二

36 什么时候需要分表分库?

你好，我是刘超。

在当今互联网时代，海量数据基本上是每一个成熟产品的共性，特别是在移动互联网产品中，几乎每天都在产生数据，例如，商城的订单表、支付系统的交易明细以及游戏中的战报等等。

对于一个日活用户在百万数量级的商城来说，每天产生的订单数量可能在百万级，特别在一些活动促销期间，甚至上千万。

假设我们基于单表来实现，每天产生上百万的数据量，不到一个月的时间就要承受上亿的数据，这时单表的性能将会严重下降。因为 MySQL 在 InnoDB 存储引擎下创建的索引都是基于 B+ 树实现的，所以查询时的 I/O 次数很大程度取决于树的高度，随着 B+ 树的树高增高，I/O 次数增加，查询性能也就越差。

当我们面对一张海量数据的表时，通常有分区、NoSQL 存储、分表分库等优化方案。

分区的底层虽然也是基于分表的原理实现的，即有多个底层表实现，但分区依然是在单库下进行的，在一些需要提高并发的场景中的优化空间非常有限，且一个表最多只能支持 1024 个分区。面对日益增长的海量数据，优化存储能力有限。不过在一些非海量数据的大表中，我们可以考虑使用分区来优化表性能。

> 分区表是由多个相关的底层表实现的，这些底层表也是由句柄对象表示，所以我们也可以直接访问各个分区，存储引擎管理分区的各个底层表和管理普通表一样（所有的底层表都必须使用相同的存储引擎），分区表的索引只是在各个底层表上各自加上一个相同的索引，从存储引擎的角度来看，底层表和一个普通表没有任何不同，存储引擎也无须知道这是一个普通表，还是一个分区表的一部分。

而 NoSQL 存储是基于键值对存储，虽然查询性能非常高，但在一些方面仍然存在短板。例如，不是关系型数据库，不支持事务以及稳定性方面相对 RDBMS 差一些。虽然有些 NoSQL 数据库也实现了事务，宣传具有可靠的稳定性，但目前 NoSQL 还是主要用作辅助存储。

什么时候要分表分库?

分析完了分区、NoSQL 存储优化的应用，接下来我们就看看这讲的重头戏——分表分库。

在我看来，能不分表分库就不要分表分库。在单表的情况下，当业务正常时，我们使用单表即可，而当业务出现了性能瓶颈时，我们首先考虑用分区的方式来优化，如果分区优化之后仍然存在后遗症，此时我们再来考虑分表分库。

我们知道，如果在单表单库的情况下，当数据库表的数据量逐渐累积到一定的数量时（5000W 行或 100G 以上），操作数据库的性能会出现明显下降，即使我们使用索引优化或读写库分离，性能依然存在瓶颈。此时，如果每日数据增长量非常大，我们就应该考虑分表，避免单表数据量过大，造成数据库操作性能下降。

面对海量数据，除了单表的性能比较差以外，我们在单表单库的情况下，数据库连接数、磁盘 I/O 以及网络吞吐等资源都是有限的，并发能力也是有限的。所以，在一些大数据量且高并发的业务场景中，我们就需要考虑分表分库来提升数据库的并发处理能力，从而提升应用的整体性能。

如何分表分库?

通常，分表分库分为垂直切分和水平切分两种。

垂直分库是指根据业务来分库，不同的业务使用不同的数据库。例如，订单和消费券在抢购业务中都存在着高并发，如果同时使用一个库，会占用一定的连接数，所以我们可以将数据库分为订单库和促销活动库。

而垂直分表则是指根据一张表中的字段，将一张表划分为两张表，其规则就是将一些不经常使用的字段拆分到另一张表中。例如，一张订单详情表有一百多个字段，显然这张表的字段太多了，一方面不方便我们开发维护，另一方面还可能引起跨页问题。这时我们就可以拆分该表字段，解决上述两个问题。

水平分表则是将表中的某一列作为切分的条件，按照某种规则（Range 或 Hash 取模）来切分为更小的表。

水平分表只是在一个库中，如果存在连接数、I/O 读写以及网络吞吐等瓶颈，我们就需要考虑将水平切换的表分布到不同机器的库中，这就是水平分库分表了。

结合以上垂直切分和水平切分，我们一般可以将数据库分为：单库单表 - 单库多表 - 多库多表。在平时的业务开发中，我们应该优先考虑单库单表；如果数据量比较大，且热点数据比较集中、历史数据很少访问，我们可以考虑表分区；如果访问热点数据分散，基本上所有的

数据都会访问到，我们可以考虑单库多表；如果并发量比较高、海量数据以及每日新增数据量巨大，我们可以考虑多库多表。

这里还需要注意一点，我刚刚强调过，能不分表分库，就不要分表分库。这是因为一旦分表，我们可能会涉及到多表的分页查询、多表的 JOIN 查询，从而增加业务的复杂度。而一旦分库了，除了跨库分页查询、跨库 JOIN 查询，还会存在跨库事务的问题。这些问题无疑会增加我们系统开发的复杂度。

分表分库之后面临的问题

然而，分表分库虽然存在着各种各样的问题，但在一些海量数据、高并发的业务中，分表分库仍是最常用的优化手段。所以，我们应该充分考虑分表分库操作后所面临的一些问题，接下我们就一起看看都有哪些应对之策。

为了更容易理解这些问题，我们将对一个订单表进行分库分表，通过详细的业务来分析这些问题。

假设我们有一张订单表以及一张订单详情表，每天的数据增长量在 60W 单，平时还会有一些促销类活动，订单增长量在千万单。为了提高系统的并发能力，我们考虑将订单表和订单详情表做分库分表。除了分表，因为用户一般查询的是最近的订单信息，所以热点数据比较集中，我们还可以考虑用表分区来优化单表查询。

通常订单的分库分表要么基于订单号 Hash 取模实现，要么根据用户 ID Hash 取模实现。订单号 Hash 取模的好处是数据能均匀分布到各个表中，而缺陷则是一个用户查询所有订单时，需要去多个表中查询。

由于订单表用户查询比较多，此时我们应该考虑使用用户 ID 字段做 Hash 取模，对订单表进行水平分表。如果需要考虑高并发时的订单处理能力，我们可以考虑基于用户 ID 字段 Hash 取模实现分库分表。这也是大部分公司对订单表分库分表的处理方式。

1. 分布式事务问题

在提交订单时，除了创建订单之外，我们还需要扣除相应的库存。而订单表和库存表由于垂直分库，位于不同的库中，这时我们需要通过分布式事务来保证提交订单时的事务完整性。

通常，我们解决分布式事务有两种通用的方式：两阶事务提交（2PC）以及补偿事务提交（TCC）。有关分布式事务的内容，我将在第 41 讲中详细介绍。

通常有一些中间件已经帮我们封装好了这两种方式的实现，例如 Spring 实现的 JTA，目前阿里开源的分布式事务中间件 Fescar，就很好地实现了与 Dubbo 的兼容。

2. 跨节点 JOIN 查询问题

用户在查询订单时，我们往往需要通过表连接获取到商品信息，而商品信息表可能在另外一个库中，这就涉及到了跨库 JOIN 查询。

通常，我们会冗余表或冗余字段来优化跨库 JOIN 查询。对于一些基础表，例如商品信息表，我们可以在每一个订单分库中复制一张基础表，避免跨库 JOIN 查询。而对于一两个字段的查询，我们也可以将少量字段冗余在表中，从而避免 JOIN 查询，也就避免了跨库 JOIN 查询。

3. 跨节点分页查询问题

我们知道，当用户在订单列表中查询所有订单时，可以通过用户 ID 的 Hash 值来快速查询到订单信息，而运营人员在后台对订单表进行查询时，则是通过订单付款时间来进行查询的，这些数据都分布在不同的库以及表中，此时就存在一个跨节点分页查询的问题了。

通常一些中间件是通过在每个表中先查询出一定的数据，然后在缓存中排序后，获取到对应的分页数据。这种方式在越往后面的查询，就越消耗性能。

通常我们建议使用两套数据来解决跨节点分页查询问题，一套是基于分库分表的用户单条或多条查询数据，一套则是基于 Elasticsearch、Solr 存储的订单数据，主要用于运营人员根据其它字段进行分页查询。为了不影响提交订单的业务性能，我们一般使用异步消息来实现 Elasticsearch、Solr 订单数据的新增和修改。

4. 全局主键 ID 问题

在分库分表后，主键将无法使用自增长来实现了，在不同的表中我们需要统一全局主键 ID。因此，我们需要单独设计全局主键，避免不同表和库中的主键重复问题。

使用 UUID 实现全局 ID 是最方便快捷的方式，即随机生成一个 32 位 16 进制数字，这种方式可以保证一个 UUID 的唯一性，水平扩展能力以及性能都比较高。但使用 UUID 最大的缺陷就是，它是一个比较长的字符串，连续性差，如果作为主键使用，性能相对来说会比较差。

我们也可以基于 Redis 分布式锁实现一个递增的主键 ID，这种方式可以保证主键是一个整数且有一定的连续性，但分布式锁存在一定的性能消耗。

我们还可以基于 Twitter 开源的分布式 ID 生产算法——snowflake 解决全局主键 ID 问题，snowflake 是通过分别截取时间、机器标识、顺序计数的位数组成一个 long 类型的主键 ID。这种算法可以满足每秒上万个全局 ID 生成，不仅性能好，而且低延时。

5. 扩容问题

随着用户的订单量增加，根据用户 ID Hash 取模的分表中，数据量也在逐渐累积。此时，我们需要考虑动态增加表，一旦动态增加表了，就会涉及到数据迁移问题。

我们在最开始设计表数据量时，尽量使用 2 的倍数来设置表数量。当我们需要扩容时，也同样按照 2 的倍数来扩容，这种方式可以减少数据的迁移量。

总结

在业务开发之前，我们首先要根据自己的业务需求来设计表。考虑到一开始的业务发展比较平缓，且开发周期比较短，因此在开发时间比较紧的情况下，我们尽量不要考虑分表分库。但是我们可以将分表分库的业务接口预留，提前考虑后期分表分库的切分规则，把该冗余的字段提前冗余出来，避免后期分表分库的 JOIN 查询等。

当业务发展比较迅速的时候，我们就要评估分表分库的必要性了。一旦需要分表分库，就要结合业务提前规划切分规则，尽量避免消耗性能的跨表跨库 JOIN 查询、分页查询以及跨库事务等操作。

思考题

你使用过哪些分库分表中间件呢？欢迎分享其中的实现原理以及优缺点。

上一页 下一页