫做数据倾斜。对这个数据分片的访问量也可能大，导致负载不均衡。

  基于记录key进行哈希后再取模，好处是能把数据打得比较散，不太容易引起数据倾斜，还是为了访问时请求负载能在不同数据分片分布地均衡些，提高访问性能。

- 优秀的吉吉国王 2020-08-24 09:02:31
  槽相当于虚拟节点，这样可以灵活的扩缩容，因为是按槽数分的key，这是主要的优点，而且只需要存槽与机器实例的对应关系，不用每个实例都存一份所有的键对应的实例，节省内存 [1赞]

- test 2020-08-24 08:54:57
  用表保存key和实例对应关系的话表数据量太大了，而且不灵活。 [1赞]

- 个大大 2020-08-26 22:37:48
  咨询老师一个问题，我的理解16384个槽，每个槽相当于哈希表的一个桶，如果当切片集群的键值对达到几十万甚至更多的时候，哈希碰撞是不是无法避免，导致链式存储数据过长，单机还可以通过扩容rehash解决，切片集群有没有什么解决方式呢？希望老师可以解决一下疑惑，谢谢。

- 范闲 2020-08-26 11:42:26
  对于Redis而言，切片管理有两种方式
  1.中心化方式，典型的Codis.Codis是首先对key进行crc32的计算,然后对于1024求模(这个值可配置)得到槽位所在Redis实例信息,然后进行处理~~所有的槽位信息都是保存在zookeeper/etcd的中间件，发生变更的时候会同步到proxy中. 另外对于数据迁移而言，会有migrating的状态做管理, codis本身建议的key的value大小小于1M,这样可以加速数据迁移~~大key会有阻塞情况
  2.去中心化方式，典型的Redis Cluster.redis cluster由Node组成，Node中会保存槽位的实例信息。如果请求不在当前Node中，会出现Moved响应，告诉客户端应该去请求key所在的节点。slot = crc16(key)%16384.

- riryoutexi 2020-08-26 11:21:13
  请问 redis Cluster，扩容的时候，迁移的策略是什么？迁移的过程中怎么访问这些正在迁移中的key？

- 三木子 2020-08-26 10:07:28
  如果用一个表直接把键值对和实例的对应关系记录下来，这样就会消耗一定的空间存放。

- Mr.蜜 2020-08-25 23:41:13
  key的数量并不固定，所以key表的维护、集群间的key表同步都会比较复杂，增加了key与整个集群的耦合度，无用开销也太大。
  同时有个问题想请教老师，为什么slot是16384个？

- yyl 2020-08-25 12:28:10
  解答：
  1. 引入哈希槽，将key的分布与具体的Redis实例解耦，有利于Redis数据的均衡分布。

2. 不采用哈希槽的话，Redis实例的扩容和缩容，需要针对无规则的key进行处理，实现数据迁移。此外，需要引入负载均衡中间件来协调各个Redis实例的均衡响应，确保数据的均匀分布；中间件需要保存key的分布状态，保证key的查询能够得到响应。
增加了Redis系统的复杂性 与 可维护性。

看到问题后的第一反应，理解不够深刻，讲述不够清楚。贵在思考啦😉

作者回复2020-08-26 08:58:53
回答的挺好，对hash算法可用于打散键值对分布的理解到位！

- Vincent 2020-08-24 23:17:13
请问老师两个问题
1.为什么要采用crc16
2.各个节点的ping/pong消息大小大概会有多大呢，怎么来计算呢

- MrPolo 2020-08-24 17:28:36
先前在研究 redis cluster 時有注意到,若改為 cluster mode 後有些 command 會無法使用,這部分請問老師會在後面課程講解嗎?

- 流浪地球 2020-08-24 17:02:16
请问老师，集群中的每个切片都是主从配置的吗?

作者回复2020-08-24 23:49:47
切片集群中每个切片可以配置从库，也可以不配置。不过一般生产环境中还是建议对每个切片做主从配置。

可以使用cluster replicate命令进行配置。

- 篮球不是这么打的 2020-08-24 17:02:05
你好老师，请问下如何算的Redis中每个键值对的大小

- 那时刻 2020-08-24 16:03:34
老师，请问一个Redis延迟的问题。

Redis情况：
单实例Redis，内存10G，最大吞吐量在25000ops/second。数据持久化方式：AOF - 每秒fsync一次。
高峰期的ops是每秒8000，key的数量是4万，内存占用量是2.5G。

遇到的问题：
低峰期Redis延迟0.125ms左右，但是在高峰期的时候，延迟比较大，有1ms左右。

通过INFO命令，看到latest_fork_usec是0。INFO命令其它的信息，我也看了下，感觉都还正常。

我能想到的是通过增加Redis分片机制，来缓解压力。

请问老师，针对这种情况，其一，如何来诊断Redis延迟高的情况呢？其二，如何来缓解Redis延迟高?

- williamcai 2020-08-24 12:41:22
如果按照键值存储的话，数据量很大，导致该映射占用空间会很大，进而影响查询速度，采用映射卡擦的方式有些与多级目录有异曲同工之妙

- 小喵喵 2020-08-24 11:18:28
  请教小老师，我对ASKING命令不太理解，我觉得会返回一个包含bool值的信息，告诉客户端是否可以从这个实例中获取数据，若可以，然后客户端重新发送请求来获取数据，不知道这么理解对不对，请老师解惑。谢谢。