关于分库分表策略的分析和总结

1994_老叶 关注

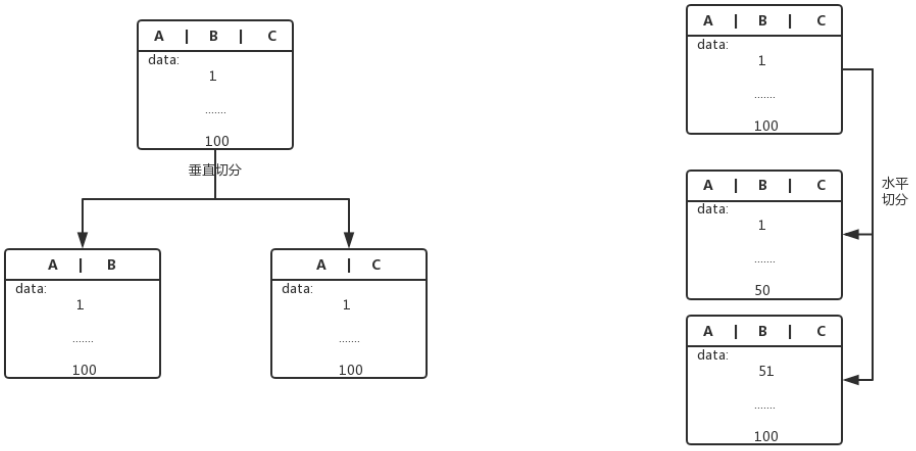
0.728 2018.04.28 14:46:47 字数 1,998 阅读 4,530

一.分库分表的原因

我个人觉得原因其实很简单：
1.随着单库中的数据量越来越大，相应的，查询所需要的时间也越来越多，而面对MySQL这样的数据库，在进行添加一列这样的操作时会有锁表的操作，期间所有的读写操作都要等待，这个时候，相当于数据的处理遇到了瓶颈
2.其实就是有意外发生的时候，单库发生意外的时候，需要修复的是所有的数据，而多库中的一个库发生意外的时候，只需要修复一个库（当然，也可以用物理分区的方式处理这种问题）

二.分库分表常用的策略

在我网上搜集的过程中，以及自己的实践，得到的分库策略可以简单分为以下几种方式：（如果有不正确的地方，请大家给我指出来，万分感谢）
首先，分为垂直切分和水平切分：
先说垂直切分吧，我的认为是根据业务的不同，将原先拥有很多字段的表拆分为两个或者多个表，这样的代价我个人觉得很大，原来对这应这个表的关系，开始细分，需要一定的重构，而且随着数据量的增多，极有可能还要增加水平切分；
水平切分，数据表结构，将数据分散在多个表中；



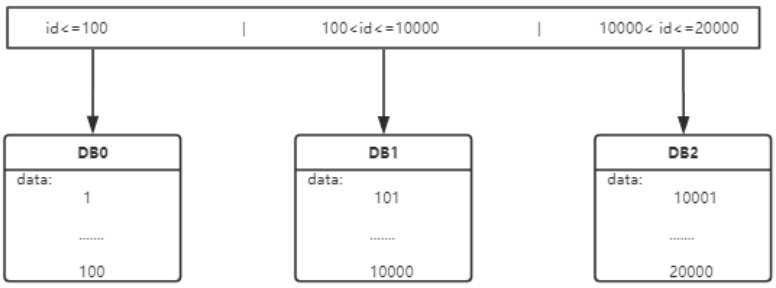
简单的示意图的了解一下。
对于垂直切分，好像能说的并不多，说的比较多一点的是水平切分。
水平切分时候，理想的情况是不进行数据迁移，无感知的进行，当然这就需要一点点小小的分库分表的策略。

1.有瑕疵的简单分库分表（按id的大小分库分表）

我觉得读书挺好的
阅读 49

读spring源码记录（七）--invokeBeanFactoryPostProcessors
阅读 61

按照分片键（我们这里就用id表示了）的大小来进行分库分表，如果你的id是自增的，而且能保证在进行分库分表后也是自增的，那么能进行很好的改造，以id大小水平切分，而且极有可能不用迁移数据。

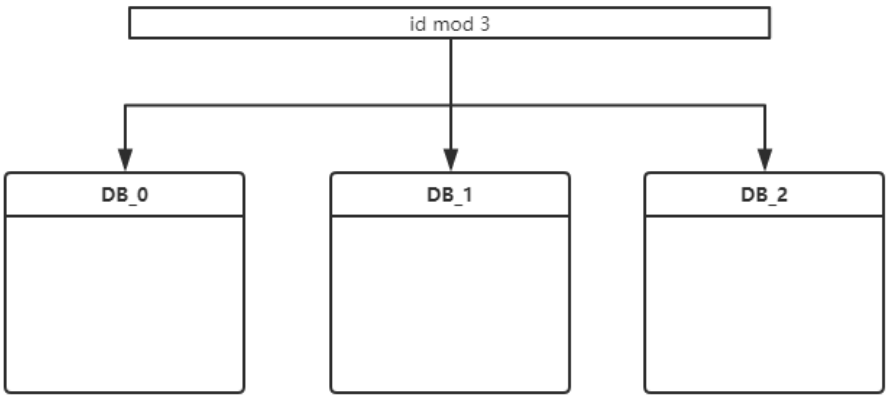


按id大小分库.PNG

当然，这里只列举了比较小的数据量，实际情况的分库的界限还是要依据具体的情况而定。这样的分库分表，因为新的数据总在一个库里，很可能导致热点过于集中（读写可能集中在一个库中），这是采取这种方式需要考虑的事情。
如果无法保证你的id是自增长的，那么你的数据就会凌乱的分散在各个数据库，这样热点确实就分散了，可是每当你增加一个数据库的时候，你就有可能进行大量的数据迁移，应对这种情况，就是尽量减少数据迁移的代价，所以这里运用 一致性hash 的方式进行分库分表比较好，可以尽可能的减少数据迁移，并且也能让解决热点过于集中的问题。一致性hash的分库策略去百度一下或者谷歌一下应该很容易搜到。如果你百度了还是不知道，欢迎你来跟我讨论。
这里按id的大小来分库，还可以发散到按照时间来分库，比如说一个月的数据放在一个库，这个使用mycat比较容易实现按时间分库，不过你需要思考的数据的离散性，数据集中于一个两月，而剩下的几个月数据稀疏，这样的又可能需要按照数据的生产规律合并几个月到一个库中，使得数据分布均匀。

2.比较方便的取模分库

一般的 取模分库分表 是就是将id mod n，然后放入数据库中，这样能够使数据分散，不会有热点的问题，那么，剩下的是，在扩容的时候，是否会有数据迁移的问题，一般的扩容，当然是会有数据迁移的。

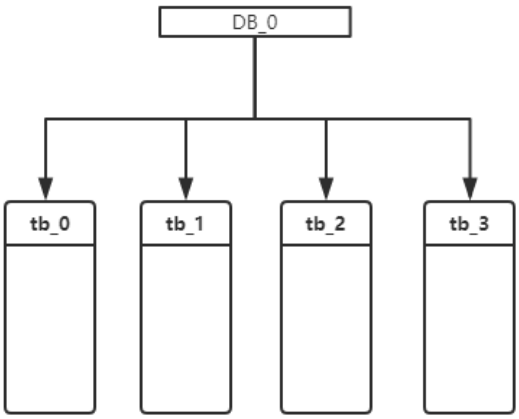


取模.PNG

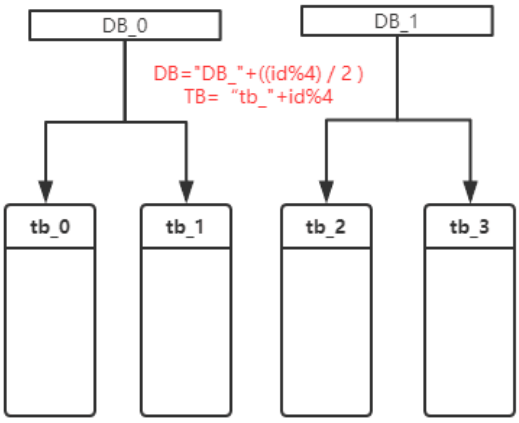
例子中，对3取模，当需要扩容的时候(假设增加两个库)，则对5取模，这样的结果必然是要进行数据迁移的，但是可以运用一些方法，让它不进行数据迁移， scale-out扩展方案 能够避免在取模扩容的时候进行数据迁移。这个方案是我看到的，我个人觉得很好的方案了，这是*原文*。
我也想介绍一下这个方案（主要想检测一下自己理解了没）：

(1)第一种扩容的方式：根据表的数据增加库的数量

首先，我们有一个数据库——DB_0,四张表——tb_0,tb_1,tb_2,tb_3
那么我们现在数据到数据库是这样的:
DB="DB_0"
TB=“tb_"+id%4

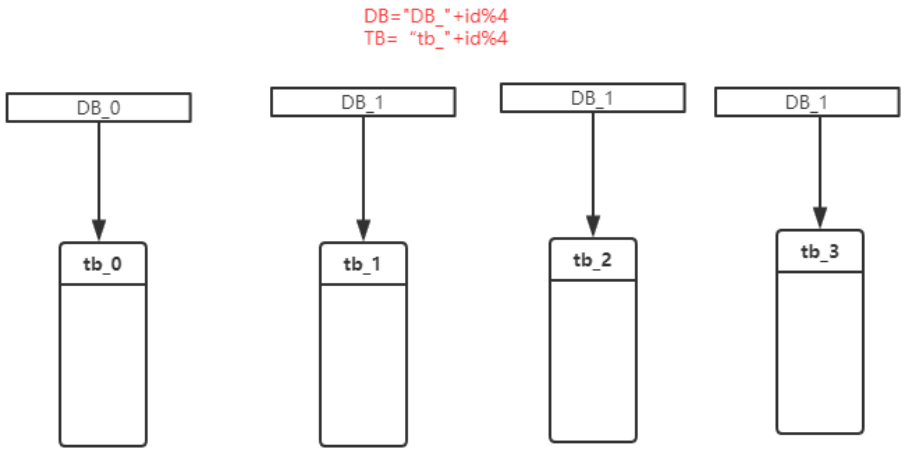


当数据增加，需要进行扩容的时候，我增加一个数据库DB_1
DB="DB_"+((id%4)/2)
TB=“tb_"+id%4



当我们的数据继续飙升，这个时候又需要我们增加库，这个时候进行加库操作的时候，就不是增加一个库，而必须是两个，这样才能保证不进行数据迁移。

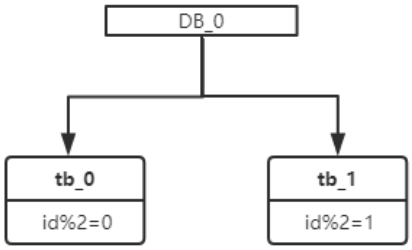
DB="DB_"+id%4
TB="tb_"+id%4



这个时候到了这个方案的加库上限，不能继续加库了，否则就要进行数据迁移，所以这个方案的弊端还是挺大了，这样的方式，也应该会造成单表的数据量挺大的。

(2)第二种扩容的方式：成倍的增加表

首先，我们还是一个数据库——DB_0,两张表——tb_0,tb_1
那么我们现在数据到数据库是这样的:
DB="DB_0"
TB="tb_"+id%2



假设当我们数据量打到一千万的时候，我们增加一个库，这时需要我们增加两张表tb_0_1,tb_1_1,并且原来的DB_0中库的表tb_1整表迁移到DB_1中，tb_0和tb_0_1放在DB_0中，tb_1和tb_1_1放到DB1中。
DB="DB_"+id%2
tb:
if(id<1千万) { return "tb_" + id % 2 }
else if(id>=1千万) { return "tb_"+ id % 2 + "_1" }

数据的增长不可能到此为止，当增加到两千万的时候，我们需要加库，这个时候，按照这种做法，我们需要增加两个库(DB_2,DB_3)和四张表(tb_0_2,tb_1_2,tb_2_2,tb_3_2)，将上次新增的表整表分别放进两个新的库中，然后每个库里再生成一张新表。

DB:

if(id < 1千万) { return "DB_" + id % 2 }

else if(1千万 <= id < 2千万) { return "DB_"+ id % 2 +2 }

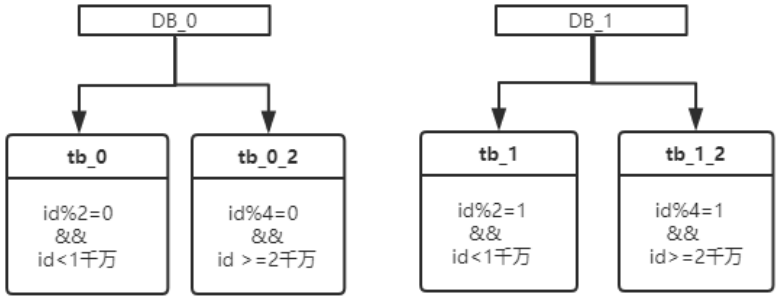
else if(2千万 <= id) { return "DB_"+ id % 4 }

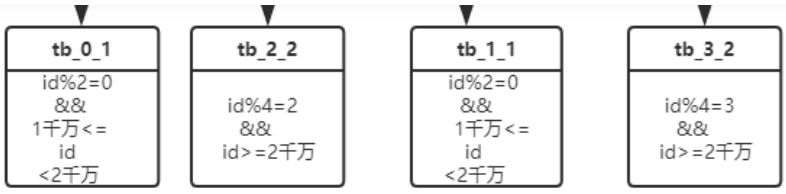
tb:

if(id < 1千万) { return "tb_" + id % 2 }

else if(1千万 <= id < 2千万) { return "tb_"+ id % 2 +"*1" }*

else if(id >= 2千万) { return "tb"+ id % 4+"_2" }





值得注意的一点，id超出范围的时候，该给怎么样的提示是值得思考的。

15赞

赏

赞赏

(3)第二种扩容的方式：一个一个的增加。(我在这里和原文有点出入，大家不看也罢)

上一种方式是成倍的增加，有的时候往往不需要这样，现在我们基于上一个例子的第二阶段(1千万到2千万的阶段)，添加一个库DB_2,新增两张表tb_0_2,tb_1_2;将tb_0和tb_1放在DB_0中，最为旧文件的查询，tb_0_1和和tb_1_1分别放入DB_1和DB_2中，再在这两个库中生成新的表

DB:

if(id < 1千万) { return "DB_0"}

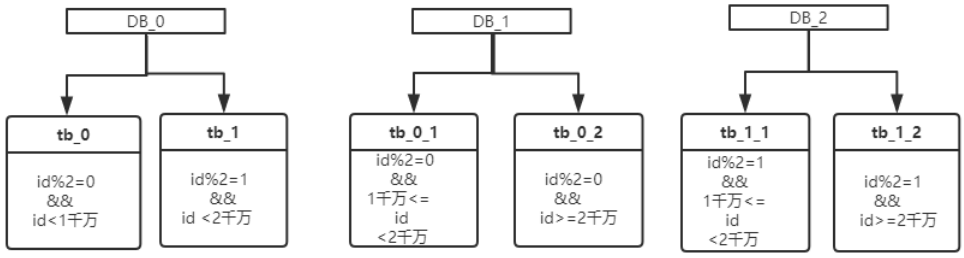
else if(1千万 <= id < 2千万) { return "DB_"+ (id % 2 + 1)

else if(id >= 2千万) {return "DB_"+ id%3}

tb:

if(id < 1千万) { return "tb_" + id%2}

else if(1千万 <= id < 2千万) { return "tb_"+ (id % 2) +"1"
else if(id >= 2千万) {return "DB"+ id%2 +"_0"}



第三种扩展方式，按照原文的介绍，会在旧的数据库中加入新的数据库，而且当继续扩容的时候，也会又一定的困难，我这样的方式，对于新的扩容，比较困难，所以第三种方式总的来说是我认为是失败，我个人觉得最优的方式是第二种，我这里的id>n是指在数据量达到n这个数据量，而不是指id按大小进行比较，那样的话，和按照id大小进行扩容又什么区别，哈哈哈。
总得来说，对于数据库扩容，总得思考方向为两点：一个是是否进行数据迁移；一个是数据是否分布均匀，会不会造成热点集中的情况。数据迁移也不一定是坏的，这些都依据场景而定。

二.分库分表后的考虑

分库分表之后常常会遇到数据分页的问题，这个问题其实解决的办法很多，但是都没有一个完美的方法，总的来说，还是需要妥协，例如在不分库分表前：select * from t_msg order by time offset 200 limit 100 这样的语句，在分库分表的后，我看到的有这样几种处理

写下你的评论... 评论0 赞15 ···

服务层对得到的N*(X+Y)条数据进行内存排序，内存排序后再取偏移量X后的Y条记录。
方法二：业务折衷法-禁止跳页查询
用正常的方法取得第一页数据，并得到第一页记录的time_max。
每次翻页，将order by time offset X limit Y，改写成order by time where time>$time_max limit Y以保证每次只返回一页数据，性能为常量。
方法三：业务折衷法-允许模糊数据（数据分布足够随机的情况下，各分库所有非patition key属性，在各个分库上的数据分布，统计概率情况应该是一致的）
将order by time offset X limit Y，改写成order by time offset X/N limit Y/N。
方法四：二次查询法
将order by time offset X limit Y，改写成order by time offset X/N limit Y；
找到最小值time_min；
between二次查询，order by time between $$time_min and $time_i_max；
设置虚拟time_min，找到time_min在各个分库的offset，从而得到time_min在全局的offset；
得到了time_min在全局的offset，自然得到了全局的offset X limit Y。

最后我想说，不同的业务场景对应不同的策略，不能为了追求最新的东西，而忽略真正的业务场景，这样的得不偿失。任何一件事都具有两面性，就看你如何取舍，放之四海而皆准。不管什么事情都能解决的，所以，遇到问题不要慌。

15人点赞 > 日记本 ···

"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下

1994_老叶 肩挑清风与日月，在生活中负重前行
总资产7 (约0.67元) 共写了2.0W字 获得140个赞 共58个粉丝

关注

写下你的评论...

全部评论 0 只看作者

按时间倒序 按时间正序

被以下专题收入，发现更多相似内容

 我爱编程  程序员

推荐阅读

更多精彩内容 >

Java面试宝典Beta5.0

pdf下载地址：Java面试宝典 第一章内容介绍 20 第二章JavaSE基础 21 一、Java面向对象 21 ...

 王震阳 阅读 78,985 评论 25 赞 513

跨库分页

一、需求缘起 分页需求 互联网很多业务都有分页拉取数据的需求，例如： （1）微信消息过多时，拉取第N页消息 （2）...

 duzhongli 阅读 94 评论 0 赞 2

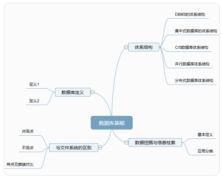
数据库水平切分的实现原理解析——分库，分表，主从，集群，负载均衡器（转）

http://www.cnblogs.com/zhongxinWang/p/4262650.html 第1章 引言...

 bin_xin 阅读 108 评论 0 赞 0

数据库的简单学习

需要原文的可以留下邮箱我给你发，这里的文章少了很多图，懒得网上粘啦 1数据库基础 1.1数据库定义 1）数据库(D...



极简纯粹_ 阅读 3,990 评论 13 赞 47

云市场陷入病态,大降价将云厂商赶往“鬼门关”

王者 14310116049 转载自：http://www.elecfans.com/tongxin/yunton...

六六六六六六66 阅读 33 评论 0 赞 0