淘宝去O,所见，所闻，所感

淘宝-通用产品-JAVA中间件
王晶昱(花名：沈询)

TAOBAOJM

ADC
Alibaba Developer Conference

自我介绍

- 沈询
  - 4年淘宝小二生涯
  - 进公司第二件事情就是参与数据层
  - 几乎支持了淘宝所有去o项目
  - **TDDL owner**

引

- 一次关于“失败”的分享
  - 讲讲历程
  - 谈谈我们所经历过的失败
  - 数据被隐去了 见谅
- 模式：
  - 黑色是成功点
  - 红色是失败点

去O的原因回溯

- 成长
  - 数据，访问量指数级上涨
    - TPS，数据量 一年翻一翻
  - 新上的业务越来越多
- 成本
  - 以小型机+高端存储存放一些不那么重要的数据，显然不够经济。
  - 业务量一年翻一翻，小型机也一年翻一翻？

AGENDA

- 探索阶段
- 全面推广阶段
- 收官阶段

探索阶段

TAOBAOJM

探索阶段

- 接到需求时候的纠结：
  - 需求
    - 屏蔽下层各类存储
    - 对应用层完全透明
  - 纠结
    - 如何实现join?
    - 多机事务怎么办？
    - 异地机房如何支持？

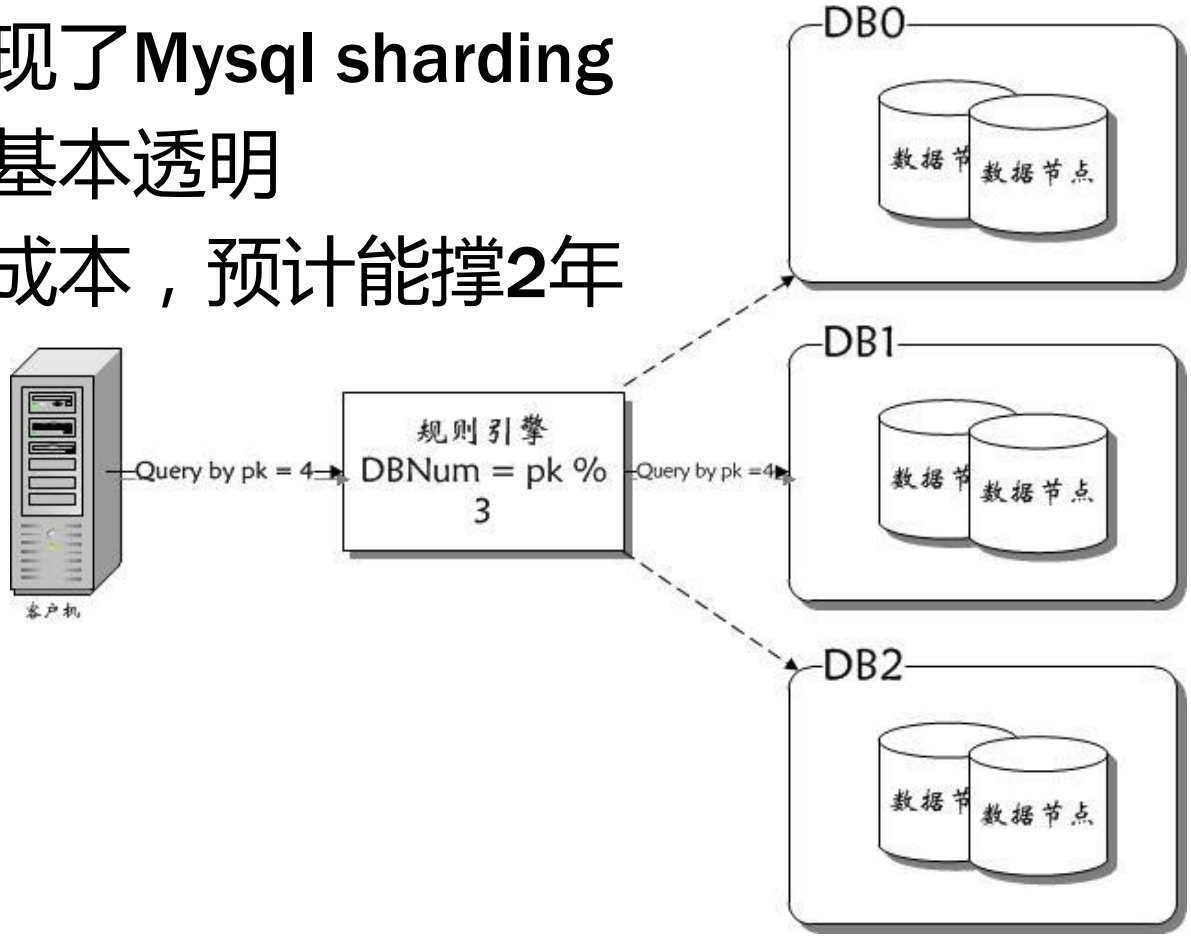
    三个人坐在哪里干想了2个月。。。。

探索阶段

- Done is better than perfect
  - DBA选择了MySQL
    - 产品稳定
    - 社区广泛，坑都被踩过
  - 先做起来
    - 学习和参考amoeba
    - 放弃amoeba的server模型
    - 解析重写，规则改进

探索阶段

- 收藏夹
  - 第一次实现了Mysql sharding
  - 对业务层基本透明
  - 节省一半成本，预计能撑2年



探索阶段

- 多对多关系支持不了
  - 一个买家会收藏多个商品，一个商品可能被多个买家收藏。
  - 商品有title , sku spu等多种属性，冗余将造成大量空间浪费。
  - 妥协后：一张表存用户收藏的商品id,按照用户切分。一张表存商品信息，按商品id切分，商品信息表额外的放到Tair(Memcached)中缓存以加快速度
  - 导致无法满足的需求：
    - 一个用户希望查询自己的所有商品中名字为xxx的数据。性能太低无法实现。

探索阶段

- 收获：
  - **Mysql使用、运维经验积累**
  - 非核心业务，迁移的信心
  - 核心业务能撑更久。。

- 付出
  - 增加了业务完成功能时的复杂度
  - 部分功能无法简便高效的支持
  - 业务不能直接从单机扩容到多机，还是要做改造

全面推广阶段

TAOBAOJM

全面推广阶段

- 主要需求
  - 随着MySQL的稳定性得到验证，小业务迁出同时，核心业务也蠢蠢欲动。。。

全面推广阶段

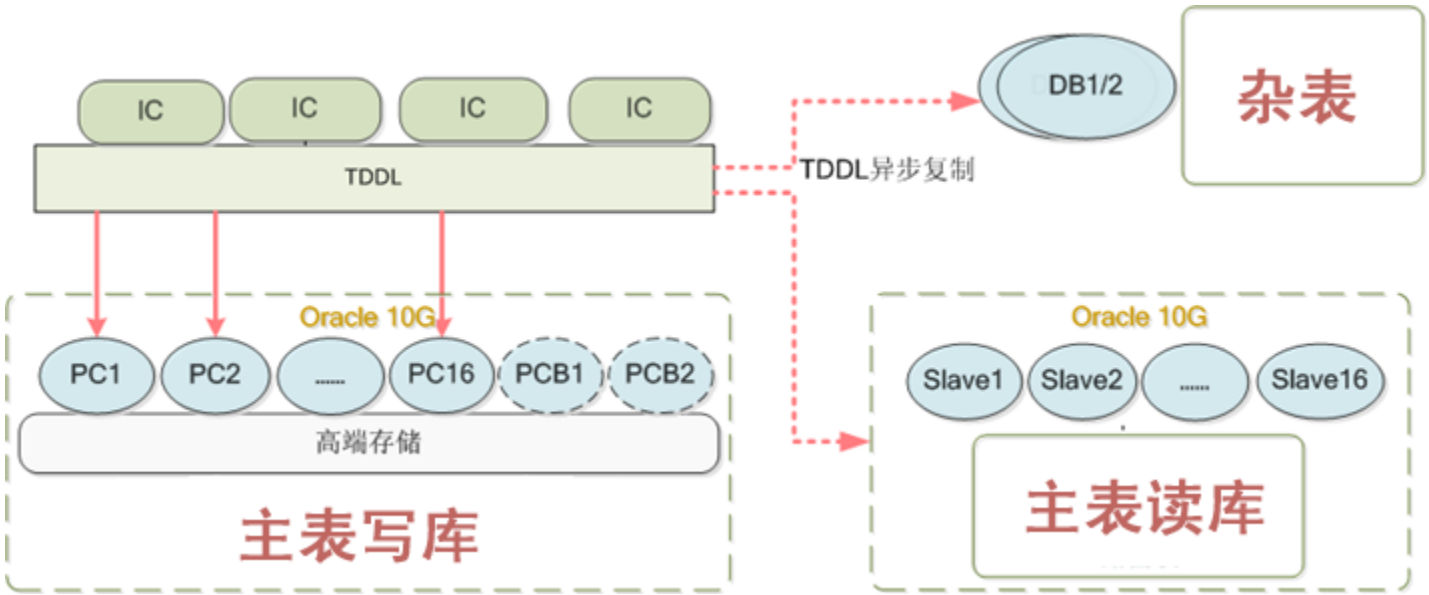
- 商品mysql迁移
  - 淘宝最重要的系统之一
  - 大量开发折腾了近一年才完成
  - 积累了大量沉默成本
  - 今天的我们的主角儿

全面推广阶段

- 背景
  - 商品增长迅速，短短一年商品数量从7亿增长到15亿。
  - 按这个算，满足两年需要需要额外八套小机。
  - 线上服务不能停止
  - 用pc server能否撑的住访问量，不确定。

全面推广阶段

- 项目一期
  - 以oracle+pc server * 16 ，按照sellerid切分。作为读库，2台小机作为主库，切读流量到备机，缓解主机压力
  - TDDL作为数据复制的组件被引入到ic里面



全面推广阶段

- 为了满足按照商品id查询的需求，引入了itemid->sellerid的路由映射表。
  - Select * from tab where sellerid=?
    - 直接按sellerid走读库
  - Select * from tab where id = ?
    - 先走id->sellerid映射表拿到sellerid
    - 以sellerid走规则，得到在哪个库，按id 查读库
  - Select * from tab where UUID=?
    - 先查UUID->sellerid 映射表拿到sellerid
    - 以sellerid走规则，得到在哪个库，按UUID 查读库

全面推广阶段

- 为了适应新的存储模型，需要推动业务调用方去掉UUID字串。
- 因为业务方改进需要时间，系统内不得不遗留大量ugly的兼容性代码。
- 项目持续了近半年的时间才基本完成

全面推广阶段—痛苦的失败

- 容量预估失败
  - 写数据就已将磁盘io吃满
  - 读分离到新读库，tair(memcached)命中一波动，数据库就卡死。
  - 数据库机器 Load 最高到9
  - 上去以后疲于解决各种问题，无数天睡不好觉

全面推广阶段

- Oracle 不像想象中那么给力
  - 经常因为波动hung住，需要断电源重启
  - ora在线支持不理想，黑盒子，一抹黑
  - 11g 线下测试出现各种问题，无法直接上线使用
  - 10g 的oci驱动 bug比较多。。。
  - Oracle的会话共享占用过多业务内存。

全面推广阶段

- 卖家id切库 数据分布不均匀

name	db_role	os	load1
	primary	Linux[2.6.9-89.ELsmp]	9.68
	primary	Linux[2.6.9-89.ELsmp]	5.15
	standby	Linux[2.6.9-89.ELsmp]	4.31
	standby	Linux[2.6.9-89.ELsmp]	3.75
	primary	Linux[2.6.9-89.ELsmp]	3.62
	primary	Linux[2.6.9-89.ELsmp]	2.96
	primary	Linux[2.6.9-89.ELsmp]	2.88
	primary	Linux[2.6.9-89.ELsmp]	2.62
	primary	Linux[2.6.9-89.ELsmp]	2.58
	primary	Linux[2.6.9-89.ELsmp]	2.57
	primary	Linux[2.6.9-89.ELsmp]	2.53
	primary	Linux[2.6.9-89.ELsmp]	2.42
	primary	Linux[2.6.9-89.ELsmp]	2.40
	standby	Linux[2.6.9-89.ELsmp]	2.21
	primary	Linux[2.6.9-89.ELsmp]	2.19
	primary	Linux[2.6.9-89.ELsmp]	2.17
	primary	Linux[2.6.9-89.ELsmp]	2.15
	primary	Linux[2.6.9-89.ELsmp]	2.13
	primary	Linux[2.6.9-89.ELsmp]	1.62

全面推广阶段

- 数据复制后验证困难
  - 15亿数据，考虑线上压力情况，全部跑一次查询验证要7天时间
  - 谁也说不清数据是否全部复制成功
  - 抽样有一定误差，大量时间消耗在复制数据一致性问题排查。

全面推广阶段

- 事务问题
  - 多个数据源并存
  - 老代码中事务逻辑散落各处
  - 原来是个事务的操作，现在不是了。。
  - 某业务场景**rollback**失败，赔钱。

全面推广阶段

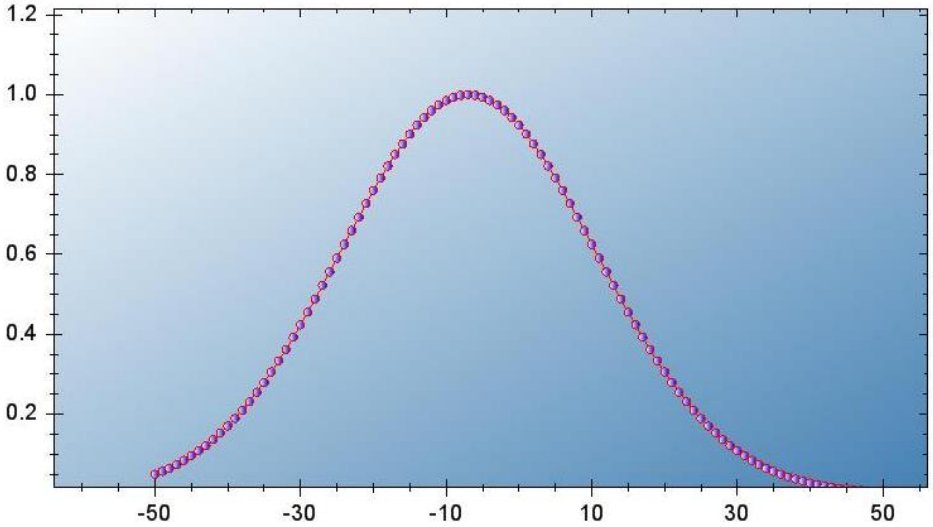
- 基本上这次尝试没有成功，但有积累
  - 容量预估失败问题：
    - 尽可能用线上环境测试，不行就模拟真实线上环境
  - Oracle不给力问题：
    - Oracle 不大行，mysql?
  - 分布不均匀问题：
    - 按照卖家切分真的合适么？商品id切分更均匀简单
  - 备库验证困难问题
    - 为什么要走备库模式？验证后，直接用pc机做主机就好了
  - 事务问题
    - 测试测试测试

全面推广阶段

- 商品二期
  - 直接主库去小机
  - 使用Mysql
  - 褚霸团队协助，引入了SSD做二级缓存的技术，解决了磁盘iops问题。
  - 放弃路由表，使用商品id作为切分条件。
  - 前端tair机器扩展，缓存率做到90%+

全面推广阶段

- 压力测试（压力测试团队）
  - 线下采用正态分布模型进行5倍，10倍流量测试。
  - 收集了线上访问数据，尽可能线下拟合。
  - 上线前，利用线上机器，动态引流压力测试



全面推广阶段

- 切换与回滚准备
  - 因为数据量太大，为了保证线上正常服务，将数据准备分为全量和增量两个部分进行
  - 制定了详细的分批流量切换方案。
  - 回滚方案则是偷偷做的。。
    - 不少人觉得回滚方案会增加复杂度，额外增加风险
    - 但我们的项目负责人还是偷偷的准备了mysql复制回oracle的退路。。

全面推广阶段

- 收获
  - 将SSD成功的引入了淘宝数据库体系—楮霸团队
  - 一套成功的针对核心项目的压测体系—性能测试团队
  - 服务结构梳理和代码测试完善—业务服务团队
  - 各种异常情况的处理方法固化—TDDL团队
  - 一批新的工具和组件—TDDL团队
  - 新的负载数据和运维经验—DBA团队
  - 淘宝核心系统去小机流程和操作方法—淘宝网

全面推广阶段

- 付出
  - 代价
    - 读库一套苦撑了近一年，没能带来多少生产收益
    - 测试开发 近一年的时间 ，处理各种隐藏问题。
  - 业务开发速度降低
    - 去o版本和日常版本双分支迭代，合并痛苦
  - 加班严重，精神压力大
    - 老有问题解决不了啊。。压在身上

收官阶段

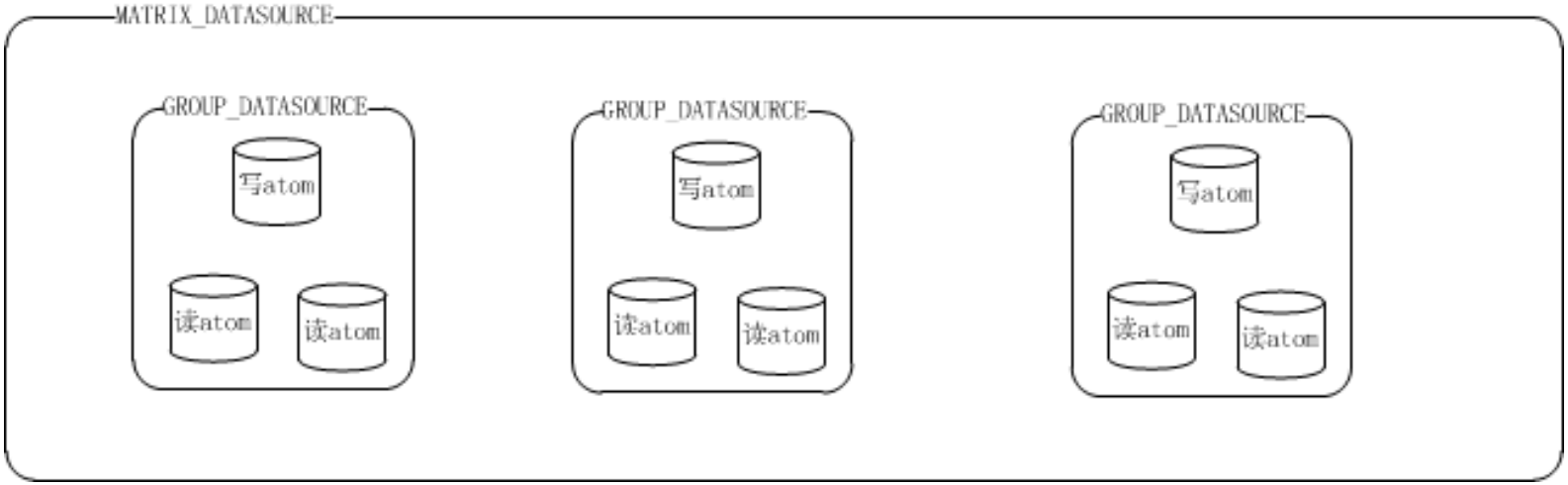
TAOBAOJM

收官阶段

- 需求
  - 小应用要求快速迭代，数据切分后有些事情做不了或做起来难度加大，影响开发效率
  - 数据库越来越多，维护成本猛增

收官阶段

- TDDL对产品进行了组件化改进
  - 三层数据源
  - Matrix 对应需要分库分表的业务应用（有所限制）
  - Group 对应不分库分表的小业务应用（允许做所有事）

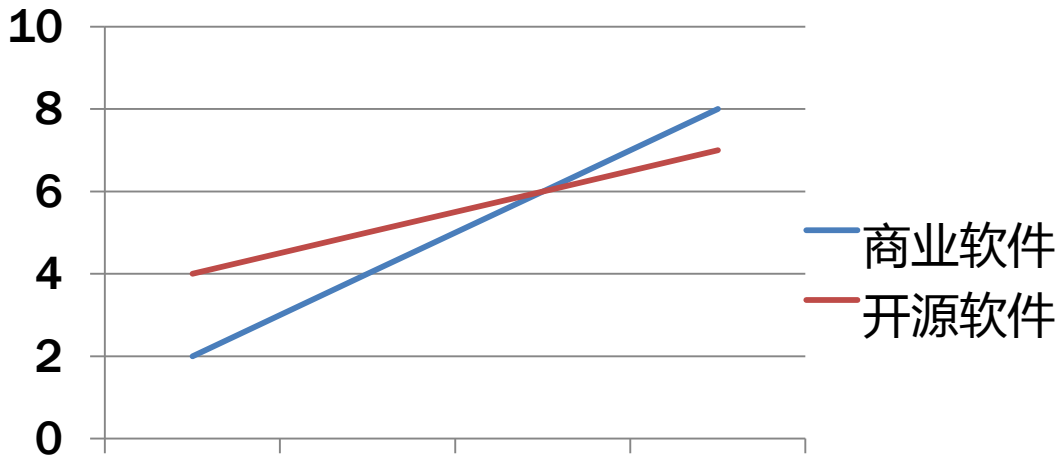


收官阶段

- TDDL对产品进行了组件化改进
  - 额外提供外围系统运维工具
    - 自动扩容
    - 整库迁移
    - 数据库管控配置系统
    - 监控系统

小结

- 去O不是关键，节省成本才是问题的核心。
  - EMC AIX 是成本的关键
  - 适合的场景选择适合的软硬件结构
  - 能不动，就不动，避免瞎折腾。



THANK YOU

TAOBAOJM