海量数据业务有哪些优化手段?

原创 TomGE 微观技术 2021-04-13 08:00

收录于合集
#架构设计

57个

互联网时代，亿级用户各种网络行为产生大量数据，如何解决海量数据存储？如何高性能读写？解决思路有哪些，本文列举了常用的解决方案：

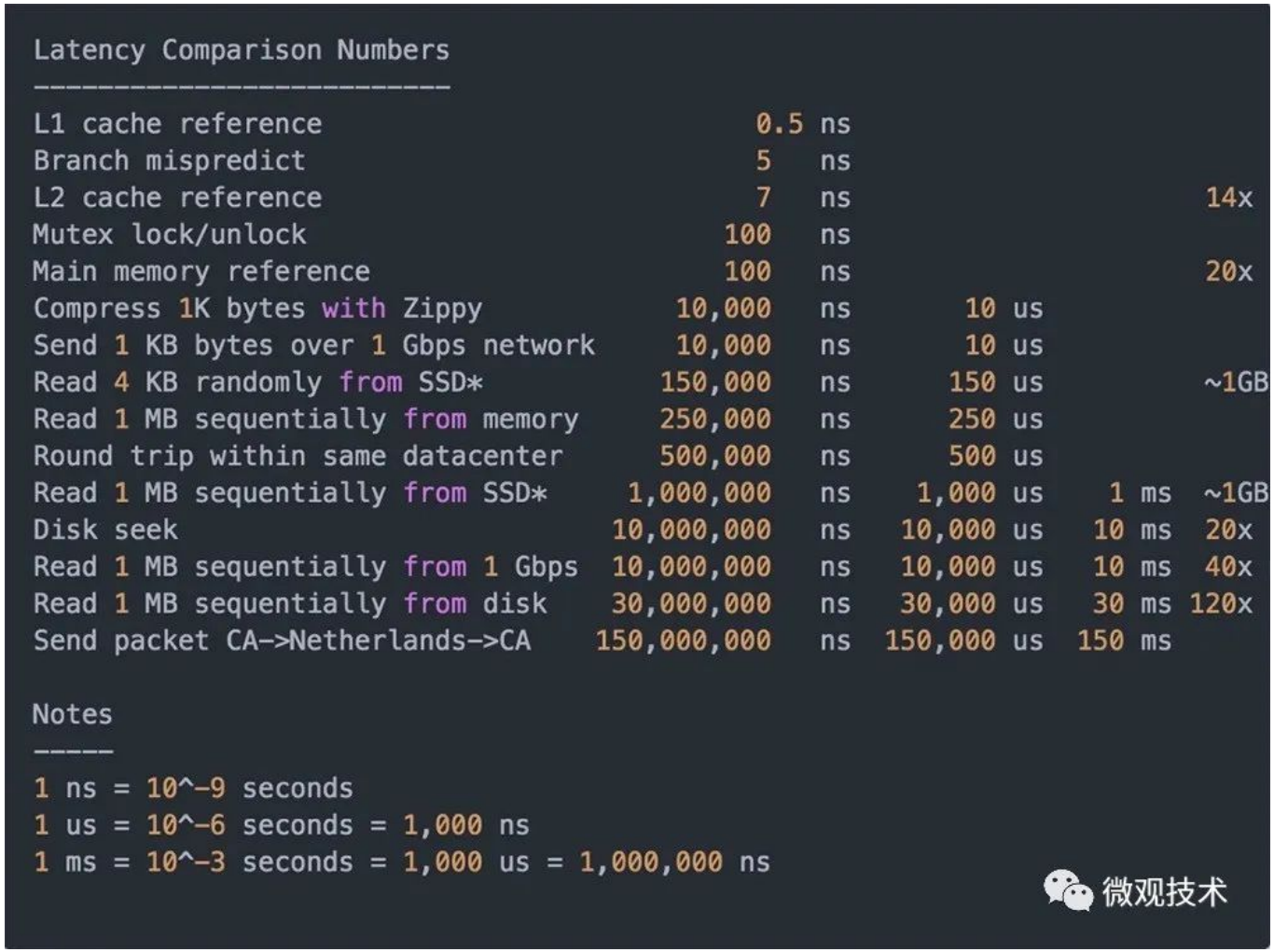
- 缓存加速
- 读写分离
- 垂直拆分
- 分库分表
- 冷热数据分离
- ES助力复杂搜索
- NoSQL
- NewSQL

🌴缓存加速

缓存就是为了弥补存储系统在这些复杂业务场景下的不足，其基本原理是将可能重复使用的数据放到内存中，一次生成、多次使用，避免每次使用都去访问存储系统。

缓存能够带来性能的大幅提升，以 Memcache 为例，单台 Memcache 服务器简单的 key-value 查询能够达到 TPS 50000 以上；Redis性能数据是10W+ QPS

为什么缓存的速度那么快？



从上图中发现，同机房两台服务器跑个来回，再从内存中顺序读取1M数据，共耗时0.75ms。如果从硬盘读取，做一次磁盘寻址需要10ms，再从磁盘里顺序读取1M数据需要30ms。可见，使用内存缓存性能上提高多个数量级，同时也能支持更高的并发量。

常见的缓存分为本地缓存和分布式缓存，区别在与是否要走网络通讯。

本地缓存是部署在应用服务器中，而我们应用服务器通常会部署多台，当数据更新时，我们不能确定哪台服务器本地中了缓存，更新或者删除所有服务器的缓存不是一个好的选择，所以我们通常会等待缓存过期。因此，这种缓存的有效期很短，通常为分钟或者秒级别，以避免返回前端脏数据。

相反，分布式缓存采用集群化管理，支持水平扩容，并提供客户端路由数据，数据一致性维护更好。虽然有不到 1ms的网络开销，但比起其优势，这点损耗微不足道。

> 注意：在引入缓存后，如果数据库的访问量依旧很大，我们可以考虑对数据库读写分离，通过多个读库分摊压力。

🌴 读写分离

互联网业务有个重要特性，不知道大家有没有发现？大多数业务都是读多写少，如：刷朋友圈的请求量肯定比发朋友圈的量大，淘宝上一个商品的浏览量也肯定远大于它的下单量。

那么数据库如何抵抗更高的查询请求？那么首先你需要把读写流量区分开，因为这样才方便针对读流量做单独的扩展，这就是我们所说的主从读写分离。

读写分离定义

> 每次写数据时会同步多份到其它的存储系统，生成多个备份，当用户读取数据时直接从备份存储系统获取数据。

应用场景：

- 读多写少
- 数据量较大
- 数据查询频率很高，且对性能要求很高

实现思路：

1、由于数据存在备份，甚至是多份备份。那么如何来实现数据备份？

- 直接方式是修改业务代码，这也是新手常用的方式。在写入主库后，同步更新备库。

> 缺点：如果备库较多，会同步调用多次，如果备库做了调整，业务代码也要跟着修改。优点：实时性好，所见即所得。

- 监控数据库 binlog 日志，（如：引入canal组件），由数据同步中心，将变更数据同步到备库中。该方式实现了业务代码解耦，扩展性较好，也是实际工作推崇的技术方案。

> 缺点：数据同步需要花费一定时间，如果这期间查询备库，查询到的是旧数据，此类业务场景需要特别注意。

2、数据备份有哪些存储介质？

- mysql。关系型数据库，容易上手
- Elasticsearch。可以定制索引结构，满足多样化复杂的业务查询。另外采用分片结构，可以满足较大量数据存储。
- MongoDB
- HBase

> 市面的开源框架较多，要注意技术选型。

3、查询数据如何实现?

这个没有什么可以讲得，以上的中间件开源社区都有封装好的API，直接调用即可。但要注意一点。 数据查询时，如果还没有备份完成怎么办?

- 一种方案，不允许用户查询，用户体验较差，也不容易控制。鬼知道有没有同步完。
- 对实时性要求不高的查询，选择走备库，但页面要做好提示引导。比如付款动作，一般会有一个中间页，提示用户付款成功。一般不会直接跳到订单详情页。
- 对实时性要求非常高的查询，走主库。比如：新用户注册，立即登录。

🌴垂直拆分

垂直拆分是指按照业务功能拆分，业务表分布在不同的数据库上，这样也就将数据或者说压力分担到不同的库上面 。比如电商系统会拆分出会员、商品、交易、类目、营销、搜索等业务库，分别归属到不用的技术团队维护。

优势：

- 职责单一，业务清晰
- 便于维护，易扩展
- 通常会独立部署，独享服务器资源，数据库访问性能会有很大提升
- 耦合性低，降低不同应用间故障干扰

缺点：

- 形成跨库事务，需要引入分布式事务解决方案，提高整个应用的复杂度。
- “木桶效应”，任何一个短板有可能影响整个系统
- 不用业务表之间不能 `join` ，只能通过服务间接口调用，在应用层做数据组装，提高了复杂度

🌴分库分表

注意：数据库垂直拆分后，遇到单机数据库性能瓶颈，我们可以考虑分表。

分表又可以细分为 垂直分表 和 水平分表 两种形式。

1、垂直分表

数据表垂直拆分就是纵向地把一张表中的列拆分到多个表，表由“宽”变“窄”，简单来讲，就是将大表拆成多张小表，一般会遵循以下几个原则：

- 冷热分离，把常用的列放在一个表，不常用的放在一个表。
- 字段更新、查询频次拆分
- 大字段列独立存放
- 关系紧密的列放在一起

2、水平分表

表结构维持不变，对数据行进行切分，将表中的某些行切分到一张表中，而另外的某些行又切分到其他的表中，也就是说拆分后数据集的并集等于拆分前的数据集。

分库分表技术点：

- SQl组合。因为是逻辑表名，需要按分表键计算对应的物理表编号，根据逻辑重新组装动态的SQL
- 数据库路由。如果采用分库，需要根据逻辑的分表编号计算数据库的编号
- 结果合并。如果查询没有传入指定的分表键，会全库执行，此时需要将结果合并再输出。

目前市面有很多的开源框架，大致分为两种模式：

- Proxy模式。SQL 组合、数据库路由、执行结果合并等功能全部存放在一个代理服务中，业务方可以当做。
  - 优点：支持多种语言。升级方便。对业务代码无侵入。
  - 缺点：额外引入一个中间件，容易形成流量瓶颈，安全风险较高，有运维成本
- Client 模式。常见是 `sharding-jdbc` ，业务端系统只需要引入一个jar包即可，按照规范配置路由规则。jar 中处理 SQL 组合、数据库路由、执行结果合并等相关功能。
  - 优点：简单、轻便。不存在流量瓶颈，减少运维成本
  - 缺点：单语言，升级不方便。

实现思路：

1、如何选择分表键。

数据尽量均匀分布在不同表或库、跨库查询操作尽可能少、这个字段的值不会变。比如电商订单采用user_id。

2、分片策略。

根据范围分片、根据 hash 值分片、根据 hash 值及范围混合分片

3、如何编写业务代码。结合具体的业务实现。

4、历史数据迁移

- 增量数据监听 binlog，然后通过 canal 通知迁移程序开始增量数据迁移
- 开启任务，全量数据迁移
- 开启双写，并关闭增量迁移任务
- 读业务切换到新库
- 线上运行一段时间，确认没有问题后，下线老库的写操作

> 有一种说法：数据量大，就分表；并发高，就分库

最后：在实际的业务开发中，要做好数据量的增长预测，做好技术方案选型。另外，在引入分表方案后，要考虑数据倾斜问题，这个跟分表键有很大关系，避免数据分布不均衡影响系统性能。

🌴冷热数据分离

根据二八定律，系统绝大部分的性能开销花在20%的业务。数据也不例外，从数据的使用频率来看，经常被业务访问的数据称为热点数据；反之，称之为冷数据。

在了解的数据的冷、热特性后，便可以指导我们做一些有针对性的性能优化。这里面有业务层面的优化，也有技术层面的优化。比如：电商网站，一般只能查询3个月内的订单，如果你想看看3个月前的订单，需要访问历史订单页面。

实现思路：

1、冷热数据区分的标准是什么？要结合业务思考，可能要找产品同学一块讨论才能做决策，切记不要拍脑袋。以电商订单为例：

- 方案一：以“下单时间”为标准，将3 个月前的订单数据当作冷数据，3 个月内的当作热数据。
- 方案二：根据“订单状态”字段来区分，已完结的订单当作冷数据，未完结的订单当作热数据。
- 方案三：组合方式，把下单时间 > 3 个月且状态为“已完结”的订单标识为冷数据，其他的当作热数据。

2、如何触发冷热数据的分离

- 方案一：直接修改业务代码，每次业务请求触发冷热数据判断，根据结果路由到对应的冷数据表或热数据表。缺点：如果判断标准是 时间维度 ，数据过期了无法主动感知。
- 方案二：如果觉得修改业务代码，耦合性高，不易于后期维护。可以通过监听数据库变更日志 binlog 方式来触发

- 方案三：常用的手段是跑定时任务，一般是选择凌晨系统压力小的时候，通过跑批任务，将满足条件的冷数据迁移到其他存储介质。在途业务表中只留下来少量的热点数据。

3、如何实现冷热数据分离，过程大概分为三步：

- 判断数据是冷、还是热
- 将冷数据插入冷数据表中
- 然后，从原来的热库中删除迁移的数据

4、如何使用冷热数据

- 方案一：界面设计时会有选项区分，如上面举例的电商订单
- 方案二：直接在业务代码里区分。

🌴 ES助力复杂搜索

ES是基于索引结构，无法像mysql那样使用join语句。所以我们在构建索引时需要将主表记录及关联表打平，整合到一条记录中。以倒排索引作为核心技术原理，为你提供了分布式的全文搜索服务。

mysql与es的概念关系映射

mysql	**ES**
数据库	索引 index
表	Type
行	Document
列	Field

ES分页查询流程：

- 协调节点首先把分页查询请求分发给所有分片
- 每个分片在本地查询一个结果集列表（包含 Document id和搜索分数），返回（from+size）条记录。 特别注意：这一步返回的只是主键id
- 协调节点拿到所有分片的返回数据，按分数全局排序，并截取一页大小的数据
- 协调节点根据结果集里的Document id 向所有的分片查询完整的Document，然后协调节点将结果返回给客户端。

在读取操作流程中，Elasticsearch 集群实际上需要给协调节点返回 shards number *（from + size) 条数据，然后在单机上进行排序，最后返回给客户端这个 size 大小的数据。

随着分页的深度增加，性能会越来越差，为了避免这个问题，ES有个 `max_result_window` 配置，默认值10000，超过这个大小，ES返回错误。

如果用户确实有深度翻页的需求，可以采用 `search_after` 解决，比如 `id>20000 limit 10` 。缺点：无法实现跳页。

🌴 NoSQL

NoSQL 数据库放弃了与分布式环境相悖的 **ACID** 事务，提供了另一种聚合数据模型，从而具有可伸缩性的非关系数据库。

NoSQL 数据库分为五类：

1、KV 数据库，通常基于哈希表实现，性能非常好。其中 Value 的类型通常由应用层代码决定。常用的如 Redis，value支持 String、List、Map、Set、Zset等复合结构。

2、文档型数据库，如：MongoDB、CouchDB ，这种数据库的特点是 Schema Free（模式自由），数据表中的字段可以任意扩展，比如说电商系统中的商品有非常多的字段，并且不同品类的商品的字段也都不尽相同，使用关系型数据库就需要不断增加字段支持，而用文档型数据库就简单很多了。

3、列式数据库，比如 Hbase、Cassandra。列式数据库基于 Key 来映射行，再通过列名进行二级映射，同时它基于列来安排存储的拓扑结构，这样当仅读写大量行中某个列时，操作的数据节点、磁盘非常集中，磁盘 IO、网络 IO 都会少很多。列式数据库的应用场景非常有针对性，比如博客文章标签的行数很多，但在做数据分析时往往只读取标签列，这就很适合使用列式数据库。再比如，通过倒排索引实现了全文检索的 ElasticSearch，就适合使用列式存储存放 Doc Values，这样做排序、聚合时非常高效。

4、图数据库，在社交关系、知识图谱等场景中，携带各种属性的边可以表示节点间的关系，由于节点的关系数量多，而且非常容易变化，所以关系数据库的实现成本很高，而图数据库既没有固定的数据模型，遍历关系的速度也非常快，很适合处理这类问题。

5、时序数据库，如：InfluxDB，一般用来做 `Metrics` 打点。时序数据库的优势，在于处理指标数据的聚合，并且读写效率非常高。

> 应用场景：比如对1000 万数据进行一个统计，查询最近 60 天的数据，按照 1 小时的时间粒度聚合，统计 value 列的最大值、最小值和平均值，并将统计结果绘制成曲线图。

InfluxDB 也有不足之处：

- InfluxDB 不支持数据更新操作，毕竟时间数据只能随着时间产生新数据，肯定无法对过去的数据做修改；
- 从数据结构上说，时间序列数据数据没有单一的主键标识，必须包含时间戳，数据只能和时间戳进行关联，不适合普通业务。

相比传统关系型数据库，**NoSQL** 有哪些优势：

- 弥补了传统数据库在性能方面的不足；
- 数据库变更方便，不需要更改原先的数据结构；
- 适合互联网常见的大数据量的场景；

🌴 NewSQL

New SQL 是新一代的分布式数据库，它具备原生分布式存储系统高性能、高可靠、高可用和弹性扩容的能力，同时还兼顾了传统关系型数据库的 SQL 支持。另外，它还提供了和传统关系型数据库不相上下的、真正的事务支持，具备了支撑在线交易类业务的能力。

优点：

- 完整地支持 SQL 和 ACID 事务，提供和 Old SQL 隔离级别相当的事务能力；
- 高性能、高可靠、高可用，支持水平扩容。

常见的 New SQL 数据库有：Google 的 Cloud Spanner、阿里巴巴的 OceanBase 以及开源的CockroachDB。

往期推荐

- 10分钟掌握RocketMQ的核心知识
- ShardingSphere解决海量数据分库分表
- Redis 实现分布式锁真的安全吗?
- 【小妙招】如何借助Proxy代理，提升架构扩展性
- 还在用Mybatis? Spring Data JPA 让你的开发效率提升数倍!
- 淘宝订单自动确认收货的N种实现，秒杀面试官
- DDD是如何解决复杂业务扩展问题?
- 如何设计一个高性能的秒杀系统

我们热衷于收集高并发、系统架构、微服务、消息中间件、 RPC框架、高性能缓存、搜索、分布式数据框架、分布式协同服务、分布式配置中心、中台架构、领域驱动设计、系统监控、系统稳定性等技术知识。

原创不易，如果感觉本文对您有帮助，请转发分享，点个“在看”!

收录于合集 #架构设计 57

上一篇
淘宝双11千亿交易额的系统架构演变

下一篇
【小妙招】如何借助Proxy代理，提升架构扩展性

喜欢此内容的人还喜欢

面试官问：你离职的原因是什么？如何避坑?

微观技术