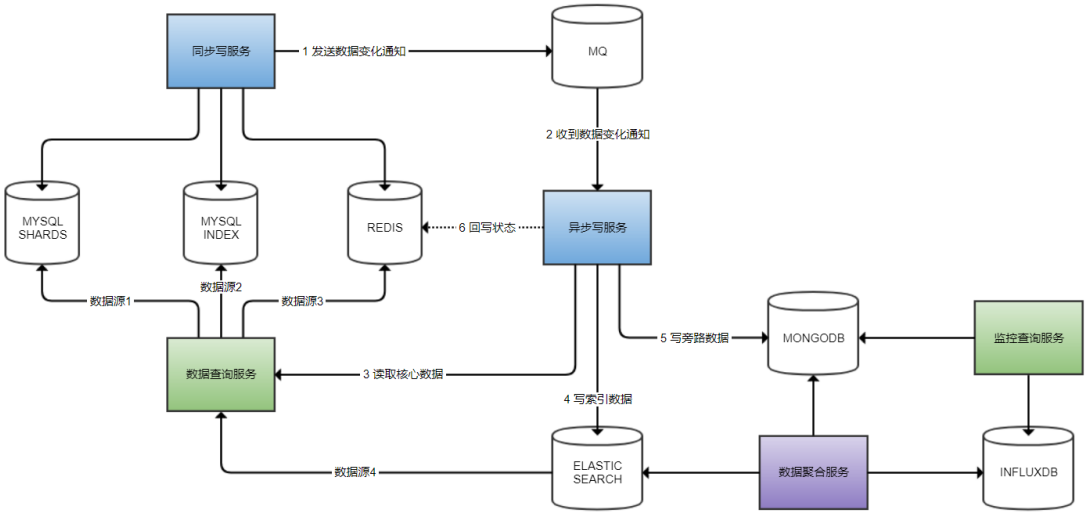
朱晔的互联网架构实践心得 S1E3：相辅相成的存储五件套

【下载本文 PDF 进行阅读】

这里所说的五件套是指**关系型数据库**、**索引型数据库**、**时序型数据库**、**文档型数据库**和**缓存型数据库**。



上图显示了一套读写服务搭配这五种类型数据库的例子：

1. 这里只是说明了我们可以这么来搭配这些类型的数据库，不是说我们所有的应用都需要用到这些类型的数据库。
2. 同步写服务负责第一时间把重要的数据落地和落缓存。
3. 异步写服务通过监听 MQ 来感知数据的变化，然后重新读取最新的数据来把数据写入其它次要数据源，比如文档性数据库和索引型数据库，需要的话可以在缓存中回写一个状态。
4. 由一个专门的数据查询服务来根据需求做数据路由，根据需求和性能因素，从不同的数据源读取数据。
5. 数据聚合服务根据需求从次要数据源进一步读取数据以时间维度进行聚合，聚合到时间序列数据库，供监控查询服务查询。

下面我们来具体说说这些存储系统。

关系型数据库

毫无疑问，强事务性的数据写入 MySQL 之类的关系型数据库是最可靠的，搭配 SSD 盘的使用，关系型数据库也很容易达到万级的 QPS。对于超大数据量加上超大并发的应用来说，单表的数据量过千万伴随着数万的 QPS 很难以单体数据库来支撑，我们需要对数据表进行 Sharding 分片处理，把数据按照一定的维度切分到比如 128 个数据表，然后分散在 8 套甚至 16 套数据集群，这样每一台 MySQL 的实例只需要承受 1/8 或 1/16 的请求压力而且数据量更小。随之带来的问题是，我们需要对应用进行改造，使之只能按照一定的查询条件来查询这个切片后的表，如果不带条件或带任意条件的话，我们是无法知道数据实际存储在哪个表哪个实例上的。

这确实是一个比较麻烦的地方，我们的查询条件可能有十几个，只能按照一个维度来查询满足不了我们的需求。一个折中的方式是我们引入所谓的 Index 数据表，也就是在写入实际的完整数据到 Sharding 的数据表的同时，我们把数据表里需要查询的字段写入一个专门的没有经过 Sharding 处理的 Index 数据表，这个数据表里存放的几乎没有 varchar 类型的数据，全部是各种 bigint 的各类业务 ID 或是 tinyint 类型的各种状态，以及时间。由于这个表非常亲，虽然数据条数多但是表空间几乎可以在数据库的缓存中容纳，性能会高不少。对于实时性要求非常强的基于条件的查询可以从这个数据表来进行查询。而 Sharding 后的数据只能用于按 ShardKey 来进行查询。

缓存数据库

Redis 是最常用的分布式缓存解决方案，几乎在任何互联网应用中都会用到，特点是：

1. 能持久化数据，但是我的观点是缓存数据库还是仅仅作为缓存的好，要能够承受丢失数据的风险，否则可能会死的比较难看。因为 RDB 或主从复制导致的一些事故也是层出不穷的。
2. 丰富的数据结构是一定要利用的，丰富的数据结构代表了可以依赖丰富的 API 在服务端做复杂的运算，性能比反序列化取出后运算再序列化存入效率高的多。有的时候甚至可以把这些数据结构和 API 组合在一起碰撞出绝妙的方案以极高效的方式实现一个高性能的业务逻辑。可以看看《Redis 实战》一书。

3. 超高的性能（当然了，配合一些集群方案比如 codis 就更上一层楼了）足以抵挡任何业务请求的直接访问，很多时候缓存的方案挂是挂在因为各种各样的原因穿透缓存而不是 Redis 档不住。
4. 丰富的集群和高可用方案以及各类各种实用的功能（管道、事务、Lua 脚本），5.0 的版本还推出了 Stream 特性来替代少有人关注的 Disque 值得关注。

所以 Redis 的应用也很广泛：

- 数据缓存
- 分布式锁
- 消息队列
- 服务端运算

在上图的架构中，我们通过同步写服务对数据库和缓存进行双写，目的也就是为了让缓存中能有新鲜热数据，不管是对内还是对外这种单条数据的查询可以直接路由到缓存。

文档型数据库

文档型数据库的代表就是耕耘多年的 Mongodb，我在一些非重要业务的场景使用过 Mongodb 几次，我的评价如下（最近 1 年多没有碰过 Mongodb，也可能评价有失偏颇）：

1. 超高的写入性能，非常不错的读取性能（和 Redis 是不能比的，性质不同），数据量增多后可能会有很厉害的性能衰退，不是 Hbase 那种无底洞型的存储，不维护就往里面一直堆数据进去最后的性能可能比如 MySQL。
2. 因为存的是文档，所以是弱结构的，存一些事先不能确定的数据非常非常合适，而且以后要查的时候可以任何加索引对需要的数据进行搜索查询。一个很实用的场景就是作为**爬虫的数据源**，数据变化多端而且不那么重要，而且写入性能很重要。
3. 不太可靠和稳定，可能会丢数据，强烈不建议作为核心数据存储，建议作为一个旁路数据库用在非关键的业务。比如在上图的架构图中，我们可能会拿到核心数据后再从其它地方去补一些数据然后进行适当的加工，保存到 Mongodb 作为一个监控数据库或者面向后台的数据库来用（MEAN 套件之一，可以想象对于简单的应用来说配合脚本语言用起来多舒服了），挂了也就挂了，没挂的话可以分担很多 MySQL 的压力。

4. 玩法虽然多，什么 Sharding、复制、集群都有，但随着数据量的增多运维可能是一个大坑，很可能遇到集群全军覆没无法启动的情况，数据的恢复耗时很长。内存的使用相当疯狂，对硬件的使用总感觉性价比不高。

索引型数据库

ElasticSearch 作为其代表是最近几年的黑马。ELK 集群各大互联网公司都有使用，只要集群配置得当，每秒几十万的写入不是大问题，毕竟彻底的分布式化理论上可以有无限高的写入能力。ES 的特点如下：

1. 非常丰富的查询 API，不仅仅是全文索引查询，普通的查询 API 丰富多样，组合起来可以在服务端完成各种业务逻辑，基本上 SQL+MySQL 可以实现的，ES 查询都可以实现，而且还多了更强大的全文搜索。当然，查询的语法稍显晦涩肯定没有 SQL 来的直挂。
2. 类似于 Mongodb 的 schema-free，无需实现定义表结构。
3. 还算强大的写入和读取能力，当然，索引多的话写入文档的效率肯定会降低。这也是图中对于 ES 的写入由专门的异步流程进行的原因。
4. ES 天生的分布式配置决定了，在写入亿、十亿的数据量之后，还能在相当可以接受的时间内（比如 10 秒）完成一个多条件复杂查询，对于 MySQL 这个量级下这样的查询可能需要 10 分钟甚至 100 分钟的时间来执行，完全不能接受。
5. ES 对嵌套型数据的查询支持不错，经过测试我们倾向于把多标关联的数据作为一个大的嵌套的 JSON 拍扁了直接存入 ES，比如我们可以把用户个人唯独的基本信息+充值订单+提现订单+投资订单，一人一个 JSON 存进去，然后对于嵌套的下层 JSON 数据也是可以方便的利用查询 API 进行查询。

因为这些特点，在这个架构图上，我们把 ES 也作为了查询服务的数据源，对于满足下面这些条件的查询，我们可以走 ES：

- 对数据延迟不敏感，可以接受一段时间查不到新鲜数据
- 查询特别复杂，或是全文搜索，不能走 Sharding 后的 RouteKey，Index 表也无法满足需求

- 查询的结果也不仅仅是单表的数据而是比较丰富的数据，查询数据库需要查询多个表多次

索引型数据库和文档型数据库的底层存储结构是截然不同的，虽然现在有很多人使用 ES 来完全替代 Mongodb，但是个人觉得 ES 适合存比 Mongodb 更大的一个数据量，分布式不利用起来发挥不了 ES，Mongodb 还是适合中型数据非 Sharding 的存储。

时序型数据库

InfluxDb 是时序型数据库的代表。对于按照时间段进行 Group By 查询的话，不管是 ES 还是 MySQL 还是 Mongodb 在 API 层面当然都是支持的，但是查询效率不堪入目。因此对于诸如下面的需求首当其中可以考虑时序型数据库：

- 监控图表
- 按时间维度聚合
- 查询的时间维度可以跨度很长
- 需要定期归档

如果使用传统方案的话，我们往往会以固定的时间维度来聚合保存数据，如果我们要查 1 小时和 1 年的维度，都使用 5 秒的聚合粒度显然不合适，我们需要在写入数据到时候针对不同的粒度进行聚合，需要一定的工作量，使用时间序列数据库可以少一些这样的烦恼。而且 InfluxDb 之类的数据库的性能是非常高的，写入数据的性能堪比 Redis，单节点甚至可以承受十万指标的写入，基本可以满足大部分应用场景的需求。对于一些业务指标的监控，业务事件的打点，业务数据的时间维度聚合，我们完全可以考虑引入专门的时序型数据库。

综上所述，这里的架构图只是体现了几个重要思想：

1. 使用专门的服务来做数据的写入和读取，方便进行路由。
2. 合理规划好 Sharding 的方式，以及想好 RDBMS 在 Sharding 后的全套查询方案。
3. 数据的写入区分主要数据源的同步写入和次要数据源的异步写入，让主流程更快。
4. 合理利用不同数据源的特性，组合使用发挥所长，避免所短。
5. 数据的加工可以是一个层级的关系，可以由专门业务中间件来进行数据加工。

6. RDBMS 以外的数据库如果打算作为主核心存储引擎的话千万慎重思考。
7. 采用丰富的数据源意味着维护成本的增多，数据不同步的问题在所难免，需要考虑一下我们是否可以接受一定层度的数据不一致。