



成都返空汇网络技术有限公司
CHENGDU FANKONGHUI NETWORK TECHNOLOGY CO.,LTD.

<http://www.fkhwl.com>

大数据下MYSQL数据库 设计思路探讨

主讲人：陈飞

前言

在计算机领域，当单机性能达到瓶颈时，有两种方式可以解决性能问题，一是堆硬件，进一步提升配置，二是分布式，水平扩展。当然，两者都是一样的烧钱。

实际应用中，系统出现瓶颈的环节有多种情况：软件编码质量、服务器硬件配置、网络带宽、磁盘I/O、数据库...而数据库是最终环节的问题，当达到一定量级之后，最终要解决的也是数据库的性能。



一、分析思路篇

1. 项目背景
 2. 如何分析
 3. 学会计算
 4. 分析思路
 5. 设计思路
-

1、项目背景

在项目开始之前就需要充分了解清楚项目的背景等信息，比如：

- ✓ 产品面向的行业
- ✓ 用户群体
- ✓ 系统的类型
- ✓ 注册用户大致容量
- ✓ 业务数据产生规模
- ✓ ...

2、如何分析

- 以返空汇为例：我们面向的是大宗物流业务，要解决的问题的是车货匹配的问题，我们要面向的用户群体是传统行业中的司机大佬、信息部以及货主。
- 我们产品规划的是以手机App和PC端管理软件为主的系统，行业内货车的保有量大概为1500万，信息部家数大致在10万家以内，货主量暂时无法估计，以千万级计。
- 业务数据可粗略的根据业务情况做个估算，不需要准确，能帮助判断就好。

3、学会计算

- 计算什么？
- 假设要开发一个能承受500万PV/天的网站，服务器每秒要处理多少个请求才能应对？如果计算呢？
- 要解决这个问题，首先就得先回答500万PV是什么概念！
- 500万PV，服务器每秒需要承载多少的并发请求？服务器需要多大的带宽才能满足需求（有图片和无图片）？数据库服务器需要多少个连接？数据量大概多大？...
- 额...好像有点打岔了，这个算是分布式系统设计或性能测试里的一个范畴。
- 总之，学会计算PV、QPS、带宽等是一项基本的技能。

4、分析思路

- ① 区分业务类型，可能达到千万用户级别的应用业务场景，不同类型的业务场景做法会不一样，主要是由业务性质决定；
- ② 应用业务的核心KPI数值，产品每天的日活跃用户量大概多少？若是网站类型应用，还需要加入其他参数PV、UV等数据辅助决策，即时通信IM的消息量，系统的新增核心数据；

4、分析思路

- ③ 系统中每个用户可能产生的数据量大概多大，分固定部分，以及动态部分的方式统计分析，对非固定部分以参考值和结合实践跨度（注释：1年为硬性指标，2年为预期，3年可选，再长的时间段不考虑）的方式进行分析，然后预测出整个系统的用户所产生的数据条数和数据容量大概的估值；
- ④ 注册用户并不等于活跃用户，为此需要预估日活跃用户量大概多少？周活跃用户量大概多少？月活跃用户量大概多少？系统设计的最高并发量为多少？这数字还是非常有必要，不管是对数据量预估，还是对技术方案的选择都有帮助；

4、分析思路

- ⑤ 根据应用业务的特点，以及系统不同模块的功能特点，初期必须判断出可能负载最大的系统模块，对于可以静态化模块或功能，尽量要**Cache**起来，以降低系统的负载和提高前端响应速度；若是非**Cache**技术能解决的，是否可以考虑独立或通过整体水平扩展方式解决系统的负载和性能问题；
- ⑥ 针对系统中各个模块的功能或业务特点，大致那些用户数据会累计比较大，以及那些数据操作频率比较高；

4、分析思路

- ⑦ 不同的业务其对数据操纵不一样，要大致明白自己的应用：读写比如关系，也即：
SELECT:UPDATE:DELETE:UPDATE=?
- ⑧ 系统的整体架构中，必须考虑系统的稳定性、负载均衡和响应速度，为此必须考虑一些模块借助**Cache**、异步、消息队列等技术，进行一些特殊处理或折中做法，以达到目标；

5、设计思路

- ① 设计开始之前，必须优先进行业务的数据流梳理（注释：必须尽量考虑应用所有可能的功能模块），以及对业务优先进行优化和规划，然后根据数据流和功能考虑数据库的结构设计和优化；
- ② 千万级别用户量，若是非游戏行业的产品，建议考虑用户数据拆分架构设计，以及考虑后续未来1-2年的承受量，若是SNS平台必须考虑拆分，除非考虑上SSD、Fusion-io、存储等更高端的设备，用金钱换时间的方式支持技术改造；

5、设计思路

- ③ 数据拆分的核心与难处：同一个用户的数据尽量放在一起（拆分规则要尽量简单可执行），拆分之后用户关系的数据如何保存的抉择有多种（存2份或存1份放一个地方），难处数据的分页，统计合并等；
- ④ 要考虑一些冗余的方式解决SQL性能问题，但是又不能过多引入冗余而造成IO开销增加太多，冗余字段要尽量整型字段；
- ⑤ 数据库表对象的字段属性，要尽量考虑数字化（这里会涉及到字段的优化、表的优化、索引的优化等等）；

5、设计思路

- ⑥ 数据库设计过程中，对于索引组织结构要偏向共同操作最优先，其次应用外部用户级别的操作性能优先，最后内部用户的操作，应尽量隔离，例如：驾驶员、信息部的日常使用操作、内部营销或运营的查询、审核、统计等操作；
- ⑦ 数据库要从设计角度规避一些无法通过其他技术手段解决的模糊查询，类似全文索引的模糊查询，要走搜索引擎的模式，再通过数据库读具体的数据，一些必要的计数类型的数据，适当地考虑缓存；

5、设计思路

- ⑧ 重点解决数据库级别的数据分页问题，要学会从前端应用用户的体验不降低的情况下，达到更高效的数据分页做法；
- ⑨ 数据库的设计必须考虑使用什么类型配置的物理服务器，核心参数：内存、CPU、硬盘（这个是关键：硬盘类型（注：SATA、SAS、SSD）、多少块盘、转速、容量，以及做RAID几），RAID卡内存及RAID写模式也需要考虑进去，必须结合数据量和读写能力要求进行一个预算规划，不一定超准确，但是要八九不离十。



二、架构实战篇

1. 垂直拆分
 2. 水平拆分
 3. 分库分表策略
-

实战篇开始之前，先看个栗子，例子...

栗子！

假设我们有一台服务器，它可以承担1百万/秒的请求，这个请求可以的是通过http访问网页，通过tcp下载文件，jdbc执行sql，RPC调用接口...，现在我们有一条数据的请求是2百万/秒，很显然服务器hold不住了，会各种拒绝访问，甚至崩溃，宕机，怎么办呢？

一台机器解决不了的问题，那就两台。所以我们加一台机器，每台承担1百万。如果请求继续增加呢，两台解决不了的问题，那就三台呗。这种方式我们称之为水平扩展。如何实现请求的平均分配便是负载均衡了。

另一个栗子！

另一个栗子，我们现在有两个数据请求：


数据A：90万，数据B：80万。

上面那台机器也hold不住，我们加一台机器来负载均衡一下，每台机器处理45万数据A和40万数据B，但是平分太麻烦，不如一台处理数据A，一台处理数据B，同样能解决问题，这种方式我们称之为垂直拆分。



问题？

以返空汇为例，咋们的系统如何架构？？



好！刚刚例子里已经反复提到了两个概念：水平拆分和垂直拆分。

那么下面就针对这两点内容进行展开讨论。

小结一下：

- 什么是水平拆分？
- 什么是垂直拆分？

1、什么是垂直拆分？

垂直分库是根据数据库里面的数据表的相关性进行拆分，比如：一个数据库里面既存在用户数据，又存在订单数据，那么垂直拆分可以把用户数据放到用户库、把订单数据放到订单库。垂直分表是对数据表进行垂直拆分的一种方式，常见的是把一个多字段的大表按常用字段和非常用字段进行拆分，每个表里面的数据记录数一般情况下是相同的，只是字段不一样，使用主键关联。

简而言之：垂直拆分就是要把表按模块划分到不同数据库表中或者把单表按不同的字段进行拆分成多表（当然原则还是不破坏第三范式）。

又是一个栗子...

比如原始的用户表是：

Users

ID	username	email	first	last	location
1	jsmith	smith@example.com	John	Smith	Seattle, WA
2	ramsey	ramsey@example.com	Ramsey	White	Mt. Airy, MD
3	siegfried	crossbow@example.com	Siegfried	Faust	Rostock, Germany
4	amber	amber@example.com	Amber	Simpson	Las Vegas, NV
5	burke.j	jburke@example.com	Juliet	Burke	Miami, FL
6	owenh	harper@example.com	Owen	Harper	Cardiff, UK

垂直拆分（表字段拆分，非库拆分）后是：

Users

ID	username	email
1	jsmith	smith@example.com
2	ramsey	ramsey@example.com
3	siegfried	crossbow@example.com
4	amber	amber@example.com
5	burke.j	jburke@example.com
6	owenh	harper@example.com

UsersExtra

ID	first	last	location
1	John	Smith	Seattle, WA
2	Ramsey	White	Mt. Airy, MD
3	Siegfried	Faust	Rostock, Germany
4	Amber	Simpson	Las Vegas, NV
5	Juliet	Burke	Miami, FL
6	Owen	Harper	Cardiff, UK

垂直拆分解问题：

解决问题：

- ① 表与表之间的io竞争

不解决问题：

- ① 单表中数据量增长出现的压力

垂直拆分的优点:

- ① 可以使得行数据变小，一个数据块(Block)就能存放更多的数据，在查询时就会减少I/O次数(每次查询时读取的Block 就少)
- ② 可以达到最大化利用Cache的目的，具体在垂直拆分的时候可以将不常变的字段放一起，将经常改变的放一起
- ③ 数据维护简单

垂直拆分的缺点：

- 主键出现冗余，需要管理冗余列
- 会引起表连接JOIN操作（增加CPU开销）可以通过在业务服务器上进行join来减少数据库压力
- 单表大数据量依然存在性能瓶颈；依然存在单表数据量过大的问题（需要水平拆分）
- 事务处理复杂

2、什么是水平拆分？

水平拆分是通过某种策略将数据分片来存储，分库内分表和分库两部分，每片数据会分散到不同的MySQL表或库，达到分布式的效果，能够支持非常大的数据量。前面的表分区本质上也是一种特殊的库内分表。

库内分表，仅仅是单纯的解决了单一表数据过大的问题，由于没有把表的数据分布到不同的机器上，因此对于减轻MySQL服务器的压力来说，并没有太大的作用，大家还是竞争同一个物理机上的IO、CPU、网络，这个就要通过分库来解决。

简而言之：水平拆分就是把一个表按照某种规则把数据划分到不同表或数据库里。

上述栗子中修改为水平拆分，拆分结果是：

Users_A_M

ID	username	email	first	last	location
1	jsmith	smith@example.com	John	Smith	Seattle, WA
4	amber	amber@example.com	Amber	Simpson	Las Vegas, NV
5	burke.j	jburke@example.com	Juliet	Burke	Miami, FL

Users_N_Z

ID	username	email	first	last	location
2	ramsey	ramsey@example.com	Ramsey	White	Mt. Airy, MD
3	siegfried	crossbow@example.com	Siegfried	Faust	Rostock, Germany
6	owenh	harper@example.com	Owen	Harper	Cardiff, UK

实际情况中往往会垂直拆分和水平拆分的结合，即将Users_A_M和Users_N_Z再拆成Users和UserExtras，这样一共四张表。

水平拆分解决问题：

解决问题：

- ① 单表中数据量增长出现的压力

不解决问题：

- ① 表与表之间的io争夺

水平拆分的优点:

- ① 不存在单库大数据和高并发的性能瓶颈
- ② 应用端改造较少
- ③ 提高了系统的稳定性和负载能力

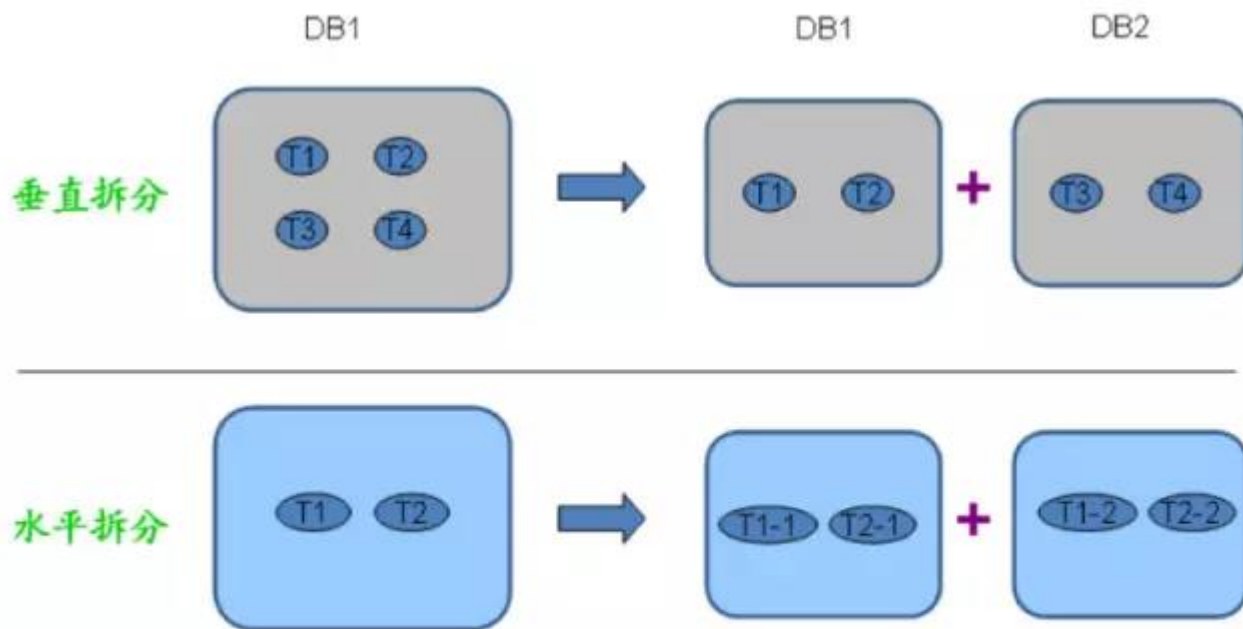
水平拆分的缺点：

- ① 分片事务一致性难以解决
- ② 跨节点Join性能差，逻辑复杂
- ③ 数据多次扩展难度跟维护量极大

又来一个小结...

- 通俗理解：水平拆分行，行数据拆分到不同表中；垂直拆分列，表数据拆分到不同表中或者直接拆分表到单独的库中（如下图）。
- 由上面栗子可知，垂直拆分能更清晰化模块划分，区分治理，水平拆分能解决大数据量性能瓶颈问题，因此常常就会把两者结合使用，这在大型网站里是种常见的策略。

垂直拆分和水平水平拆分：



两种分库方式

嗯...知道了什么是水平拆分、垂直拆分了，那...
实际应用中，怎么拆分？
遵循什么规则拆分？

什么事都是讲原则、讲逻辑的...

3、分库分表策略

- ① 达到一定数量级才拆分（800万）
- ② 不到800万但跟大表（超800万的表）有关联查询的表也要拆分，在此称为大表关联表
- ③ 大表关联表如何拆：小于100万的使用全局表；大于100万小于800万跟大表使用同样的拆分策略；无法跟大表使用相同规则的，可以考虑从java代码上分步骤查询，不用关联查询，或者破例使用全局表。

- ④ 破例的全局表：如item_sku表250万，跟大表关联了，又无法跟大表使用相同拆分策略，也做成了全局表。破例的全局表必须满足的条件：没有太激烈的并发update，如多线程同时update同一条id=1的记录。虽有多线程update，但不是操作同一行记录的不在此列。多线程update全局表的同一行记录会死锁。批量insert没问题。
- ⑤ 拆分字段是不可修改的

- ⑥ 拆分字段只能是一个字段，如果想按照两个字段拆分，必须新建一个冗余字段，冗余字段的值使用两个字段的值拼接而成（如大区+年月拼成zone_yyyymm字段）。
- ⑦ 拆分算法的选择和合理性评判：按照选定的算法拆分后每个库中单表不得超过800万
- ⑧ 能不拆的就尽量不拆。如果某个表不跟其他表关联查询，数据量又少，直接不拆分，使用单库即可。

数据拆分规则

- 范围
- 日期
- ID 取模
-

数据拆分方案

- SpringJDBC自行实现
- 使用第三方中间件

数据库中间件

产品名称	出品方	架构模型	支持DB	分库	分表	读写分离	是否开源
MySQL Fabric	MySQL官方	代理架构	MySQL	有	有	有	是
MySQL Proxy	MySQL官方	代理架构	MySQL	有	有	有	是
Amoeba	社区	代理架构	MySQL	有	有	有	是
Cobar	阿里巴巴	代理架构	MySQL	有	无	有	是
TDDL	阿里巴巴	客户端架构	无限制	有	有	有	部分
Atlas	奇虎360	代理架构	MySQL	有	有	有	是
ShardingJDBC	当当	客户端架构	MySQL	有	有	有	是
OneProxy	平民软件	代理架构	MySQL	有	有	有	否
MyCat	社区	代理架构	MySQL	有	有	有	是
...							

篇后

骚年，你以为了解上面的东西之后就完了吗？
了吗？ 吗？

上面的东西仅仅告诉大家一些基础的思维、逻辑和架构思路，但是，具体的实施过程还有很多坑等着你跳呢！

比如：

- 拆分键如何设计？
- 前期数据库设计如何规划？
- 跨库如何查询？
- 多表关联如何查询？
- 拆分后事务如何处理？
- 深度分页如何解决？
- 数据同步延迟了怎么办？
- 如何保证客户和运营数据都能快速的查询？
- 如何拆分扩展时不需要数据迁移？
- 异地数据备份如何操作？
- ...



未完待续...