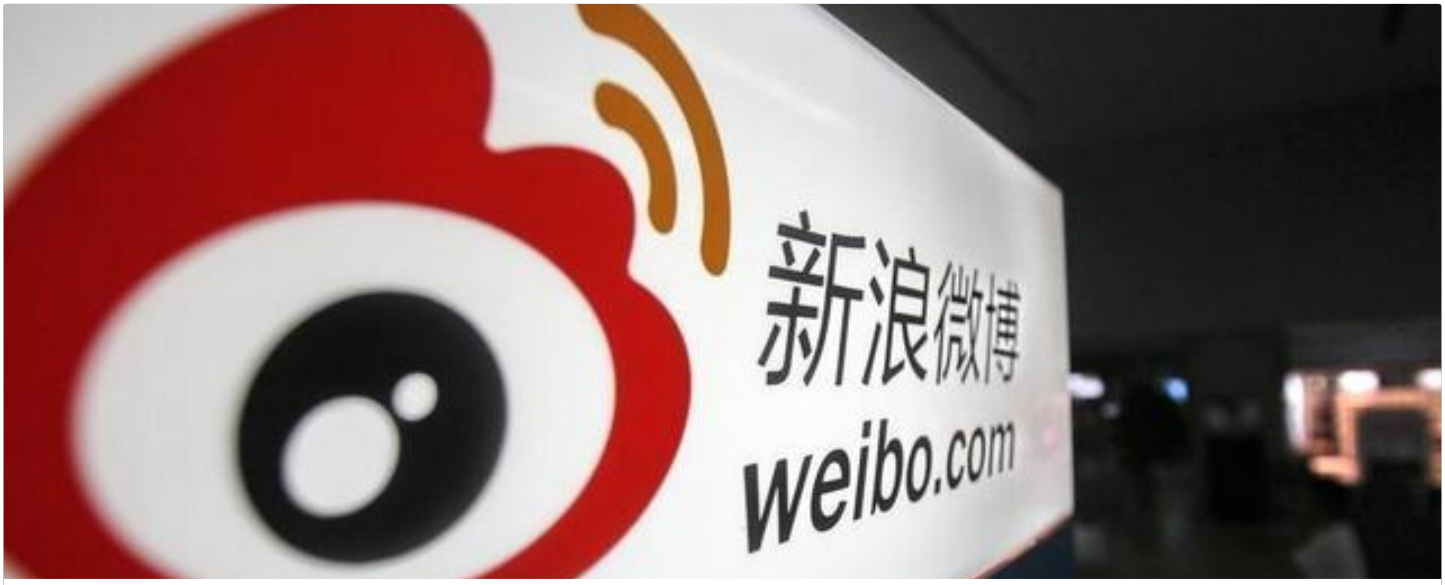

微博数据库那些事儿：3个变迁阶段背后的设计思想

高可用架构 | 2016-03-03 11:15:40 阅读数：3209

到了第三阶段，微博积累下来的数据库设计习惯，通常会采用一些“反模式”设计思路。

编者按：高可用架构分享及传播在架构领域具有典型意义的文章，本文由肖鹏在高可用架构群分享。转载请注明来自高可用架构公众号「ArchNotes」。

> 肖鹏，微博研发中心技术经理，主要负责微博数据（MySQL/Reids/HBase/Memcached）相关的业务保障、性能优化、架构设计，以及周边的自动化系统建设。经历了微博数据库各个阶段的架构改造，包括服务保障及 SLA 体系建设、微博多机房部署、微博平台化改造等项目，10 年互联网数据库架构和管理经验，专注于数据库的高性能和高可用技术保障方向。

数据库专家的成长感触

“与 MySQL 结缘主要也是源于兴趣。第一份工作是在一家小公司，由于人手有限，各个领域的工作都要接触，相比之下我发现还是对数据库最感兴趣，所以就一直从事和数据库相关的技术工作了。而随着工作年限的增加，在数据库方面积累的经验也逐渐增多，越来越觉得数据库管理员（DBA）是一个偏实践的工种，很多理论上的东西在现实中会有各种的变化，比如“反范式”设计等。因此，如果想成为数据库方面的专家，建议大家一定要挑选好环境，大平台很多时候会由于量变引发质变产生很多有挑战的问题，而解决这些问题是成为技术专家的必经之路。” —— 肖鹏

微博数据库经历的变迁

首先为大家分享微博数据库经历的几个重要的阶段。

初创阶段

初期微博作为一个内部创新产品，功能比较简洁，**数据库架构采用的是标准 1M/2S/1MB 结构，按照读写分离设计，主库承担写入，而从库承担访问。**如果访问压力过大，可以通过扩容从库的数量获得 scale out 的能力。

个人认为，在初期，这种架构其实就可以满足业务的增长了，没有必要进行过度设计，开始就搞得过于复杂可能会导致丧失敏捷的可能。

爆发阶段

随着微博上线之后用户活跃度的增高，数据库的压力也与日俱增，我们首先通过采购高性能的硬件设备来对单机性能进行 scale up，以达到支撑业务高速发展的需求。然后，通过使用高性能设备争取来的时间对微博进行整体上的业务垂直拆分，**将用户、关系、博文、转发、评论等功能模块分别独立存储，并在垂直拆分的基础上，对于一些预期会产生海量数据的业务模块再次进行了二次拆分。**

对于使用硬件这里多说几句，由于微博最开始的时候就出现了一个很高的用户增长峰值，在这个阶段我们在技术上的积累不是很丰富，而且最主要的是没有时间进行架构改造，所以通过购买 PCIE-Flash 设备来支持的很多核心业务，我现在还清楚记得最开始的 feed 系统是重度依赖 MySQL 的，在 2012 年的春晚当天 MySQL 写入 QPS 曾经飙到过 35000，至今记忆犹新。

虽然看上去高性能硬件的价格会比普通硬件高很多，但是争取来的时间是最宝贵的，很有可能在产品生命的初期由于一些性能上的问题引发产品故障，直接导致用户流失，更加得不偿失。所以，**个人认为在前期的爆发阶段，暴力投入资金解决问题其实反而是最划算的。**

继续说数据库拆分，以博文为例。博文是微博用户主要产生的内容，可预见会随着时间维度不断增大，最终会变得非常巨大，如何在满足业务性能需求的情况下，尽可能地使用较少的成本存储，这是我们面临的一个比较有挑战性的问题。

- 首先，我们**将索引同内容进行了拆分**，因为索引所需存储空间较少，而内容存储所需空间较大，且这两者的使用需求也不尽相同，访问频次也会不同，需要区别对待。
- 然后，**分别对索引和内容采用先 hash，再按照时间维度拆分的方式进行水平拆分**，尽量保障每张表的容量在可控范围之内，以保证查询的性能指标。
- 最后，**业务先通过索引获得实际所需内容的 id，再通过内容库获得实际的内容**，并通过部署 memcached 来加速整个过程，虽然看上去步骤变多，但实际效果完全可以满足业务需求。

上图乍一看和上一张图一样，但这其实只是一个博文功能模块的数据库架构图，我们可以看到索引和内容各自分了很多的端口，每个端口中又分了很多的 DB，每个 DB 下的表先 hash 后按照时间维度进行了拆分，这样就可以让我们在后期遇到容量瓶颈或者性能瓶颈的时候，可以选择做归档或者调整部署结构，无论选择那种都非常的方便。另外，在做归档之后，还可以选择使用不同的硬件来承担不同业务，提高硬件的利用率、降低成本。

在这个阶段，我们对很多的微博功能进行了拆分改造，比如用户、关系、博文、转发、评论、赞等，基本上将核心的功能都进行了数据拆分，以保障在遇到瓶颈的时候可以按照预案进行改造和调整。

沉淀阶段

在上一个阶段，微博的数据库经历了很多的拆分改造，这也就直接造成了规模成倍增长的状况，而业务经历了高速增长之后，也开始趋于稳定。在这个阶段，我们**开始着重进行自动化的建设**，将之前在快速扩张期间积攒下来的经验用自动化工具加以实现，对外形成标准化和流程化的平台服务。**我们相继建设改造了备份系统、监控系统、AutoDDL 系统、MHA 系统、巡检系统、慢查系统、maya 中间件系统。**并且为了提高业务使用效率、降低沟通成倍，相对于内部管理系统，重新开发了 iDB 系统供数据库平台的用户使用。通过 iDB 系统，用户可以很便捷地了解自己业务数据库的运行状态，并可以直接提交对数据库的 DDL 修改需求，DBA 仅需点击审核通过，即可交由 Robot 在

线上执行，不但提高了工作效率，也提高了安全性和规范性。

由于涉及的自动化系统比较多，就不一一展开描述了，其实个人理解，在产品发展到一定阶段之后无论如何运维都会进入到自动化阶段，因为前期活儿少，人工足够支持变更和其中操作，且有很多特殊情况需要人，尤其是人脑的介入判断和处理。

这里要额外重点说一下规范的重要性。**以 MySQL 开发规范来说，如果提前做好约定，并进行好限制**，虽然开发人员在使用的过程中会感觉受到的约束，但是这可以避免线上发生完全不可控的故障，并且有些问题由于规范的存在就永远不会发生了。

举个例子。MySQL 的慢查是导致线上性能慢的罪魁祸首，但是很多时候并不是没有 index，只是由于代码写得有问题，引起了隐式转换等问题。在这种情况下，我们一般建议所有的 where 条件都加上双引号，这样就可以直接消除隐式转换的可能性了，开发人员在写代码的时候也不用刻意去考虑到底是字符型还是 int 型。

继续说自动化。过了初期阶段、规模扩大之后，就会出现活多人少的情况，这种压力会促使大家自动去寻求解决方案，也就自然而然地进行了自动化改造了。当然，业务稳定后有时间开发了，其实是一个更重要的原因。

个人认为自动化分为两个阶段。**第一个阶段是机器替代人工**，也就是将大部分机械劳动交给程序来实现，解决的是批量操作，重复性劳动的问题；**第二个阶段是机器替人**，也就是机器可以替人进行一定的判断之后进行自我选择，解放的是人力。不过第二个阶段是我们一直追求的理想状态，至今也仅完成了很简单的一些小功能，比如动态调整 max mem 等逻辑非常简单的功能。

微博数据库的优化和设计

接下来介绍一下微博数据库平台最近做的一些改进和优化。

数据库平台并不仅有 MySQL 还有 Redis、Memcached、HBase 等数据库服务，而在缓存为王的趋势下，微博 2015 年重点将研发精力投入在 Redis 上。

微博使用 Redis 的时间较早，并且一开始量就很大，于是在实际使用过程中遇到了很多实际的问题，我们的内部分支版本都是针对这些实际问题进行优化的，比较有特点的有如下几个。

- **增加基于 pos 位同步功能**。在 2.4 版本中，Redis 的同步一旦出现中断就会重新将主库的数据"全部"传输到从库上，这会造成瞬时的网络带宽峰值，并且对于数据量较大的业务来说，从库恢复的时间较慢，为此我们联合架构组的同学借鉴 MySQL 的主从同步复制机制，将 Redis 的 aof 改造为记录 pos 位，并让从库记录已经同步的 pos 位，这样在网络出现波动的时候即使重传，也仅是一部分数据，并不会影响业务。
- **在线热升级**。在使用初期，由于很多新功能的加入，Redis 版本不断升级，为了不影响业务，每次升级都需要进行主库切换，给运维带来了很大的挑战，于是开发了热升级机制，通过动态加载 libredis.so 来实现版本的改变，不再需要进行主库切换，极大地提升了运维效率，也降低了变更带来的风险。
- **定制化改造**。在使用 Redis 的后期，由于微博产品上技术类的需求非常多，为此专门开发了兼容 Redis 的 redisscounter，用以专门存储技术类数据，通过使用 array 替换 hash table 极大地降低内存占用。而在此之后，开发了基于 bloom filter 的 phantom 解决判断类场景需求。

Redis 中间件

在 2015 年我们自研的 Redis 中间件 tribe 系统完成了开发和上线，**tribe 采用有中心节点的 proxy 架构设计，通过 configer server 管理集群节点，并借鉴官方 Redis cluster 的 slot 分片的设计思路来完成数据存储，最终实现了路由、分片、自动迁移、fail over 等功能**，并且预留了操作和监控的 API 接口，以便同其他的自动化运维系统对接。

我们开发 tribe 最主要的目的是解决自动迁移的问题，由于 Redis 内存使用会呈现波动性变化，很多时候前一天还是 10%，第二天有可能就变成了 80%，这种时候人工去迁移肯定无法响应业务的变化，而且如果这时候恰巧还碰到了物理内存上的瓶颈，那就更麻烦了，涉及业务进行重构数据 hash 都有可能导致故障的发生。

基于 slot 的动态迁移，首先对业务无感知，其次不再需要整台服务器，只需找有可用内存的服务器就可以将部分 slot 迁移过去，直接解决扩容迁移问题，可以极大地提高服务器的利用率、降低成本。

提供的路由功能，可以降低开发门槛，再也不用将资源逻辑配置写到代码或者前端配置文件中了，每次更改变更的时候也不用再进行上线了，这极大地提高了开发效率，也降低了线上变更引发的故障风险，毕竟 90% 的故障是主动变更引发的。

补充一点，关于重复造轮子，个人认为每个公司都有自己的场景，开源软件可以给我们提供一个很好的解决思路，但并不能百分百适应应用场景，所以重构并不是坚决不可接受的，有些事情你总要妥协。

Databus

由于我们先有 MySQL 后有 Redis 和 HBase 等数据库，故存在一种场景就是目前数据已经写入 MySQL 中，但是需要将这些数据同步到其他的数据库中，我们为此开发了 Databus，可以基于 MySQL 的 binlog 将数据同步到其他异构的数据库中，并且支持自定义业务逻辑。目前已经实现了 MySQL 到 Redis 和 MySQL 到 HBase 的数据流向，下一步计划开发 Redis 到 MySQL 的数据流向。

我们开发 databus 最初的初衷是解决写 Redis 的问题，由于有些数据即需要写入 MySQL 中，也需要写入 Redis 中。如果在前端开启双写也是可以解决的，但是这会造成代码复杂现象；如果在后端实现一个数据链路，会让代码更加清晰，同时也可以保障数据的最终一致性。后来在实际应用中，databus 慢慢也开始承担导数据的功能。

下面说一下目前**微博积累下来的数据库的设计习惯。通常来说我们都会采用一些“反范式”的设计思路**，而“反范式“设计带来便利的同时确实也带来了一些问题，尤其是在数据规模变大之后。有如下几种解决方案。

- **预拆分**。在接到需求的时候提前针对于容量进行评估，并按照先垂直后水平进行拆分，如果可以按照时间维度设计，那就纳入归档机制。通过数据库的库表拆分，解决容量存储问题。
- **引入消息队列**。利用队列的一写多读特性或多队列来满足冗余数据的多份写入需求，但仅能保障最终一致性，中间可能会出现数据延迟。
- **引入接口层**。通过不同业务模块的接口将数据进行汇总之后再返回给应用层，降低应用层开发的编码复杂度。

另外一点就是，如果数据库预估量比较大的话，我们会**参考博文的设计思路，在最开始的时候进行索引和内容的分离**，并设计好 hash 和时间维度的分表，最大可能地减少后续拆分时可能遇到的问题和困难。

微博数据库平台未来的计划

最后，我想分享一下微博数据库平台发展的一点思考，希望可以给大家提供一些思路，当然，更希望大家也给我提出一些建议和意见，让我少走弯路、少掉坑。

随着业务的发展，会遇到越来越多的场景，**我们希望可以引进最适合的数据库来解决场景问题**，比如 PostgreSQL、SSDB 等。同时，利用 MySQL 新版本的特性，比如 MySQL 5.7 的并行复制、GTID、动态调整 BP，不断优化现有服务的性能和稳定性。

另外，**推进现有 NoSQL 服务的服务化，通过使用 proxy 将存储节点进行组织之后对外提供服务**，对外降低开发人员的开发复杂度和获取资源的细度，对内提高单机利用率并解决资源层横向扩展的瓶颈问题。

同时，**尝试利用各大云计算资源，实现 cache 层的动态扩缩容**，充分利用云计算的弹性资源，解决业务访问波动的问题。

Q & A

1. 数据和索引分离是业务层做还是中间件做？感觉中间件做了很多工作，这部分能不能稍微展开一下？

由于在当初拆分改造的时候并没有考虑中间件的方案，故目前是业务逻辑上实现的索引和内容的分离。以我个人的经验来看，即使使用中间件的解决方案，依然应该在业务逻辑层将索引和内容分割。

中间件最核心的功能就是让程序和后端资源隔离，后端不管有多少资源，对于程序来说都是一个统一的入口，所以中间件解决的是水平拆分的问题，而索引和内容分离属于垂直拆分的范围，个人认为不应该由中间件解决。

2. 印象最深的一次数据库服务故障能否回忆并说几点注意事项？

要说印象最深的一次就是数据库服务的故障，当时有个同事不小心执行了 drop table 命令，了解数据库的人都知道这个命令有多大的威力，我们利用架构上面争取来的时候紧急进行了单表恢复，虽然降级了一段时间，但是整体上并没有影响用户。

对此我要说的注意事项就是规范。自那之后，我们修改了所有删表需求的流程，并保证严格执行，无论多么紧急的删除需求都必须进行24小时的冷却，具体如下。

- 执行 rename table 操作，将 table rename 成 table—will-drop。
- 等待 24 小时之后再执行 drop 操作。

3. 在微博暴涨阶段，对表进行拆分的部分，“对索引和内容进行hash，之后再按着时间纬度进行拆分”，这个做hash的部分是否能展开说一下？

这个其实没有那么复杂。首先我们会预估一下一年大概的数量级别，然后计算需要拆分的表数目，并且尽量将每个表控制在 3 千万行记录以内（当然这只是希望，现实证明计划赶不上变化）。比如我们使用模 1024 的办法，根据博文 id 将所有产生的博文分到 1024 张表中（这个博文 id 又涉及我们的 uuid 全局发号器就不展开了）。

由于微博大部分用户产生的内容都会和时间挂钩，所以时间维度对我们来说是强属性，基本都会有，假设每个月都建表，这样就等于每个月会生产 1024 张表。如果数据库的容量出现瓶颈了，我们就可以根据时间维度来解决，比如将 2010 年的所有表都迁移到其他的数据库中。

4. Linkedin 多年前出过一个类似 databus 的项目，后续也有一些开源项目支持 MySQL 到 HBase、ES 等的数据同步，微博的做法能不能展开一下？

这个异构数据的同步确实很多公司都有，据我所知阿里的 DRC 也是做同样的事情。我们的实现思路

主要就是依赖于 MySQL 的 binlog，大家都知道在 MySQL 的 binlog 设置成 row 格式的情况下，它会将所有受到影响的数据都记录到日志中，这样就提供了所有数据的变化。

我们通过解析 row 格式的 MySQL binglog 将数据的变化读取到 databus 中，然后将实际需要的业务逻辑通过。so 文件 load 到 databus 中，databus 会根据业务逻辑重新再处理一下数据的变化，然后输出给下游资源。

databus 这个项目我们已经开源，大家可以直接在 GitHub 上搜“MyBus”即可。

5. 问题 1 中说的索引是指什么，如何找到对应内容的算法？

索引其实指的并不是内容的算法，举个例子，如果你需要存储博文，那么必然会存储一个唯一的 id 来进行区分，然后会存储一些这个博文发表时候的状态，比如谁发的、什么时候发的、我们认为这些状态和 id 都是索引；而博文内容就是实际的内容，由于内容比较大，和索引存储在一起会导致 MySQL 的性能下降，而且很多查询只需要最终拿到博文 id 其实就等于拿到了实际的博文。

我们可以对索引进行各种过滤之后得到一个最终要输出给用户的索引 list，再根据这个 list 从内容库中查找实际内容，这样就符合 MySQL 的返回结果集越小性能越高的规律，从而起到优化的效果。

6. NoSQL发挥着哪些作用？

在微博NoSQL发挥了越来越重要的作用，比如说 rediscounter，我们自研的计数服务，当初计数存贮在 MySQL 中，由于所有计数都是 update set count = count +1 这种高并发对单行数据的写操作，会引发MySQL多种锁，而且并发越高锁得越厉害。

由下图可见，如果对单行的并发操作达到500 以上，tps就会从上万变成几百，所以MySQL无论怎么优化都无法很好地支持这个业务场景，而Redis就可以。

我个人认为，NoSQL 就像瑞士军刀，在最适合的地方它就是最优的方案，个人认为这也是 NoSQL 未来发展的方向，每一个都有一个最佳场景。

7. 我们公司将来会有很多智能设备，今年或许就两万个以上的设备，一年后可能再翻很多倍，要收集数据，都是 JSON 报文格式，JSON 里字段标识不同类型的报文，看似不太适合以字符串存进 varchar 或 longtext 字段，DB 存储是否可充分利用 MYSQL 5.7 的原生JSON，存一列就搞定，而不必用无法扩展的“宽列”进行存储，或者POSTGRES 还是其他的 DB 能提供更方便的存储吗？前期还用的时候，MySQL 5.7 还没 FINAL, 利用 mariadb 多主分担N多设备接进来，DB 写入层的负载，对这种大量设备一直要传数据的写入场景有什么指导？

我们其实也在看 MySQL 5.7 的新特性，其中 JSON 对于数据库设计会带来比较大的冲击，按照你的说法，确实不应该使用 varchar 或者 text 之类的字段，不过我个人建议智能设备这种场景，如果有可能，最好直接上 HBase，因为迟早数据量会变得 MySQL 难以支撑，这属于天生没有优势。

8. “数据和索引分离”是什么意思呢，是数据文件和索引文件分开存放到不同机器上吗？

应该是内容和索引，这样更好理解，大家把这个理解为业务层上的垂直拆分就好。由于进行了垂直拆分，必然存在于不同的数据库实例中，所以可以放在一台物理机上，也可以放到不同的物理机上，我们按照习惯都是放在不同的物理机上的，以避免互相干扰。

9. 哪些信息存MySQL？哪些存NoSQL？用的是哪种NoSQL？

这就涉及一个分层存储的问题了，目前我们主流是 MySQL + Redis+mc，mc 和 Redis 都用来抗热点和峰值，而 MySQL 则为数据落地，保障最终有原始数据可查。大部分请求会到 mc 或者 Redis 层就返回了，只有不到 1% 的数据会到 MySQL上。

当然也有特殊的，比如之前说的计数服务，我们就把 rediscounter 当初落地存储使用，后面并没有在 MySQL 中再存储一份数据了。

最后，个人认为 NoSQL 最大的优势是开发的便捷性，开发人员使用 NoSQL 会比使用 MySQL 更简单，因为就是 KV 结构，所有查询都是主键查询，不用考虑 index 优化的问题，也不用考虑如何建表，这对于现在的互联网速度来说是有致命诱惑的。

10. 请问全局唯一发号器和自动生成id对业务来说优劣分别是什么？

全局唯一发号器有很多实现的方案，我们是用 mc 协议改的 Redis，主要追求性能。我也见过使用 MySQL 的自增 id 作为发号器，但是由于 MySQL 的锁太重，业务起量之后经常会出现问题，所以就放弃了。

再说另外一个好处，全局唯一发号器的开发难度并不高，可以将一些你想要的属性封装到 uuid 中，比如 id 的格式可以是这样“时间戳 + 业务flog + 自增序列数”，这样你拿到这个 id 的时候直接就可以知道时间和对应的业务了，与 MySQL 自身发出来的简单的一个没意义的序列号相比，可以做更多的事情。

了解微博技术团队分享的其他文章

- Upsync：微博开源基于Nginx容器动态流量管理方案
- 支撑微博千亿调用的轻量级RPC框架：Motan
- 微博基于Docker的混合云平台设计与实践
- 微博“异地多活”部署经验谈
- 微博基于Docker容器的混合云迁移实战
- 单表60亿记录的MySQL优化和运维之道
- 微博在大规模、高负载系统问题排查方法

微博技术团队招聘各路技术人才，包括数据库 MySQL/NoSQL DBA、大数据工程师、C/C++，Java，运维等技术岗位，工程师全部配备 MacBook Pro 及 DELL 大屏幕显示器，并具有丰富的开发秘籍及培训文档，是工程师体现技术价值，提升个人能力最佳的选择。欢迎扫描二维码了解岗位详情。

本文策划李庆丰，编辑刘芸，转播叶青，审校 Tim Yang，想了解更多数据库设计与优化文章请关注本公众号。转载请注明来自高可用架构 「ArchNotes」微信公众号及包含以下二维码。