10亿级别订单的分库分表方案

背景

随着公司业务增长，如果每天1000多万笔订单的话，3个月将有约10亿的订单量，之前数据库采用单库单表的形式已经不满足于业务需求，数据库改造迫在眉睫。

订单数据如何划分?

我们可以将订单数据划分成两大类型：分别是热数据和冷数据。

热数据：3个月内的订单数据，查询实时性较高。

冷数据A：3个月 ~ 12个月前的订单数据，查询频率不高。

冷数据B：1年前的订单数据，几乎不会查询，只有偶尔的查询需求。

可能这里有个疑惑为什么要将冷数据分成两类，因为根据实际场景需求，用户基本不会去查看1年前的数据，如果将这部分数据还存储在db中，那么成本会非常高，而且也不便于维护。另外如果真遇到有个别用户需要查看1年前的订单信息，可以让用户走离线数据查看。

对于这三类数据的存储，目前规划如下：

热数据： 使用mysql进行存储，当然需要分库分表

冷数据A: 对于这类数据可以存储在ES中，利用搜索引擎的特性基本上也可以做到比较快的查询。

冷数据B: 对于这类不经常查询的数据，可以存放到hive中

MySql 如何分库分表

一、按业务拆分

在业务初始阶段，为了加快应用上线和快速迭代，很多应用都采用集中式的架构。但是随着业务系统的扩大，系统匾额越来越复杂，越来越难以维护，开发效率变得越来越低，并且对资源的消耗也变得越来越大，通过硬件提高系统性能的成本会变得更高。

通常一般的电商平台，包含了用户、商品、订单等几大模块，简单的做法是在同一个库中分别建4张表。

但是随着业务的提升，将所有业务都放在一个库中已经变得越来越难以维护，因此我们建议，将不同业务放在不同的库中。

我们将不同的业务放到不同的库中，将原来所有压力由同一个库中分散到不同的库中，提升了系统的吞吐量。

二、分库与分表

我们知道每台机器无论配置多么好它都有自身的物理上限，所以当我们应用已经能触及或远远超出单台机器的某个上限的时候，我们惟有寻找别的机器的帮助或者继续升级的我们的硬件，但常见的方案还是通过添加更多的机器来共同承担压力。

我们还得考虑当我们的业务逻辑不断增长，我们的机器能不能通过线性增长就能满足需求？因此，使用数据库的分库分表，能够立竿见影的提升系统的性能，关于为什么要使用数据库的分库分表的其他原因这里不再赘述，主要讲具体的实现策略。

1)、分表策略

我们以订单表为例，在订单表中，订单id肯定是不可重复的，因此将该字段当做shard key 是非常适合的,其他表类似。假设订单表的字段如下：

```
create table order(
    order_idbigint(11) ,
    user_idbigint(11),
    phonevarchar(15),
    ...
)
```

我们假设预估单个库需要分配100个表满足我们的业务需求，我们可以简单的取模计算出订单在哪个子表中，例如： order_id % 100。

这时候可能会有人问了，如果我根据order_id 进行分表规则，但是我想根据user_id 查询相应的订单，不是定位不到哪个子表了吗，的确是这样，一旦确定shard key，就只能根据shard key定位到子表进而查询该子表下的数据；如果确实想根据user_id 去查询相关订单，那应该将shard key设置为user_id，那分表规则也相应的变更为： user_id % 100;

2)、分库实现策略

数据库分表能够解决单表数据量很大的时候数据查询的效率问题，但是无法给数据库的并发操作带来效率上的提高，因为分表的实质还是在一个数据库上进行的操作，很容易受数据库IO性能的限制。

因此，如何将数据库IO性能的问题平均分配出来，很显然将数据进行分库操作可以很好地解决单台数据库的性能问题。

分库策略与分表策略的实现很相似，最简单的都是可以通过取模的方式进行路由。

我们还是以order表举例，

例如：order_id % 库容量,

如果order_id 不是整数类型，可以先hash 在进行取模，

例如： hash(order_id) % 库容量

3)、分库分表结合使用策略

数据库分表可以解决单表海量数据的查询性能问题，分库可以解决单台数据库的并发访问压力问题。有时候，我们需要同时考虑这两个问题，因此，我们既需要对单表进行分表操作，还需要进行分库操作，以便同时扩展系统的并发处理能力和提升单表的查询性能，就是我们使用到的分库分表。

如果使用分库分表结合使用的话，不能简单进行order_id 取模操作，需要加一个中间变量用来打散到不同的子表，公式如下：

1中间变量 = shard key %（库数量*单个库的表数量）；

2库序号 = 取整（中间变量 / 单个库的表数量）；

3表序号 = 中间变量%单个库的表数量;

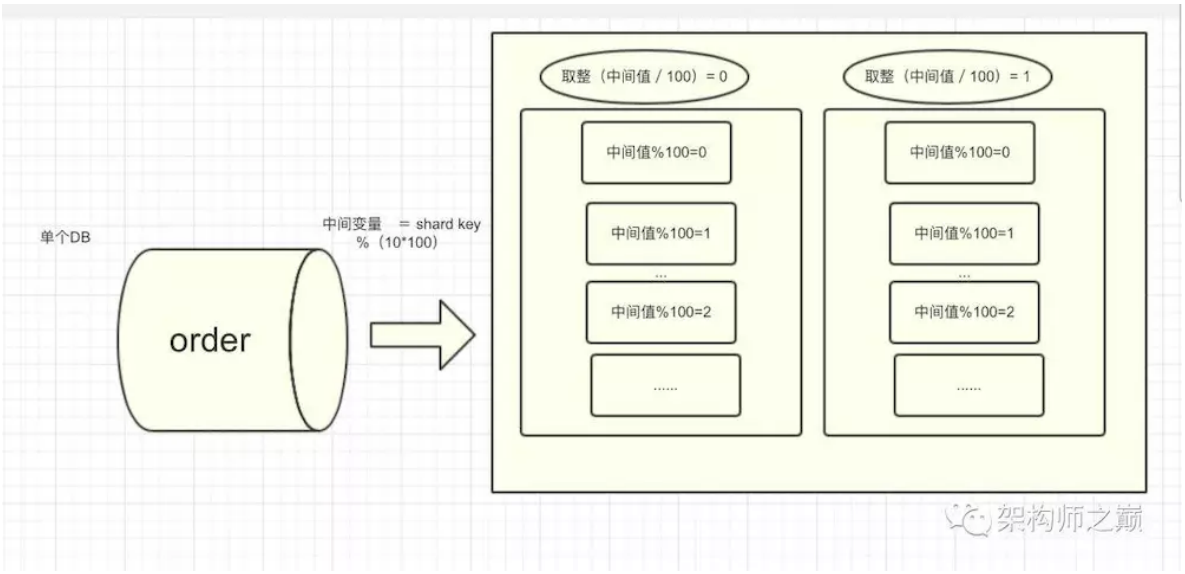
例如：数据库有10个，每一个库中有100个数据表，用户的order_id = 1001，按照上述的路由策略，可得：

1中间变量 = 1001 %（10*100）= 1;

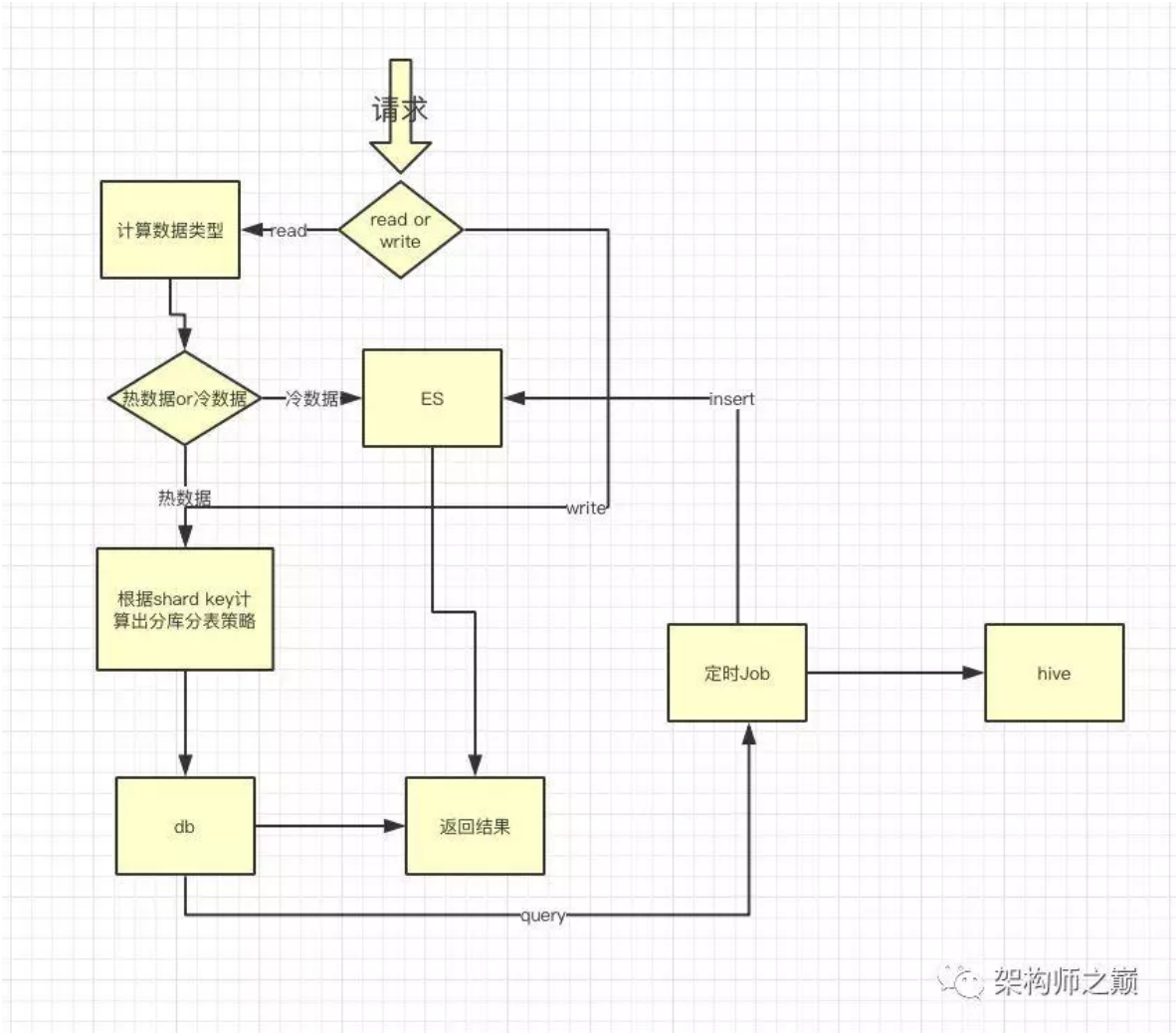
2库序号 = 取整（1 / 100）= 0;

3表序号 = 1%100=1;

这样的话，对于order_id = 1001，将被路由到第1个数据库的第2个表中(索引0 代表1，依次类推)。



整体架构设计



从图中我们将请求分成read和write请求，write请求比较简单，就是根据分库分表规则写入db即可。

对于read请求，我们需要计算出查询的是热数据还是冷数据，一般order_id生成规则如下，“商户所在地区号+时间戳+随机数”，我们可以根据时间戳计算出查询的是热数据还是冷数据，（当然具体业务需要具体对待，这里不再详细阐述）

另外架构图中的冷数据指的是3个月~12个月前的数据，如果是查询一年前的数据，建议直接离线查hive即可。

图中有一个定时Job，主要用来定时迁移订单数据，需要将冷数据分别迁移到ES和hive中。