37-数据分布优化：如何应对数据倾斜？

你好，我是蒋德钧。

在切片集群中，数据会按照一定的分布规则分散到不同的实例上保存。比如，在使用Redis Cluster或Codis时，数据都会先按照CRC算法的计算值对Slot（逻辑槽）取模，同时，所有的Slot又会由运维管理员分配到不同的实例上。这样，数据就被保存到相应的实例上了。

虽然这种方法实现起来比较简单，但是很容易导致一个问题：数据倾斜。

数据倾斜有两类。

- **数据量倾斜**：在某些情况下，实例上的数据分布不均衡，某个实例上的数据特别多。
- **数据访问倾斜**：虽然每个集群实例上的数据量相差不大，但是某个实例上的数据是热点数据，被访问得非常频繁。

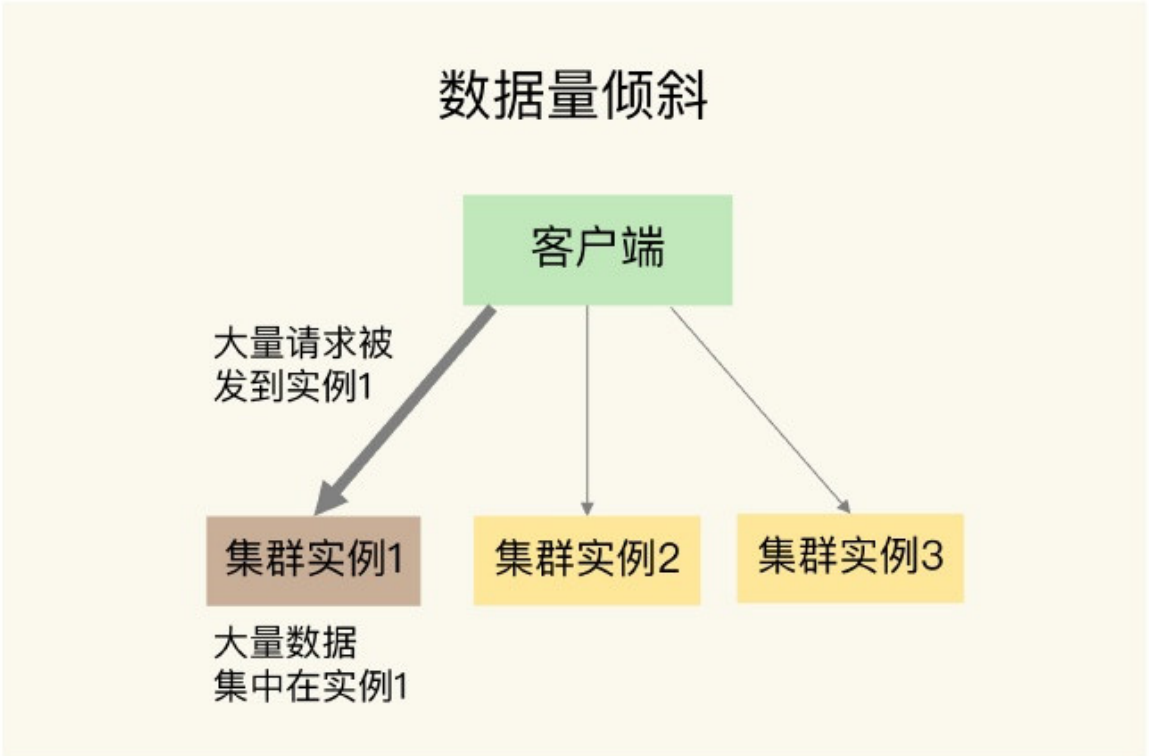
如果发生了数据倾斜，那么保存了大量数据，或者是保存了热点数据的实例的处理压力就会增大，速度变慢，甚至还可能会引起这个实例的内存资源耗尽，从而崩溃。这是我们在应用切片集群时要避免的。

今天这节课，我就来和你聊聊，这两种数据倾斜是怎么发生的，我们又该怎么应对。

数据量倾斜的成因和应对方法

首先，我们来看数据量倾斜的成因和应对方案。

当数据量倾斜发生时，数据在切片集群的多个实例上分布不均衡，大量数据集中到了一个或几个实例上，如下图所示：



那么，数据量倾斜是怎么产生的呢？这主要有三个原因，分别是某个实例上保存了bigkey、Slot分配不均衡以及Hash Tag。接下来，我们就一个一个来分析，同时我还会给你讲解相应的解决方案。

bigkey导致倾斜

第一个原因是，某个实例上正好保存了bigkey。bigkey的value值很大（String类型），或者是bigkey保存了大量集合元素（集合类型），会导致这个实例的数据量增加，内存资源消耗也相应增加。

而且，bigkey的操作一般都会造成实例IO线程阻塞，如果bigkey的访问量比较大，就会影响到这个实例上的其它请求被处理的速度。

其实，bigkey已经是我们课程中反复提到的一个关键点了。为了避免bigkey造成的数据倾斜，一个根本的应对方法是，**我们在业务层生成数据时，要尽量避免把过多的数据保存在同一个键值对中**。

此外，**如果bigkey正好是集合类型，我们还有一个方法，就是把bigkey拆分成很多个小的集合类型数据，分散保存在不同的实例上**。

我给你举个例子。假设Hash类型集合user:info保存了100万个用户的信息，是一个bigkey。那么，我们就可以按照用户ID的范围，把这个集合拆分成10个小集合，每个小集合只保存10万个用户的信息（例如小集合1保存的是ID从1到10万的用户信息，小集合2保存的是ID从10万零1到20万的用户）。这样一来，我们就可以把一个bigkey化整为零、分散保存了，避免了bigkey给单个切片实例带来的访问压力。

需要注意的是，当bigkey访问量较大时，也会造成数据访问倾斜，我一会儿再给你讲具体怎么应对。

接下来，我们再来看导致数据量倾斜的第二个原因：Slot分配不均衡。

Slot分配不均衡导致倾斜

如果集群运维人员没有均衡地分配Slot，就会有大量的数据被分配到同一个Slot中，而同一个Slot只会在一个实例上分布，这就会导致，大量数据被集中到一个实例上，造成数据倾斜。

我以Redis Cluster为例，来介绍下Slot分配不均衡的情况。

Redis Cluster一共有16384个Slot，假设集群一共有5个实例，其中，实例1的硬件配置较高，运维人员在给实例分配Slot时，就可能会给实例1多分配些Slot，把实例1的资源充分利用起来。

但是，我们其实并不知道数据和Slot的对应关系，这种做法就可能会导致大量数据正好被映射到实例1上的Slot，造成数据倾斜，给实例1带来访问压力。

为了应对这个问题，我们可以通过运维规范，在分配之前，我们就要避免把过多的Slot分配到同一个实例。如果是已经分配好Slot的集群，我们可以先查看Slot和实例的具体分配关系，从而判断是否有过多的Slot集中到了同一个实例。如果有的话，就将部分Slot迁移到其它实例，从而避免数据倾斜。

不同集群上查看Slot分配情况的方式不同：如果是Redis Cluster，就用CLUSTER SLOTS命令；如果是Codis，就可以在codis dashboard上查看。

比如说，我们执行CLUSTER SLOTS命令查看Slot分配情况。命令返回结果显示，Slot 0 到Slot 4095被分配

到了实例192.168.10.3上，而Slot 12288到Slot 16383被分配到了实例192.168.10.5上。

```
127.0.0.1:6379> cluster slots
1) 1) (integer) 0
   2) (integer) 4095
   3) 1) "192.168.10.3"
      2) (integer) 6379
2) 1) (integer) 12288
   2) (integer) 16383
   3) 1) "192.168.10.5"
      2) (integer) 6379
```

如果某一个实例上有太多的Slot，我们就可以使用迁移命令把这些Slot迁移到其它实例上。在Redis Cluster中，我们可以使用3个命令完成Slot迁移。

1. CLUSTER SETSLOT：使用不同的选项进行三种设置，分别是设置Slot要迁入的目标实例，Slot要迁出的源实例，以及Slot所属的实例。
2. CLUSTER GETKEYSINSLOT：获取某个Slot中一定数量的key。
3. MIGRATE：把一个key从源实例实际迁移到目标实例。

我来借助一个例子，带你了解下这三个命令怎么用。

假设我们要把Slot 300从源实例（ID为3）迁移到目标实例（ID为5），那要怎么做呢？

实际上，我们可以分成5步。

第1步，我们先在目标实例5上执行下面的命令，将Slot 300的源实例设置为实例3，表示要从实例3上迁入Slot 300。

```
CLUSTER SETSLOT 300 IMPORTING 3
```

第2步，在源实例3上，我们把Slot 300的目标实例设置为5，这表示，Slot 300要迁出到实例5上，如下所示：

```
CLUSTER SETSLOT 300 MIGRATING 5
```

第3步，从Slot 300中获取100 个key。因为Slot中的key数量可能很多，所以我们需要在客户端上多次执行下面的这条命令，分批次获得并迁移key。

```
CLUSTER GETKEYSINSLOT 300 100
```

第4步，我们把刚才获取的100个key中的key1迁移到目标实例5上（IP为192.168.10.5），同时把要迁入的数据库设置为0号数据库，把迁移的超时时间设置为timeout。我们重复执行MIGRATE命令，把100个key都迁移完。

```
MIGRATE 192.168.10.5 6379 key1 0 timeout
```

最后，我们重复执行第3和第4步，直到Slot中的所有key都迁移完成。

从Redis 3.0.6开始，你也可以使用KEYS选项，一次迁移多个key（key1、2、3），这样可以提升迁移效率。

```
MIGRATE 192.168.10.5 6379 "" 0 timeout KEYS key1 key2 key3
```

对于Codis来说，我们可以执行下面的命令进行数据迁移。其中，我们把dashboard组件的连接地址设置为ADDR，并且把Slot 300迁移到编号为6的codis server group上。

```
codis-admin --dashboard=ADDR -slot-action --create --sid=300 --gid=6
```

除了bigkey和Slot分配不均衡会导致数据量倾斜，还有一个导致倾斜的原因，就是使用了Hash Tag进行数据切片。

Hash Tag导致倾斜

Hash Tag是指加在键值对key中的一对花括号{}。这对括号会把key的一部分括起来，客户端在计算key的CRC16值时，只对Hash Tag花括号中的key内容进行计算。如果没用Hash Tag的话，客户端计算整个key的CRC16的值。

举个例子，假设key是user:profile:3231，我们把其中的3231作为Hash Tag，此时，key就变成了user:profile:{3231}。当客户端计算这个key的CRC16值时，就只会计算3231的CRC16值。否则，客户端会计算整个“user:profile:3231”的CRC16值。

使用Hash Tag的好处是，如果不同key的Hash Tag内容都是一样的，那么，这些key对应的数据会被映射到同一个Slot中，同时会被分配到同一个实例上。

下面这张表就显示了使用Hash Tag后，数据被映射到相同Slot的情况，你可以看下。

数据key	哈希计算	对应的Slot
user:profile:{3231}	CRC16('3231') mod 16384	1024
user:profile:{5328}	CRC16('5328') mod 16384	3210
user:order:{3231}	CRC16('3231') mod 16384	1024
user:order:{5328}	CRC16('5328') mod 16384	3210

其中，user:profile:{3231}和user:order:{3231}的Hash Tag一样，都是3231，它们的CRC16计算值对16384取模后的值也是一样的，所以就对应映射到了相同的Slot 1024中。user:profile:{5328}和user:order:{5328}也是相同的映射结果。

那么，Hash Tag一般用在什么场景呢？其实，它主要是用在Redis Cluster和Codis中，支持事务操作和范围查询。因为Redis Cluster和Codis本身并不支持跨实例的事务操作和范围查询，当业务应用有这些需求时，就只能先把这些数据读取到业务层进行事务处理，或者是逐个查询每个实例，得到范围查询的结果。

这样操作起来非常麻烦，所以，我们可以使用Hash Tag把要执行事务操作或是范围查询的数据映射到同一个实例上，这样就能很轻松地实现事务或范围查询了。

但是，使用Hash Tag的潜在问题，就是大量的数据可能被集中到一个实例上，导致数据倾斜，集群中的负载不均衡。那么，该怎么应对这种问题呢？我们就需要在范围查询、事务执行的需求和数据倾斜带来的访问压力之间，进行取舍了。

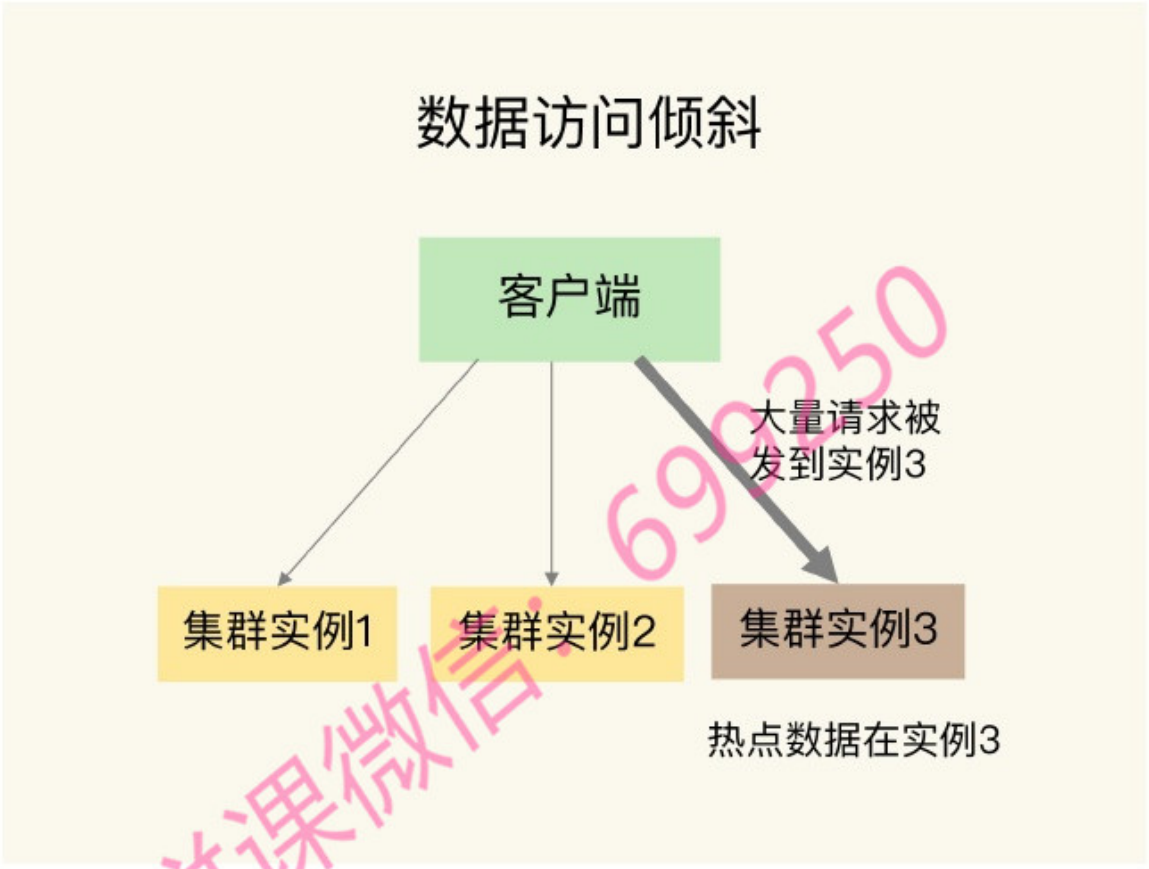
我的建议是，如果使用Hash Tag进行切片的数据会带来较大的访问压力，就优先考虑避免数据倾斜，最好不要使用Hash Tag进行数据切片。因为事务和范围查询都还可以放在客户端来执行，而数据倾斜会导致实例不稳定，造成服务不可用。

好了，到这里，我们完整地了解了数据量倾斜的原因以及应对方法。接下来，我们再来看数据访问倾斜的原因和应对方法。

数据访问倾斜的成因和应对方法

发生数据访问倾斜的根本原因，就是实例上存在热点数据（比如新闻应用中的热点新闻内容、电商促销活动中的热门商品信息，等等）。

一旦热点数据被存在了某个实例中，那么，这个实例的请求访问量就会远高于其它实例，面临巨大的访问压力，如下图所示：



那么，我们该如何应对呢？

和数据量倾斜不同，热点数据通常是一个或几个数据，所以，直接重新分配Slot并不能解决热点数据的问题。

通常来说，热点数据以服务读操作为主，在这种情况下，我们可以采用**热点数据多副本**的方法来应对。

这个方法的具体做法是，我们把热点数据复制多份，在每一个数据副本的key中增加一个随机前缀，让它和其它副本数据不会被映射到同一个Slot中。这样一来，热点数据既有多个副本可以同时服务请求，同时，这些副本数据的key又不一样，会被映射到不同的Slot中。在给这些Slot分配实例时，我们也要注意把它们分配到不同的实例上，那么，热点数据的访问压力就被分散到不同的实例上了。

这里，有个地方需要注意下，**热点数据多副本方法只能针对只读的热点数据**。如果热点数据是有读有写的话，就不适合采用多副本方法了，因为要保证多副本间的数据一致性，会带来额外的开销。

对于有读有写的热点数据，我们就要给实例本身增加资源了，例如使用配置更高的机器，来应对大量的访问压力。

小结

这节课，我向你介绍了数据倾斜的两种情况：数据量倾斜和数据访问倾斜。

造成数据量倾斜的原因主要有三个：

2. Slot手工分配不均，导致某个或某些实例上有大量数据；
3. 使用了Hash Tag，导致数据集中到某些实例上。

而数据访问倾斜的主要原因就是有热点数据存在，导致大量访问请求集中到了热点数据所在的实例上。

为了应对数据倾斜问题，我给你介绍了四个方法，也分别对应了造成数据倾斜的四个原因。我把它们总结在下表中，你可以看下。

倾斜类型	倾斜成因	应对方法
数据量倾斜	存在bigkey	业务层避免创建bigkey
		把集合类型的bigkey拆分成多个小集合，分散保存
	Slot手工分配不均	制定运维规范，避免把过多Slot分配到一个实例上
	使用Hash Tag，导致大量数据集中到一个Slot	如果Hash Tag会造成数据倾斜，优先避免数据倾斜，不使用Hash Tag
数据访问倾斜	存在热点数据	采用带有不同key前缀的多副本方法

当然，如果已经发生了数据倾斜，我们可以通过数据迁移来缓解数据倾斜的影响。Redis Cluster和Codis集群都提供了查看Slot分配和手工迁移Slot的命令，你可以把它们应用起来。

最后，关于集群的实例资源配置，我再给你一个小建议：在构建切片集群时，尽量使用大小配置相同的实例（例如实例内存配置保持相同），这样可以避免因实例资源不均衡而在不同实例上分配不同数量的Slot。

每课一问

按照惯例，我给你提个小问题，在有数据访问倾斜时，如果热点数据突然过期了，而Redis中的数据是缓存，数据的最终值保存在后端数据库，此时会发生什么问题？

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你分享给你的朋友或同事。我们下节课见。

精选留言：

- Kaito 2020-11-16 00:06:53
  在有数据访问倾斜时，如果热点数据突然过期了，而 Redis 中的数据是缓存，数据的最终值保存在后端数据库，此时会发生什么问题？

  此时会发生缓存击穿，热点请求会直接打到后端数据库上，数据库的压力剧增，可能会压垮数据库。

  Redis 的很多性能问题，例如导致 Redis 阻塞的场景：bigkey、集中过期、大实例 RDB 等等，这些场景都与数据倾斜类似，都是因为数据集中、处理逻辑集中导致的耗时变长。其解决思路也类似，都是把集中变分散，例如 bigkey 拆分为小 key、单个大实例拆分为切片集群等。

从软件架构演进过程来看，从单机到分布式，再到后来出现的消息队列、负载均衡等技术，也都是为了将请求压力分散开，避免数据集中、请求集中的问题，这样既可以让系统承载更大的请求量，同时还保证了系统的稳定性。 [17赞]

- nxcat 2020-11-16 00:22:16
  终于追上了，期待课代表的留言！课后问题我理解的话，只读模式下会发生缓存击穿，严重的话还可能造成雪崩。 [3赞]

- Lemon 2020-11-17 10:11:20
  课后题：将发生缓存击穿，导致数据库压力激增，可能导致数据库奔溃。与之相对的解决方法是不设置热点 Key 的过期时间，并以采用热点数据多副本的方法减少单实例压力。

  疑问：老师您好，热点数据多副本的方法使得每一个数据副本的 key 都有一个随机前缀，那么客户端在读取的时候怎么获取这个随机前缀？又怎么保证带上随机前缀后的热点 Key 会被较为均匀的请求呢？

- Geek_9a0c9f 2020-11-16 14:35:58
  缓存击穿，压力打到mysql.可能瞬间打爆mysql

- 花轮君 2020-11-16 09:47:48
  最终的结果是雪崩

- test 2020-11-16 08:43:04
  课后题：会发生缓存击穿