《分布式数据库设计及反范式设计》

http://Weibo.com/caoz
http://web.4399.com/job

个人简介

- 非技术专家
  - 更关注产品目标，而非技术深度
  - 拿来主义，拒绝重复造轮子
  - 各种野路子招法，拒绝教条
  - 信奉简单
    - 口头禅，你要是想复杂了，你一定想错了
  - 抓大放小
    - 不苛求完美
  - 杂而不精
- 衷告
  - 技术的价值在于产品表现，而非技术本身

分布式数据库

- 基本前提
  - 索引优化
  - 应用缓存
  - 运维优化
    - 特别是i/o优化
- 简单概念
  - 数据记录数 3000万条以上
  - 每秒查询请求数 1500次以上
  - 每秒写入请求数300次以上
  - 根据具体业务不同，以上数据并非通适

分布式数据库

- 设计目标
  - 成本的可控性
    - 资源投入与业务增长比例应等于或低于线性关系
  - 业务可扩展性
    - 可以方便的通过资源投入迅速扩展业务支撑能力
  - 可维护性，安全性
    - 摈弃单点故障隐患
    - 备份与数据安全
- 面临的挑战
  - 数据量的不断增加
  - 读取请求规模的增加
  - i/o压力的增加

分布式数据库

- 主从结构
  - 优点
    - 部署简单
  - 局限性
    - i/o压力无法分布，性价比不高
    - 同步延时是躲不开的问题
    - 主数据库是典型的单点隐患，很难做成自动故障转移
  - 适合场景
    - 在线热备
    - 主作为单服应用时，从提供故障自动转移

分布式数据库

- 分库，分表
  - 优点
    - 负载分担较好
    - 不存在同步延迟
    - 拆分方法灵活
  - 局限性
    - 需要有备份和自动故障转移的方案
    - 需要应用端配合，无法完全满足关联查询的需求
- 推荐方案
  - 以分库，分表为负载和数据支撑方案
  - 以主从结构为热备和故障转移方案
  - 使用中间件作为分布式数据库的前端
    - 我们目前使用 amoeba，自己做了一些调整

分布式数据库

- 分库原则
  - 基于业务拆分
    - 易于管理，对应用端友好
    - 负载不能均分
  - 基于负载拆分
    - 负载相对可以均摊
    - 管理不方便
  - 基于安全性拆分
    - 运维优化，易于管理
      - Sync-binlog, innodb_flush_log_at_trx_commit

分布式数据库

- 建议方案及拆分范例
  - 混合策略
    - 独立的数据库实体并非是独立的数据库服务器
  - 先基于安全策略拆分出数据库实体A
    - 牺牲性能，保障安全性
  - 对剩余部分基于业务策略拆分成B,C,D
  - 对于负载较高的D 拆分成d1,d2,d3
  - 服务器可以配置为
    - A,B,C　存在于一台服务器
    - D1,D2,D3 为独立的服务器
    - 主从热备另算

分布式数据库

- 拆表方案
  - 纵向拆表
    - 案例：用户信息表
  - 等分拆表
    - 基于主键或特定索引的哈希取模
    - 优点：负载均分；缺点：未来可能遇到二次拆分问题
  - 递增拆表
    - 基于一定的规模自动拆表
    - 优点：自动维护；缺点：负载不均
  - 冷热拆表
    - 热门数据单独拆表，或历史数据单独存储
    - 适合复杂查询情况的查询优化

分布式数据库

- 总结
  - 利用拆分解决数据容量问题和负载问题
  - 利用主从解决数据安全问题和单点隐患问题
    - 通过心跳监测和虚ip指向
  - 拆分是必要的，但是不是充分的
  - 去关联化处理
    - 关联查询的拆解
    - 业务逻辑的拆解
  - 数据一致性问题
    - 通过业务逻辑，在应用层处理，而非数据库处理

反范式设计

- 去关联化
  - 强调索引，弱化外键的概念
  - 不考虑触发器及其他内部的存储过程
- 适度冗余
  - 基于展现的冗余
  - 基于查询优化的冗余
  - 基于统计优化的冗余
  - 基于i/o压力优化的冗余
- 一致性问题
  - 业务层处理
  - 可容忍性

反范式设计

- 基于展现的冗余
  - 展现内容涉及多表查询
  - 如相关联表所需内容简单且更改度不高，可做冗余字段
  - 案例：社区的消息表
    - 原始字段 fromuid,touid,msg,dateline
    - 冗余字段 fromuname,fromusex
  - 案例：游戏战报，好友列表，好友动态

反范式设计

- 基于查询的冗余
  - 场景1：查询条件，涉及多个表的字段
    - 简单情况下用冗余字段解决
    - 查询相对固定化可建立查询冗余表
  - 场景2：分表情况下查询需求不能用一个键满足
    - 需要建立冗余查询结构
    - 案例： 用户账号密码查询，用户id查询
      - 主表 uid, uname,upass, email,qq,intro,…..
      - 冗余表 uname,uid
      - 多项登录 冗余表 key,keytype,uid
    - 案例：游戏积分排行，用户积分排行
      - 基于gameid， uid的 全冗余结构

反范式设计

- 基于统计的冗余
  - 较为频繁的count, sum需求
  - 通常使用冗余字段
  - 案例：论坛的今日发帖，板块发帖数等
    - 在论坛表，板块表有冗余统计字段
    - 注意增删一致性问题
  - 案例：积分游戏实时排名
    - Select count(*) from gamescore where score>’$score’
    - 分段统计冗余结构
    - Redis 的 zset结构就是一个典型的统计冗余结构

反范式设计

- 基于i/o压力优化的冗余
  - 单次请求多次写入的情况
    - 请求频次较高，i/o压力较大
    - 存在高频读取请求，数据可靠性要求高
  - 其他可用方案
    - 数据压缩存储
    - 写入缓存队列
  - 通过冗余结构，合并为一次写入
  - 案例：游戏组队pk
  - 案例：实时统计的top refer；top url处理

反范式设计

- 总结
  - 适度冗余可以减少查询请求
  - 适度冗余可以解决分表带来的索引查询问题
  - 适度冗余可以解决统计类负载较高查询问题
  - 适度冗余可以减少i/o请求频次，提高i/o支撑能力
- 反范式设计
  - 减少关联性
  - 通过冗余解决分布式的衍生问题
  - 一致性问题会相当突出，需要业务层面把控
    - 部分一致性问题可容忍，案例：用户级别的展现
  - 寻求平衡而非极端

欢迎交流

新浪微博: @caoz　　邮箱:Caoz@4399.com

百度文库搜索　“mysql性能优化教程”

4399诚邀各路高手交流切磋,加盟.

ArchSummit

中国·深圳 2012.08

INTERNATIONAL ARCHITECT SUMMIT

全 球 架 构 师 峰 会

详情请访问：architectsummit.com

- 3天 • 6场主题演讲
- 3场圆桌论坛 • 9场专题会议
- 国内外30余家IT、互联网公司的50多位来自一线的讲师齐聚一堂

主办方：InfoQ 战略合作伙伴：Tencent 腾讯 特别支持：

http://architectsummit.com


QCon
杭州站 · 2012年10月25日~27日
www.qconhangzhou.com（6月启动）
QCon北京站官方网站和资料下载
www.qconbeijing.com