# 阿里分布式数据库服务原理与实践

沈询

# 自我介绍


- 花名 沈询
- DRDS 目前负责架构设计
- 阿里分布式数据层（TDDL）负责人
- 参与过阿里集团大部分的Oracle到MySQL的迁移工作
- 在分布式存储领域经验比较丰富

# Agenda


- DRDS 介绍
- 在线数据迁移原理与应用
- 在线应用数据拆分经验

# DRDS介绍

# DRDS介绍


- 起源
- 核心价值
- 应用场景
- 架构与原理

# DRDS介绍-起源


- 起源
  - DRDS 脱胎于 alibaba的cobra 分布式数据库引擎
    - 06年上线使用
    - 在alibaba有80+应用在使用，目前已经开源
    - DRDS的80%的代码出自cobra proxy
      - Sql解析器
      - 执行流程
      - 配置

# DRDS介绍-起源


- 起源
  - DRDS吸收了taobao TDDL分布式数据库引擎的大量优秀经验和解决方案
    - 08年上线使用
    - 目前在使用的应用900+
    - 大量实际应用解决方案支持
      - 分布式join
      - 分布式aggregation (group sum max min)
      - 异步索引构建
      - Auto sharding ,自动扩容缩容

# DRDS介绍-起源


- 起源
  - DRDS专门针对外部用户进行了配置的重新设计
    - 简化了配置操作规范与流程
    - 尽可能使得应用像操作一个数据库一样的操作DRDS
    - 用户的专业化指导

# DRDS介绍-核心价值


- 核心价值
  - mysql 兼容性
    - 95%以上的mysql查询可以直接在drds上运行
    - 在大部分情况下，可以把drds当做一个单机mysql来使用。
    - 适当的做出了功能上的限制，以保证用户可以一直能够享受到线性的水平扩展能力。
  - 自动数据运维
    - 把机器简单的添加到集群内就可以实现水平扩展和自动的负载均衡。
  - 管理更容易
    - 建库建表增减字段，一个命令可以搞定

# DRDS介绍-应用场景


- 应用的业务需求单机已经无法满足
  - 一个RDS数据库的最大实例也无法满足用户的需求
    - 容量瓶颈
    - 事务数瓶颈
    - 读取瓶颈

# DRDS介绍-应用场景

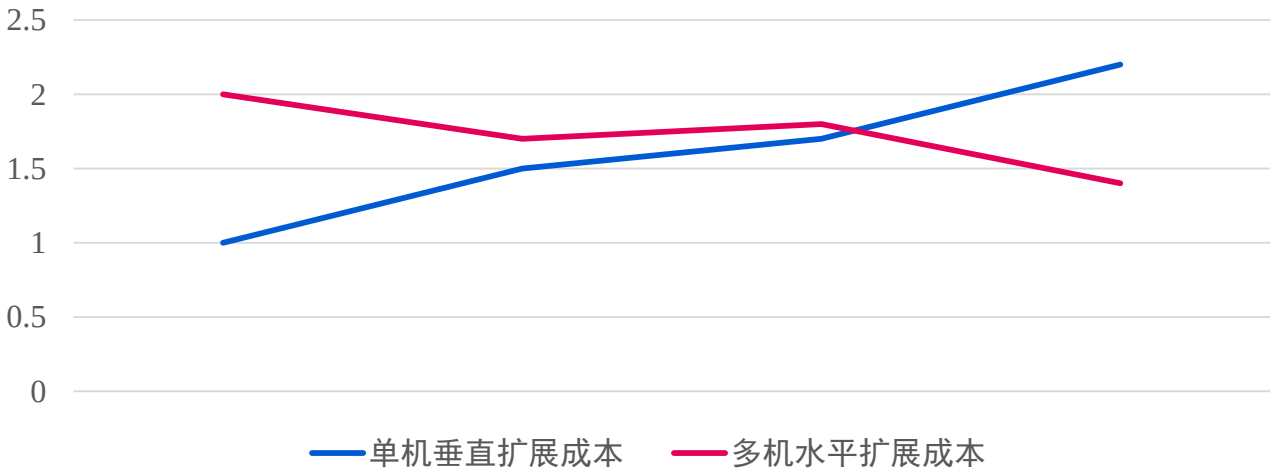

- Scale up（单机垂直扩展）
  - 购买或更换更高端的机器-oracle rac /高端存储盘柜
- 优势
  - 业务不用修改代码
  - 业务改动小
- 劣势
  - 架构被把持，更换存储成本巨大
  - 定价权在数据库软件厂商
  - 把定时炸弹的时间往后拨了一些时间，最终还是会炸的

# DRDS介绍-应用场景

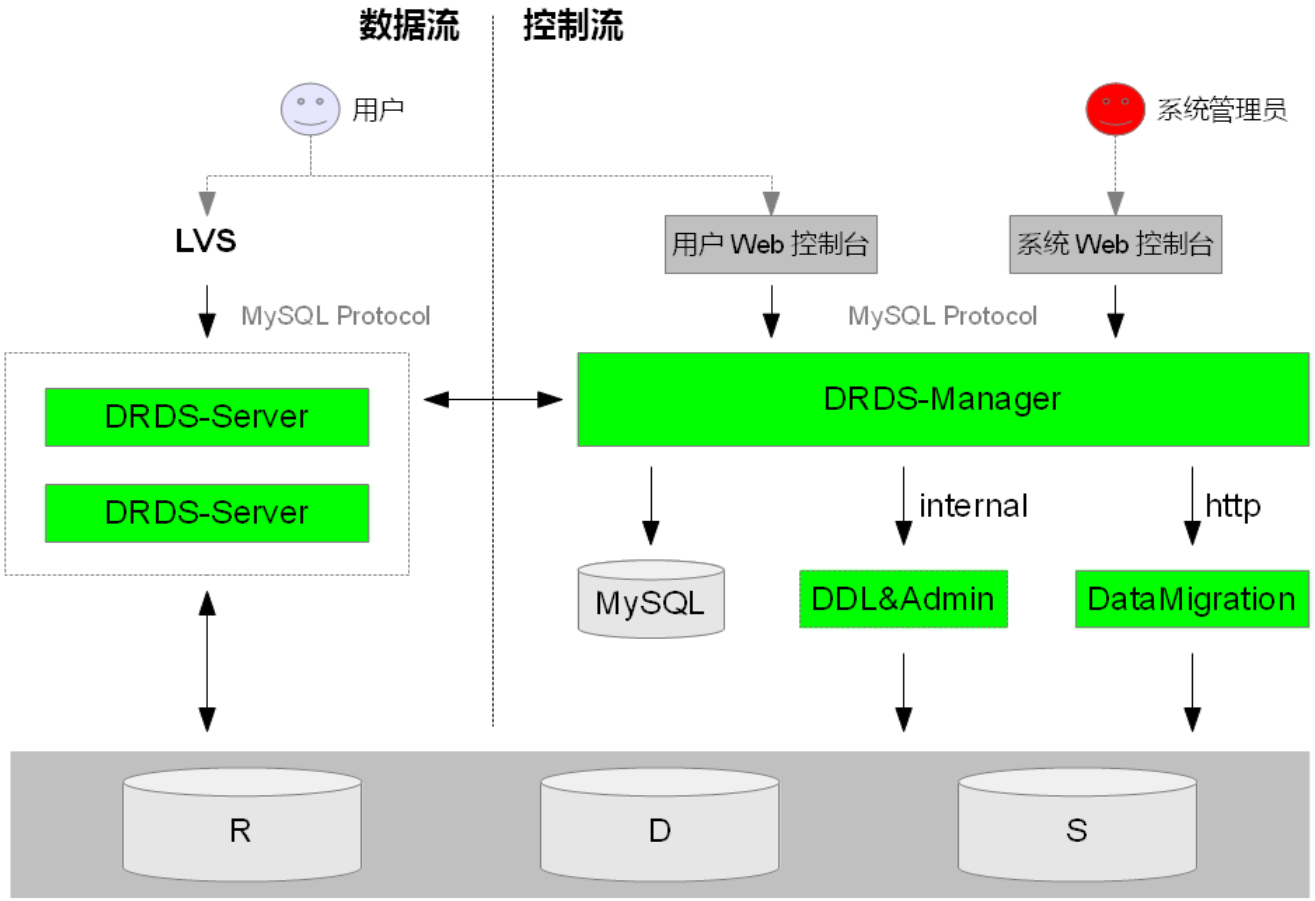

- Scale out（多机水平扩展）
  - 使用廉价数据库阵列来满足用户需求--DRDS
  - 优势
    - 更轻量的使用数据库，未来更换的成本小
    - 一次重构，以后基本再无需担心系统瓶颈
  - 劣势
    - 重构需要付出成本
    - 分布式环境下一些查询会被限制不允许执行
    - 完成相同功能需要比单机扩展付出更多成本

# DRDS介绍-应用场景


- 理想状态
  - Scale out 与scale up结合
    - 让系统架构具备scale out的能力
    - 尽可能提升单机利用率
  - 但不要过早过度设计



# DRDS介绍-架构与原理


系统架构图

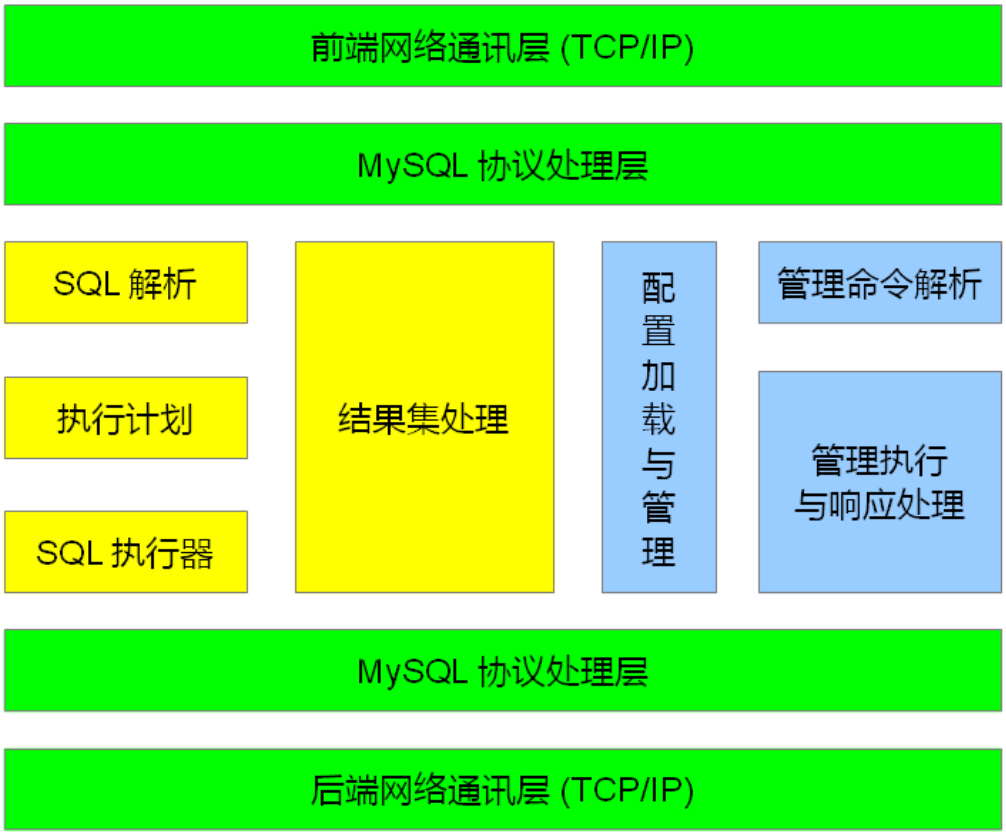
# DRDS介绍-架构与原理


- DRDS-Server
  - 直接为应用或者用户提供基于MySQL协议的数据服务，是整个系统提供服务的核心部分，数据服务以LVS集群的方式对外提供。
- DRDS-Manager
  - 为整个系统的各个子系统提供管理、控制和协调工作，并对相关配置进行持久化；该系统目前以主备的模式提供高可用服务。
- 用户管理Web控制台
  - 用户管理控制台是用户参与系统管理的入口，用户可以在上面创建表、规则、修改表结构、执行数据迁移和扩容工作等，是系统面向用户的控制台。

# DRDS介绍-架构与原理


- 系统管理Web控制台
  - WebServer，系统管理控制台是运维与运营方参与整个系统管理和监控的入口，使用方可以查看系统运行状况、监控系统关键指标等，是系统面向管理的控制台。 DRDS-Manager
- DataMigration
  - 支持由用户触发的数据迁移和扩容操作，系统采用全量+基于binlog增量的方式工作。
- RDS实例群
  - 基于MySQL的数据库实例，可以是基于现有proxy的，也可以直接基于MySQL实例的。

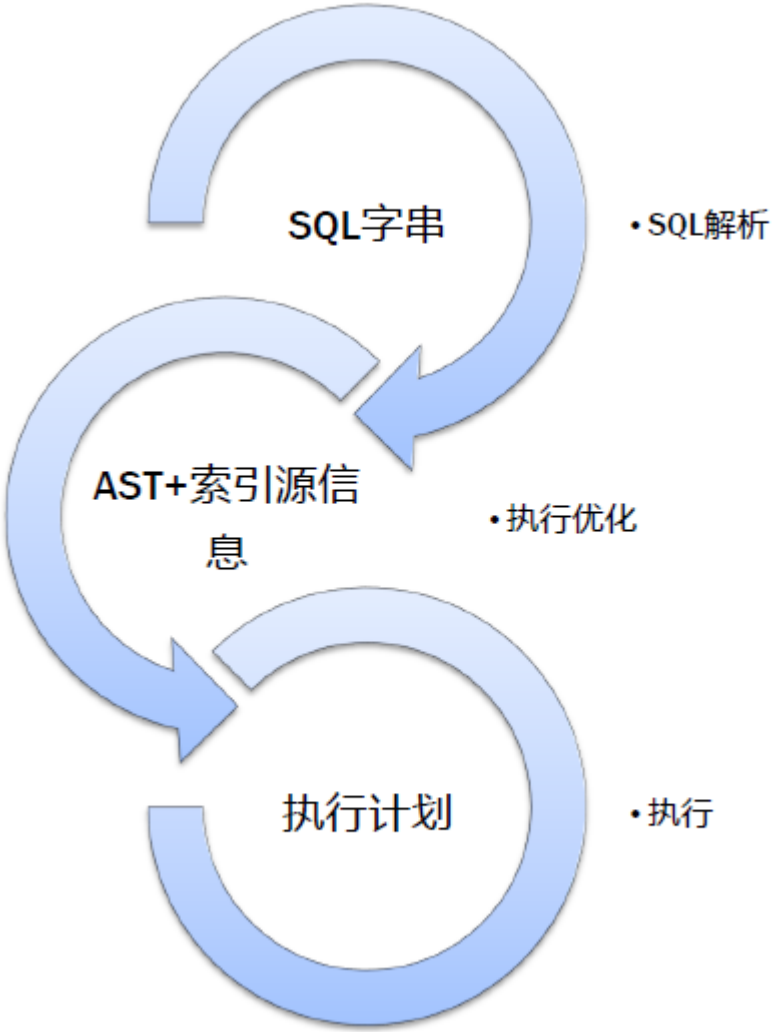
# DRDS介绍-架构与原理


模块架构图

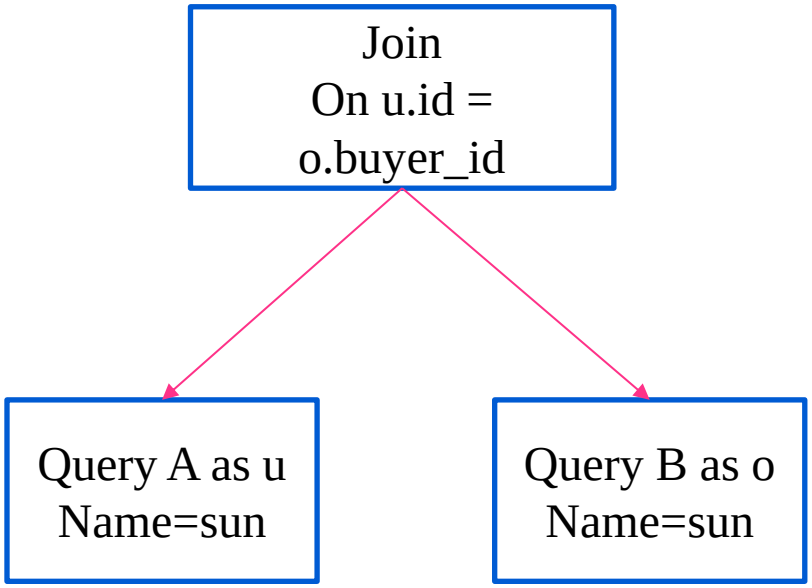
# DRDS介绍-架构与原理


- 流程
  - AST
    - 抽象语法树
    - 标记SQL的组成方式
  - 执行计划
    - 告知执行器如何高效的利用K-V



# DRDS介绍-架构与原理

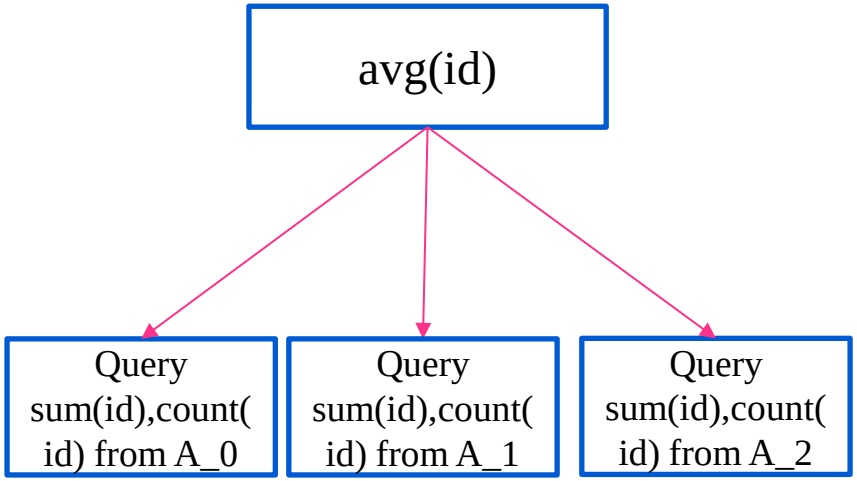

- Join的执行计划
  - 表A 在机器mA, 表B在机器mB
  - select * from A u join B o on u.id = o.buyer_id where u.name='sun‘




**Join**
leftColumns:[U.ID]
rightColumns:[O.BUYER_ID]
type:inner join
strategy:INDEX_NEST_LOOP
executeOn:mA
**left:**
select u.name,u.address,u.id from A where name = sun
**right:**
select b.id,buyer_id,seller_id from B

# DRDS介绍-架构与原理

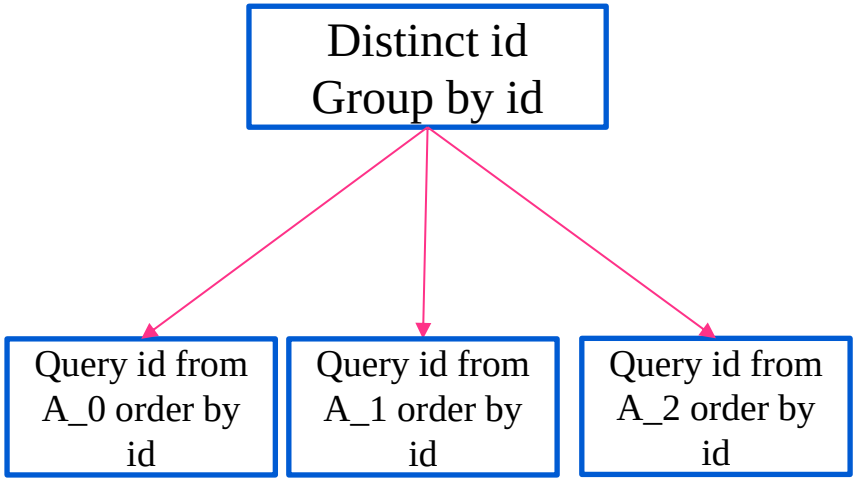

- 全表avg的执行计划
  - 表A 分库分表3个
  - select avg(id) from A




**Merge**
**avg（id)**
**subQuery**
Q1:select count(id),sum(id) A_0
Q2:select count(id),sum(id) A_1
Q3:select count(id),sum(id) A_2

# DRDS介绍-架构与原理


- 全表distinct groupby的执行计划
  - 表A 分库分表3个
  - Select distinct id from A group by A




**Merge**

**distinct id , group by id**

**subQuery**

Q1:select id from A_0 order by id

Q2:select id from A_1 order by id

Q2:select id from A_2 order by id

# DRDS介绍-小结


- 起源
  - Alibaba cobra + taobao TDDL + 面向终端用户的运维体系
- 应用场景
  - 单个数据库不足以满足用户的需要
- 核心价值
  - 用户体验基本与mysql一致，有适当限制
- 架构与原理
  - 使用了proxy架构

# 在线数据迁移原理与应用

# 在线数据迁移原理与应用


- 核心价值与目标场景
- 基本原理
- 操作方法

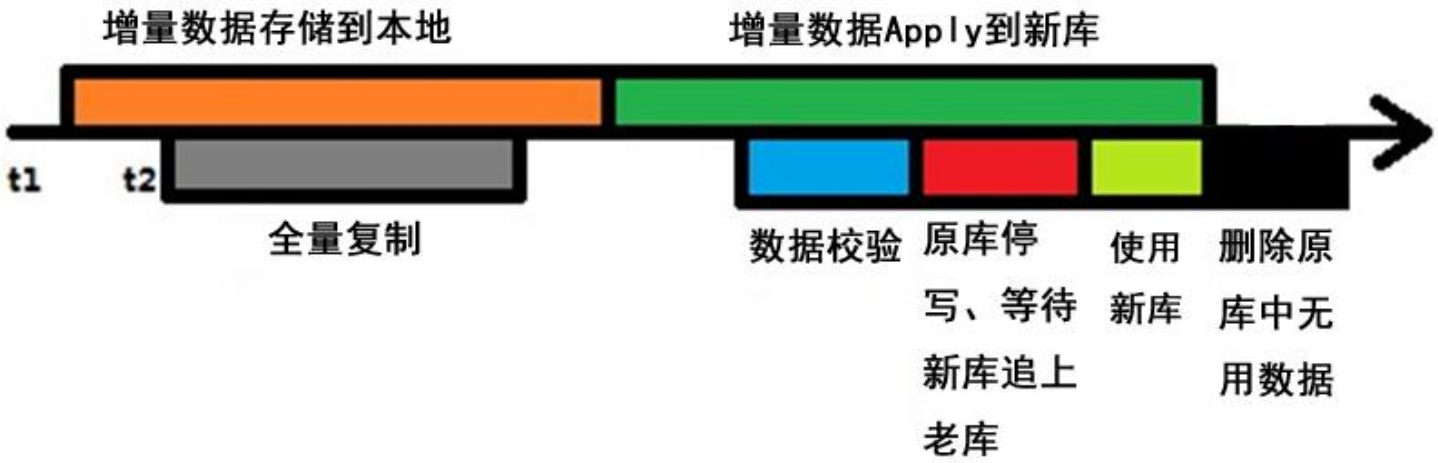
# 核心价值与目标场景


- 用户可以持续的将数据从一组机器复制到另外一组
  - 源机器可以不停止服务
  - 目标机器与原始机器只有很小的数据复制延迟
- 常见场景
  - 外部Oracle、mysql 快速迁移至RDS服务器内
  - 内部RDS for oracle 迁移至单机mysql
  - 内部RDS for oracle 迁移至drds
  - 自动扩容缩容

# 基本原理


SALE

- 流程


增量数据存储到本地
增量数据Apply到新库
t1
t2
全量复制
数据校验
原库停写、等待新库追上老库
使用新库
删除原库中无用数据

# 操作方法


- 准备一台机器，装好java环境
- 进行简单配置，并开启任务
- 程序会进行全量复制+增量追赶
- 提示“catch up”状态时，可以认为数据的搬迁已经完毕，并且针对原库的所有写操作也会被持续的重放到目标库
- 进行必要验证
- 停原库写几秒钟，让备库与原库一致
- 进行切换

# 在线应用数据拆分经验

# 主要限制原因分析与解决思路

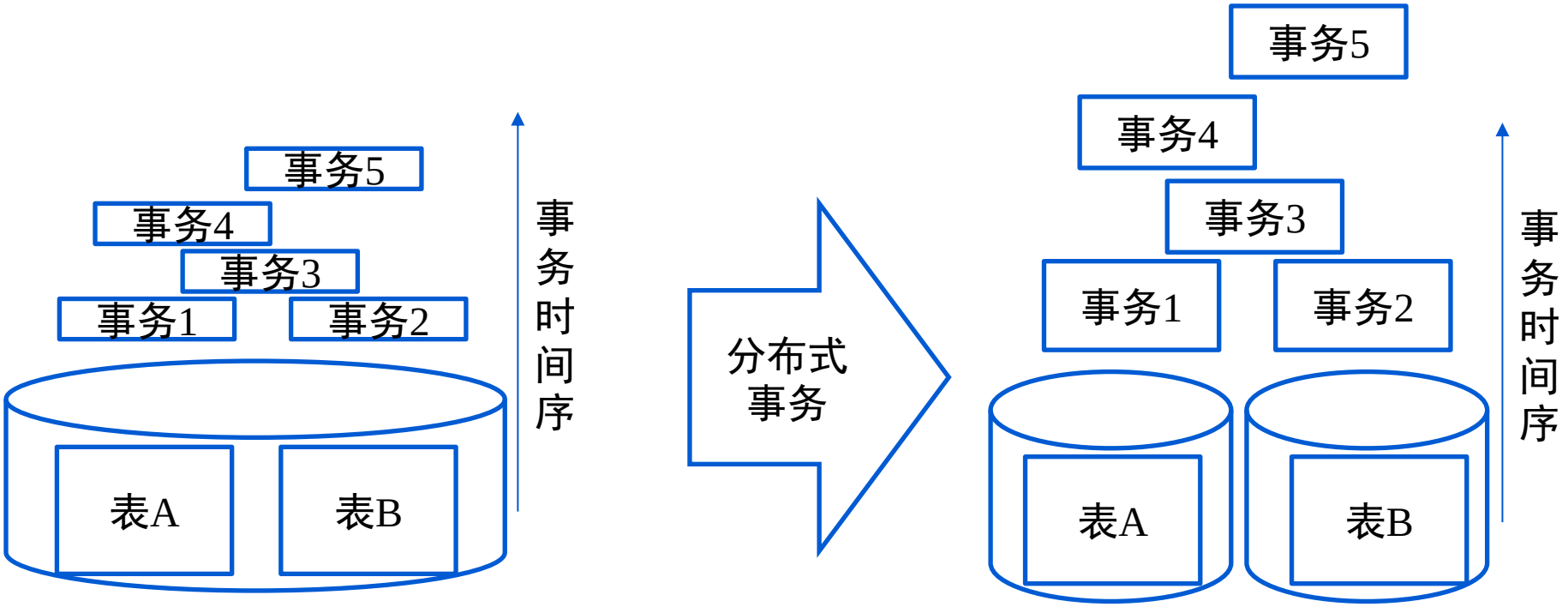

- 主要限制
- 通用的分布式解决思路和实际场景例子

# 主要限制原因分析与解决思路


- 分布式事务限制
  - 单次提交延迟从<20ms => 200ms左右
  - 记录更多的log因此需要更多资源
  - 针对同一个数据的多次更新，因为锁持有时间变长而出现tps降低
  - 如果一个数据是热点，那么系统整体tps不升反降

# 主要限制原因分析与解决思路


- 分布式事务限制



# 主要限制原因分析与解决思路


- 分布式事务的一般性解决思路
  - 半事务-减钱加钱模型
  - Bob 给smith 100块
    - Begin transaction
    - Bob has 100?
    - Bob -100
    - Smith +100
    - Transaction commit

一个事务
事务没有结束的时候，下一个更新不能进行

# 主要限制原因分析与解决思路


- 分布式事务的一般性解决思路
  - 支付宝 账务支付场景
  - 淘宝交易成交场景
  - Ebay paypal 账务支付场景

# 主要限制原因分析与解决思路


- 分布式事务的一般性解决思路
  - 半事务-减钱加钱模型
  - Bob 给smith 100块
    - Begin transaction
    - Bob has 100?
    - Bob -100

一个事务

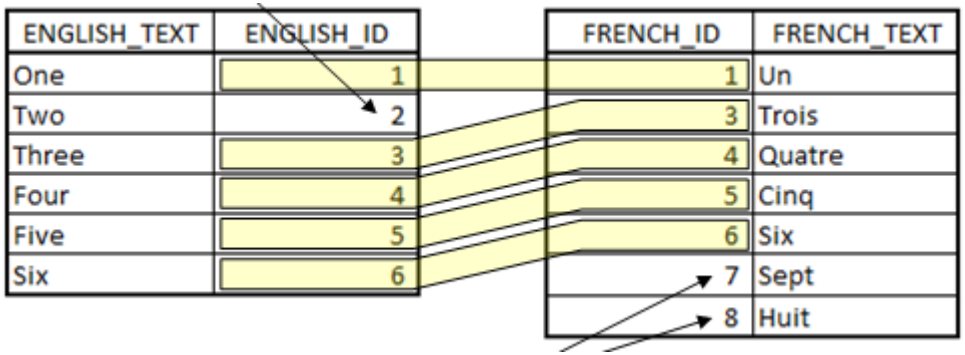
异步消息

    - Smith +100
    - Transaction commit

另个事务

# 主要限制原因分析与解决思路


- 分布式join限制 == 基于内存join->基于网络的join
  - 千兆网卡吞吐量100mb/s
  - 内存吞吐量800mb/s
  - 通过网络传输，额外的解码和编码操作开销
  - 额外的一致性读问题



# DRDS介绍-应用场景


- 主要限制原因分析与解决思路
  - 分布式join常见解决方案
  - 核心思路就是让join尽可能发生在本机
    - 小表广播
      - 如果一个小表与其他切分后大表放在一起，则将小表广播到所有服务器上
      - 这样就可以进行本地join。
    - 相同切分条件join
      - 用户，交易，评价，都按照相同的方法切分
      - 一个用户的数据物理上就在一个库，因此可以直接本地join

# 主要限制原因分析与解决思路


- 数据库的索引

| Primary Key: id | User_id | Name |
|---|---|---|
| 0 | 0 | 袜子 |
| 1 | 0 | 鞋子 |
| 2 | 1 | 计算机 |
| 3 | 3 | 电池 |

Select * from tab where user_id = ?

| User_id | id |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 2 |
| 3 | 3 |

二级索引

# 主要限制原因分析与解决思路


- 主要限制原因分析与解决思路
  - 网络延迟导致索引读一致性开销变大
    - 一致性的本质就是让快的等慢的
    - 单机索引读一致性与现在的数据库一致性一致
    - 多机索引读一致性
      - 方案1，当所有的数据节点上的索引都成功，则返回成功。
        - » 代价是更多的写入延迟 <20ms -> 200ms+
      - 方案2，解绑异步，先做完的先返回，后做完的后返回，相互不影响。
        - » 代价是没有系统级别的方案来保证索引读一致性，用户可能按照某个条件查询到的记录不是最新的

# 主要限制原因分析与解决思路


- 卖家买家问题，使用数据增量复制的方式冗余数据进行查询。这种冗余从本质来说，就是索引。

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 0 | 0 | 1 | 床上用品 |
| 1 | 0 | 2 | 路上用品 |
| 2 | 0 | 3 | 销售路由器 |
| 3 | 0 | 4 | 中文书籍 |
| 4 | 0 | 5 | 电脑 |
| 5 | 1 | 0 | ipad |
| 6 | 2 | 0 | 笔记本 |
| 7 | 3 | 0 | 铅笔 |
| 8 | 4 | 0 | 桌面 |

# 主要限制原因分析与解决思路


## buyerID % 4

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 0 | 0 | 1 | 床上用品 |
| 1 | 0 | 2 | 路上用品 |
| 2 | 0 | 3 | 销售路由器 |
| 3 | 0 | 4 | 中文书籍 |
| 4 | 0 | 5 | 电脑 |
| 8 | 4 | 0 | 桌面 |

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 5 | 1 | 0 | ipad |

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 6 | 2 | 0 | 笔记本 |

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 7 | 3 | 0 | 铅笔 |

异构复制

## sellerID % 4

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 5 | 1 | 0 | ipad |
| 6 | 2 | 0 | 笔记本 |
| 7 | 3 | 0 | 铅笔 |
| 8 | 4 | 0 | 桌面 |
| 3 | 0 | 4 | 中文书籍 |

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 0 | 0 | 1 | 床上用品 |
| 4 | 0 | 5 | 电脑 |

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 1 | 0 | 2 | 路上用品 |

| bizOrderID | buyerID | sellerID | content |
|---|---|---|---|
| 2 | 0 | 3 | 销售路由器 |

# TDDL最佳实践


- 尽一切可能利用单机资源
  - 单机事务
  - 单机join
- 好的存储模型，就是尽可能多的做到以下几点:
  - 尽可能走内存
  - 尽可能将一次要查询到的数据物理的放在一起
  - 通过合理的数据冗余，减少走网络的次数
  - 合理并行提升响应时间
  - 读取数据瓶颈，可以通过加slave节点解决
  - 写入瓶颈，用规则sharding和扩容来解决

# 主要限制原因分析与解决思路-小结


- 主要限制
  - 分布式事务
  - 分布式join
  - 分布式索引一致性
- 比较通用的分布式解决思路和实际场景例子
  - 针对事务: 半事务
  - 针对分布式join : 小表复制/同条件join切分
  - 针对分布式索引一致性：卖家买家问题

# 小结

# 小结


- DRDS 介绍
- 在线数据迁移原理与应用
- 在线应用数据拆分经验