参考： https://www.cnblogs.com/littlecharacter/p/9084291.html#m1

背景

事实上MySQL单表可以存储10亿级数据，只是这时候性能比较差，业界公认MySQL单表容量在1KW以下是最佳状态，因为这时它的BTREE索引树高在3~5之间。

既然一张表无法搞定，那么就想办法将数据放到多个地方；

目前比较普遍的提升数据库性能方案主要有：

1.加缓存和索引是通用的提升数据库性能的方式；

2.读写分离；

3.分库分表；

数据库架构原则

1. 高可用
2. 高性能
3. 可扩展
4. 一致性

读写分离

应用：

在实际应用中的绝大多数情况下读操作远大于写操作。MySQL提供了读写分离的机制，所有写操作必须对应到主库（Master），读操作可以在主库（Master）和从库（Slave）机器上进行；这种一主多从的方式可以有效地提高数据库集群的吞吐量。

缺点：

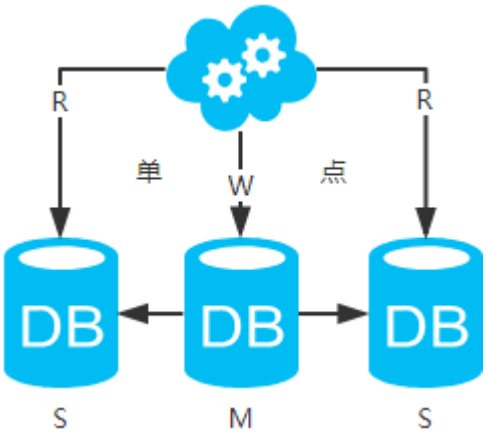
所有写操作都先在主库上进行，然后异步更新到从库上，所以从主库同步到从库机器有一定的延迟，当系统很繁忙时，延迟问题会更加严重，从库机器数量的增加也会使这个问题更严重。主库是集群的瓶颈，当写操作过多时会严重影响主库的稳定性；

实践：

- 当读操作压力很大时，可以考虑添加从库机器来分解大量读操作带来的压力，但是当从库机器达到一定的数量时，就需要考虑分库来缓解压力了。
- 当写压力很大时，就必须进行分库分表操作了。

模型： 一般配置主-主-从或者主-从-从部署模型。（主备架构主要作为容灾方案）

- **一主多从：**



1.高可用分析：主库单点，从库高可用；一旦主库挂了，写服务将无法提供；

2.高性能分析：由于大部分情况读多写少，读会先成为瓶颈，进而影响整体性能；

主库可以不用索引，从库可以加上相同索引；

3.一致性分析：包括主从不一致和缓存与数据库不一致；

主从不一致：

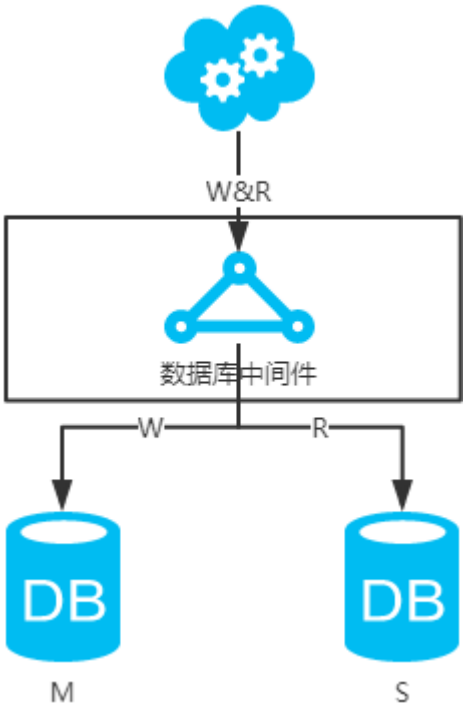
现象：

数据同步（MySQL主从复制，简单来说就是从库从主库拉取binlog日志，再执行一遍）是需 要时间的，这个同步时间内主库和从库的数据会存在不一致的情况。如果同步过程中有读请求， 那么读到的就是从库中的老数据。存在数据一致性问题；

解决：

方案1：选择读主： 写操作时根据库+表+业务特征生成一个key放到缓存里并设置超时时间（大于等于主从数据同步时间）。读请求时，同样的方式生成key先去查缓存，再判断是否命中。若命中，表示缓存数据还在，未达到超时时间，主从同步未完成，则读主库，否则读从库。代价是多了一次缓存读写，基本可以忽略。

方案2：数据库中间件，引入开源（mycat等）或自研的数据库中间层，思路同选择读主，数据库中间件的成本比较高，并且还多引入了一层。



方案3：业务允许延时可不用处理；

缓存与数据库不一致：

现象：

若先更新数据库，再删除缓存，如果删除缓存失败了，这时缓存未旧数据，数据库为新数据，出现数据不一致；

若先删除缓存，再更新数据库，未更新完成时有新请求过来，这时会将现有旧数据写入缓存，随后数据变更的程序完成了数据库的修改，出现数据不一致；

解决：

方案1：写请求先删除缓存，再去更新数据库，（异步等待一段时间)再删除缓存（删除成功表示有脏数据出现）。这样缓存旧数据的概率变低，不过数据库的压力会有相应的增加。

方案2：写请求先修改缓存为指定值，再去更新数据库，再更新缓存。读请求过来后，先读缓存，判断是指定值后进入循环状态，等待写请求更新缓存。如果循环超时就去数据库读取数据，更新缓存。这种方案保证了读写的一致性，但是读请求会等待写操作的完成，降低了吞吐量；

4.扩展性分析：

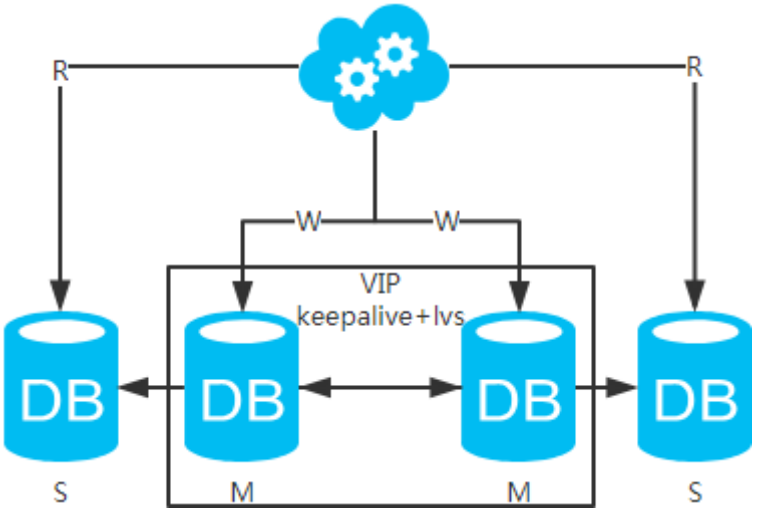
可以通过加从库来扩展读性能，进而提高整体性能。（带来的问题是，从库越多需要从主库拉取binlog日志的端就越多，进而影响主库的性能，并且数据同步完成的时间也会更长。建议不要分多层，且一台主库一般挂3-5台从库吧。一般配置的mysql，并发最好控制在2000/s，挂5台的话，整体基本能支撑1w+/s的并发，再加上缓存和二

八定律，基本能支撑小10w/s的并发，很高了。如果还不能满足需求，那还是选择去分库吧。）

5.可落地分析：

两点影响落地使用。第一，数据一致性问题，一致性解决方案可解决问题。第二，主库单点问题，MHA+Mysql主从配置实现MySQL高可用，MHA是一个用于故障切换和主从提升的软件，要搭建MHA，要求集群至少要有三个节点，即一主二从

- **双主多重：**



1. **高可用分析**：高可用。
2. **高性能分析**：高性能。
3. **一致性分析**：存在数据一致性问题（参考上面解决）
4. **扩展性分析**：可以通过加从库来扩展读性能，进而提高整体性能。

5. 可落地分析：数据同步又多了一层，数据延迟更严重；存在数据一致性问题；多个主库同时插入数据时，主键冲突问题，ID统一地由分布式ID生成服务来生成可解决问题；（主库可分别用奇偶ID，设置步长）。

分库分表

需求

关系型数据库本身比较容易成为系统瓶颈，单机存储容量、连接数、处理能力都有限。当单表的数据量达到1000W或100G以后，由于查询维 度较多，即使添加从库、优化索引，做很多操作时性能仍下降严重。此时就要考虑对其进行切分了，切分的目的就在于减少数据库的负担，缩短查询时间。

分表： 解决单表海量数据的查询性能问题

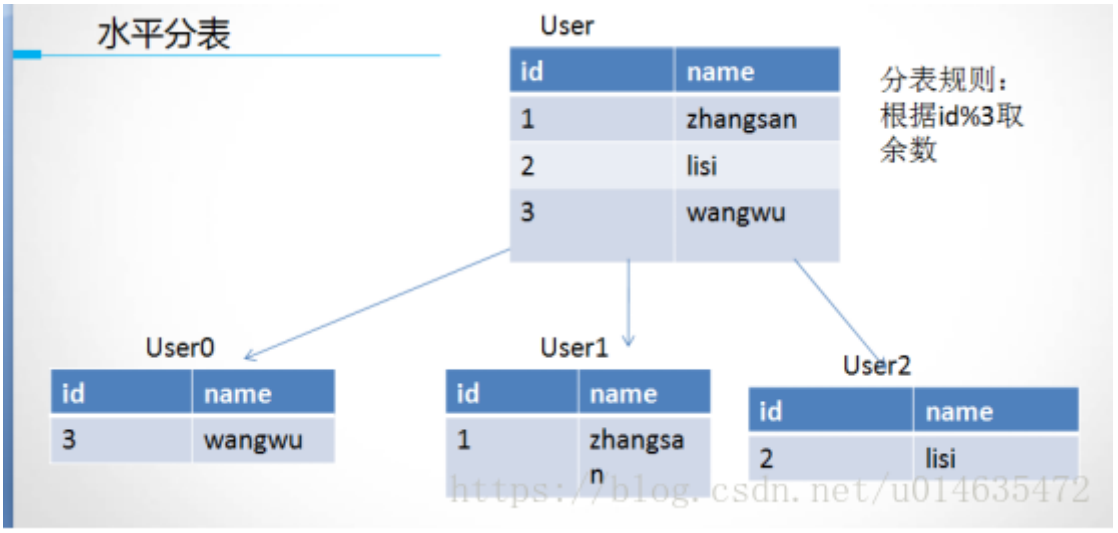
方式：分表分为水平分表和垂直分表。

库内分表只解决了单一表数据量过大的问题，但没有将表分布到不同机器的库上，因此对于减轻MySQL数据库的压力来说，帮助不是很大，大家还是竞争同一个物理机的CPU、内存、网络IO，最好通过分库分表来解决。

水平分表：

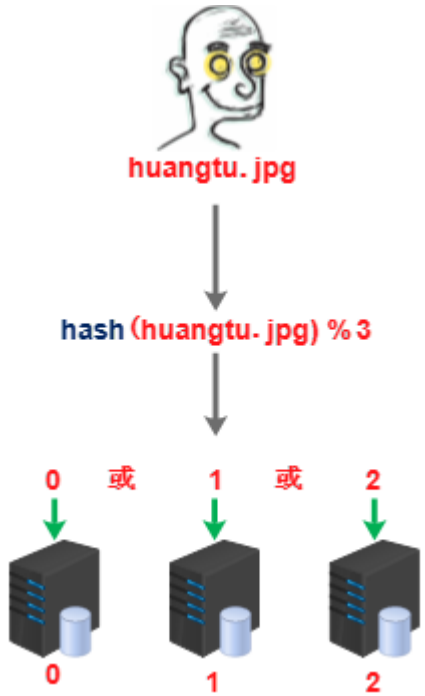
1.根据取模分表：分表策略通常是用户ID取模，如果不是整数，可以首先将其进行hash获取到整。

优点：数据分片相对比较均匀，不容易出现热点和并发访问的瓶颈；应用端改造较小，不需要拆分业务模块



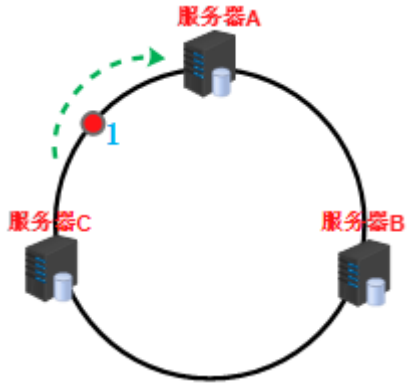
缺点：

- 跨分片的事务一致性难以保证；
- 跨库的join关联查询性能较差；
- 后期分片集群扩容时，需要迁移旧的数据；（使用一致性hash算法能较好的避免这个问题）



假设我们增加1台服务器，服务器的数量由3变成了4，此时仍然用上述方法对同一张图片进行缓存，那么这张图片所在的服务器的编号必定是与原来的3台服务器所在的编号是不同的，因为除数3变成了4，被除数不变的情况下，余数肯定不同，这情况带来的结果就是当服务器数量变动时，所有缓存的位置都要发生改变；
同理，如果3台缓存服务器中突然有一台出现了故障，无法进行缓存数据，那么需要移除故障机器，但是如果移除了一台缓存服务器后，数量从3变成了2，如果想访问有一张图片，这张图片缓存为位置必定发生改变，以前缓存的图片也会失去缓存的作用和意义，由于大量缓存在同一时间失效，造成了缓存的雪崩(血崩)；

一致性hash算法也是取模运算，只是，上面描述的取模算法是对服务器数量进行取模，而一致性hash是对2^32取模，对2^32取余是有依据的IPV4最大数量就是2^32，所以这样就能保证所有的ip的地址进行取余时不会重复一对应hash环上面的正数。



仍然使用图片的名称作为找到图片的Key，那么我们使用以下公式可以将图片映射到上图中的hash环上。hash(图片名称) % 2^32
因为从图片开始的位置开始，沿着顺时针方向遇到的第一个服务器就是A服务器，所以会被缓存到A服务器上。
算法说明：通过一致性hash算法这种方法，判断一个对象应该被缓存到那台服务器上的，将缓存服务器与被缓存对象都映射到hash环上以后，从被缓存对象的位置出发，沿着顺时针方向遇到的第一个服务器，就是当前对象将要缓存的服务器，由于被缓存对象与服务器hash后的值都是固定的，所以服务器不变的情况下，一张图片必定会被缓存到固定的服务器上，那么，当下次访问这张图片时，只要再次使用相同的算法进行计算，即可算出这张图片被缓存在那个服务器上，直接去对应的服务器上查找即可。
算法优势：服务器数量发生改变时，并不是所有都会失效，而只有部分缓存失效，前端的缓存仍然能分担整个系统的压力，而不至于所有压力都在同一个时间集中到后端服务器上。
（服务器A失效节点1会顺时针保存至服务器B）

2.根据数值范围分表，例1-9999一张表，10000-20000一张表

优点：单表大小可控，便于水平扩展，后期如果想对整个分片集群扩容时，只需要添加节点即可，无需对其他分片的数据进行迁移；连续分片可快速定位分片进行快速查询，有效避免跨分片查询的问题。

缺点：热点数据成为性能瓶颈。连续分片可能存在数据热点，被频繁的读写，而有些分片存储的历史数据，则很少被查询；

垂直分表：

方法：

1. 把大字段独立存储到一张表中
2. 把不常用的字段单独拿出来存储到一张表
3. 把经常在一起使用的字段可以拿出来单独存储到一张表

优势分析：MySQL底层是通过数据页存储的，一条记录占用空间过大会导致跨页，造成额外的性能开销。另外数据库以行为单位将数据加载到内存中，这样表中字段长度较短且访问频率较高，内存能加载更多的数据，命中率更高，减少了磁盘IO，从而提升了数据库性能。

垂直拆分标准：

1.表的体积大于2G并且行数大于1千万

2.表中包含有text，blob，varchar(1000)以上

3.数据有时效性的，可以单独拿出来归档处理

缺点：

- 部分表无法join，只能通过接口聚合方式解决，提升了开发的复杂度（可以通过增加汇总的冗余表改善，虽然数据量很大，但是可以用于后台统计或者查询时效性比较底的情况）
- 分布式事务处理复杂

分库：解决单台数据库的并发访问压力问题。

根据业务耦合性，将关联度低的不同表存储在不同的数据库

水平分库、垂直分库，类同分表；

表分区：

就是将一个数据量比较大的表,用某种方法把数据从物理上分成若干个小表来存储（类似水平分表）,从逻辑来看还是一个大表。

可将读写分离与分片结合起来，解决读写分离架构缺陷（每个节点必须保存完整数据，集群的扩展能力受限于单节点的存储能力）

分库分表带来的问题

1、事务一致性问题

分布式事务：分布式事务指事务的操作位于不同的节点上，需要保证事务的 AICD 特性。当更新内容同时分布在不同库中，不可避免会带来跨库事务问题。跨分片事务也是分布式事务，一般可使用"XA协议"和"两阶段提交"处理。

两阶段提交:通过引入协调者（Coordinator）来协调参与者的行为

2、跨节点关联查询 join 问题

解决这个问题的一些方法：

1）全局表：系统中所有模块都可能依赖的一些表，为了避免跨库join查询，可以将这类表在每个数据库中都保存一份

2）字段冗余：一种典型的反范式设计，利用空间换时间，为了性能而避免join查询。例如：订单表保存userId时候，也将userName冗余保存一份，这样查询订单详情时就不需要再去查询"买家user表"了。

3）数据组装：在系统层面，分两次查询，第一次查询的结果集中找出关联数据id，然后根据id发起第二次请求得到关联数据。最后将获得到的数据进行字段拼装。

4）ER分片：关系型数据库中，如果可以先确定表之间的关联关系，并将那些存在关联关系的表记录存放在同一个分片上，那么就能较好的避免跨分片join问题。

3、跨节点分页、排序、函数问题

4、全局主键避重问题

5、数据迁移、扩容问题