# 数据库Sharding的基本思想和切分

一、基本思想

Sharding的基本思想就要把一个数据库切分成多个部分放到不同的数据库(server)上，从而缓解单一数据库的性能问题。不太严格的讲，对于海量数据的数据库，如果是因为表多而数据多，这时候适合使用垂直切分，即把关系紧密（比如同一模块）的表切分出来放在一个server上。如果表并不多，但每张表的数据非常多，这时候适合水平切分，即把表的数据按某种规则（比如按ID散列）切分到多个数据库(server)上。当然，现实中更多是这两种情况混杂在一起，这时候需要根据实际情况做出选择，也可能会综合使用垂直与水平切分，从而将原有数据库切分成类似矩阵一样可以无限扩充的数据库(server)阵列。下面分别详细地介绍一下垂直切分和水平切分.

垂直切分的最大特点就是规则简单，实施也更为方便，尤其适合各业务之间的耦合度非

常低，相互影响很小，业务逻辑非常清晰的系统。在这种系统中，可以很容易做到将不同业

务模块所使用的表分拆到不同的数据库中。根据不同的表来进行拆分，对应用程序的影响也

更小，拆分规则也会比较简单清晰。（这也就是所谓的”share noting”）

水平切分于垂直切分相比，相对来说稍微复杂一些。因为要将同一个表中的不同数据拆

分到不同的数据库中，对于应用程序来说，拆分规则本身就较根据表名来拆分更为复杂，后

期的数据维护也会更为复杂一些。

让我们从普遍的情况来考虑数据的切分：一方面，一个库的所有表通常不可能由某一张表全部串联起来，这句话暗含的意思是，水平切分几乎都是针对一小搓一小搓（实际上就是垂直切分出来的块）关系紧密的表进行的，而不可能是针对所有表进行的。另一方面，一些负载非常高的系统，即使仅仅只是单个表都无法通过单台数据库主机来承担其负载，这意味着单单是垂直切分也不能完全解决问明。因此多数系统会将垂直切分和水平切分联合使用，先对系统做垂直切分，再针对每一小搓表的情况选择性地做水平切分。从而将整个数据库切分成一个分布式矩阵。

二、切分策略

如前面所提到的，切分是按先垂直切分再水平切分的步骤进行的。垂直切分的结果正好为水平切分做好了铺垫。垂直切分的思路就是分析表间的聚合关系，把关系紧密的表放在一起。多数情况下可能是同一个模块，或者是同一“聚集”。这里的“聚集”正是领域驱动设计里所说的聚集。在垂直切分出的表聚集内，找出“根元素”（这里的“根元素”就是领域驱动设计里的“聚合根”），按“根元素”进行水平切分，也就是从“根元素”开始，把所有和它直接与间接关联的数据放入一个shard里。这样出现跨shard关联的可能性就非常的小。应用程序就不必打断既有的表间关联。比如：对于社交网站，几乎所有数据最终都会关联到某个用户上，基于用户进行切分就是最好的选择。再比如论坛系统，用户和论坛两个模块应该在垂直切分时被分在了两个shard里，对于论坛模块来说，Forum显然是聚合根，因此按Forum进行水平切分，把Forum里所有的帖子和回帖都随Forum放在一个shard里是很自然的。

对于共享数据数据，如果是只读的字典表，每个shard里维护一份应该是一个不错的选择，这样不必打断关联关系。如果是一般数据间的跨节点的关联，就必须打断。

三、切分的常见问题和应对策略

1.事务问题：

解决事务问题目前有两种可行的方案：分布式事务和通过应用程序与数据库共同控制实现事务下面对两套方案进行一个简单的对比。

方案一：使用分布式事务

优点：交由数据库管理，简单有效

缺点：性能代价高，特别是shard越来越多时

方案二：由应用程序和数据库共同控制

原理：将一个跨多个数据库的分布式事务分拆成多个仅处

于单个数据库上面的小事务，并通过应用程序来总控

各个小事务。

优点：性能上有优势

缺点：需要应用程序在事务控制上做灵活设计。如果使用

了spring的事务管理，改动起来会面临一定的困难。

2.跨节点Join的问题

只要是时行切分，跨节点Join的问明是不可避免的。但是良好的设计和切分却可以减少此类情况的发生。解决这一问题的普遍做法是分两次查询实现。在第一次查询的结果集中找出关联数据的id,根据这些id发起第二次请求得到关联数据。

3.跨节点的count,order by,group by以及聚合函数问题

这些是一类问题，因为它们都需要基于全部数据集合进行计算。多数的代理都不会自动处理合并工作。解决方案：与解决跨节点join问题的类似，分别在各个节点上得到结果后在应用程序端进行合并。和join不同的是每个结点的查询可以并行执行，因此很多时候它的速度要比单一大表快很多。但如果结果集很大，对应用程序内存的消耗是一个问题。