Gdevops

全球敏捷运维峰会

利用MySQL
构建百万级别的彩票系统

演讲人：李辉

About Me

- 李辉,常用网名：门牙没了
- 现任新浪爱彩网数据库运维负责人
- 2011年加入新浪爱彩数据库团队

大纲

- 互联网彩票业务鼎盛时的现状
- MySQL数据库相关的规范
- MySQL高可用架构选型
- MySQL sharding拆分
- 利用NoSQL为MySQL减压

互联网彩票业务鼎盛时的现状

2014年巴西世界杯新浪彩票在线用户系统的概况

- 同时在线高峰150万+, 月活跃用户1000万+
- 注册用户5000万+
- 流水日志存储3个T+，单表最大100G+，单表最大行数20亿+
- 高峰并发单DB 2万+QPS
- 高峰投注订单销量一天3亿RMB

MySQL数据库相关的规范

- 开发规范
  - 表字段、索引设计规范
  - SQL编写规范
  - Schema Review
- 运维规范
  - SQL审核
  - 权限控制
  - MySQL版本选择

开发规范-字段设计规范

- 字段数量建议不超过20-50个
- 做好数据评估
  - ⊙建议纯INT不超过1500万，含有CHAR的不要超过1000万
  - ⊙字段类型在满足需求条件下越小越好，使用UNSIGNED存储非负整数，实际使用时候存储负数场景不多
- 将字符转数字存储
  - ⊙使用UNSIGNED INT存储IPv4 地址而不是用CHAR(15)，这种方式只能存储IPv4，存储不了IPv6
  - ⊙考虑将日期转化为数字，如：from_unixtime()、unix_timestamp()
- 所有字段均定义为NOT NULL
  - ⊙除非你真的想存储null

开发规范-索引设计规范

- 所有表必须有显式主键
  - InnoDB表是以主键排序存储的IOT表。
  - 尽量使用短，自增的列做索引。
  - 复制结构中row格式中如果表有主键可以加速复制。
  - UNSIGNED INT自增列，也可以考虑BIGINT
  - TINYINT做主键可能导致mysql crash
  - 类型转换导致查询效率很低
  - 可用uuid_short() 代替uuid()，转成BIGINT存储
- 合理地建立索引
  - 选择区分度高的列作为索引
  - 单个索引字段数不超过5，单表索引数量不超过5，避免冗余索引
  - 建立的索引能覆盖80%主要的查询，不求全，解决问题的主要矛盾
  - 复合索引排序问题，多用explain去确认

开发规范-SQL编写规范

- 避免在数据库中进行大量计算任务
  - ⊙大事务拆成多个事务，分批多次操作
  - ⊙慎用text、blob大型字段，如要用考虑好拆分方案
  - ⊙频繁查询的字典表考虑用cache抗
- 优化join
  - ⊙避免大表与大表之间的join,考虑让小表去驱动大表join
  - ⊙最多允许三表join,最好控制成两表
  - ⊙控制join后面where选择的行数
- 注重where条件，多用EXPLAIN确认
  - ⊙where条件的字段，尽量用区别度高的字段，这样走索引的性能好。
  - ⊙出现子查询的SQL，确认MySQL版本，利用explain确认执行计划。
  - ⊙进行分页优化;DML时候多个value合并

开发规范-Schema Review

- 字符集问题
  - ⊙表字符集选择UTF8，如果需要存储emoj表情，需要使用UTF8mb4(MySQL 5.5.3以后支持)
- Schema设计原则
  - ⊙核心表字段数量尽可能地少，有大字段要考虑拆分
  - ⊙适当考虑一些反范式的表设计，增加冗余字段，减少JOIN
  - ⊙资金字段考虑统一*100处理成整型，避免使用decimal浮点类型存储
  - ⊙日志类型的表可以考虑按创建时间水平切割，定期归档历史数据
- Schema设计目标
  - ⊙快速实现功能为主，保证节省资源
  - ⊙平衡业务技术各个方面，做好取舍
  - ⊙ 不要在DB里进行大计算，减少复杂操作

运维规范-SQL审核

- 使用SQL审核平台
  - ⊙开源软件Inception，或自行开发
- 尤其注意Online DDL的操作时间

	ONLINE DDL			PT-ONLINE-SCHEMA-CHANGE		
CHANGE OPERATION	ROW(S) AFFECTED	IS TABLE LOCKED?	TIME (SEC)	ROW(S) AFFECTED	IS TABLE LOCKED?	TIME (SEC)
Add Index	0	No	3.76	All rows	No	38.12
Drop Index	0	No	0.34	All rows	No	36.04
Add Column	0	No	27.61	All rows	No	37.21
Rename Column	0	No	0.06	All rows	No	34.16
Rename Column + change its data type	All rows	Yes	30.21	All rows	No	34.23
Drop Column	0	No	22.41	All rows	No	31.57
Change table ENGINE	All rows	Yes	25.30	All rows	No	35.54

运维规范-权限控制

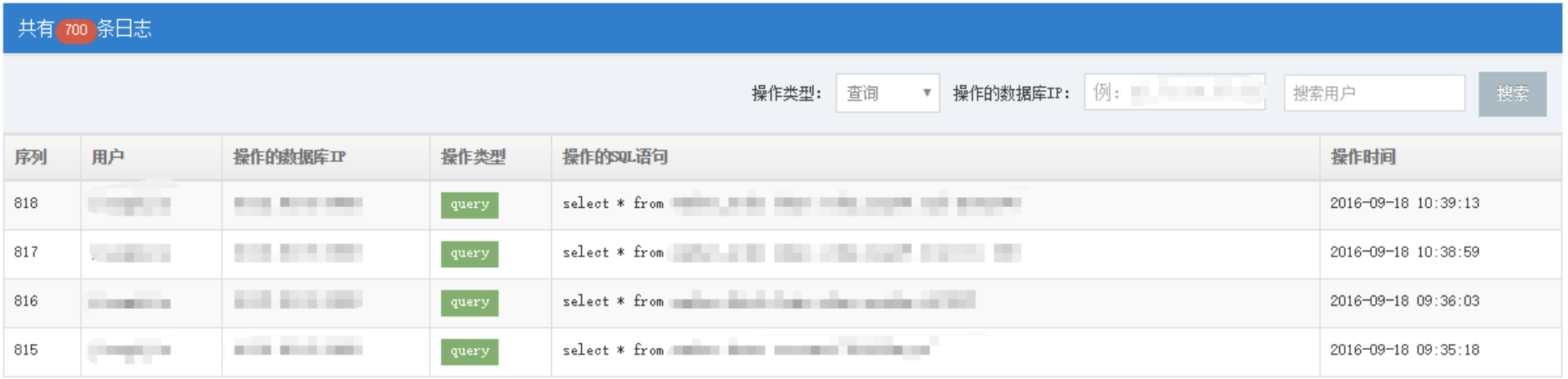
- 启用sql_safe_updates选项
- 账号只给SELECT、INSERT、UPDATE权限，DELETE的逻辑改用UPDATE实现
- 设置密码安全策略。MySQL5.6可启用密码插件，MySQL5.7的密码安全策略更完善。

运维规范-权限控制

- 使用SQL堡垒机
  - ⊙我司通过改造开源工具mywebsql实现
  - ⊙safe_updates也可以在这一层实现

运维规范-权限控制

SQL堡垒机用户操作日志



- ⊙可追溯开发人员对数据库的操作
- ⊙避免大查询或全表更新
- ⊙提供审计功能

运维规范-MySQL版本选择

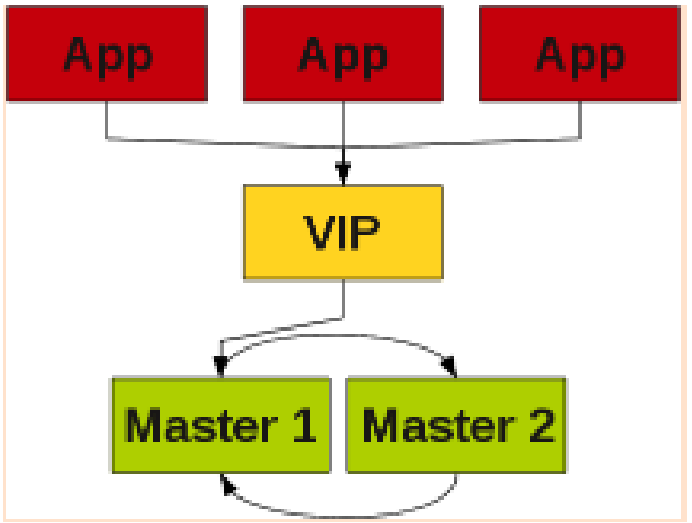
- MySQL社区版，用户群体最大
- MySQL企业版，收费
- Percona Server版，新特性多，和MySQL社区版最接近
- MariaDB版，国内用户不多
- 选择优先级：MySQL社区版> Percona Server > MariaDB > MySQL 企业版

MySQL高可用架构选型

- 双master+keepalived
- MHA
- PXC

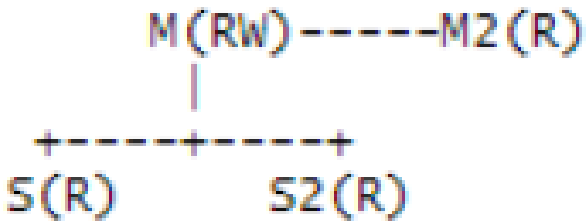
MySQL高可用架构选型-MM

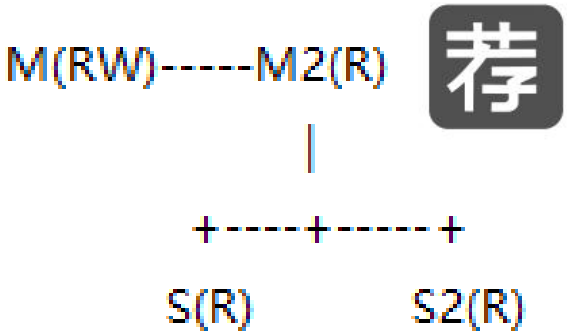
- 双master+keepalived
  - ⊙容易搭建及维护
  - ⊙节省资源，适合业务线比较长的公司
  - ⊙扩展方便，可以随意添加只读库、灾备库
  - ⊙可以进行单边的维护
  - ⊙需要加强keepalived的检测机制
  - ⊙注意脑裂问题
  - ⊙数据一致性稍差
  - ⊙重复写入问题要注意



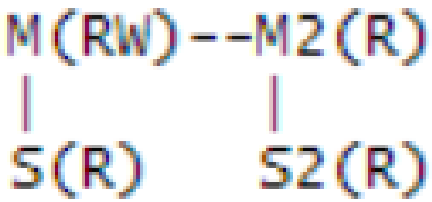
MySQL高可用架构选型-MM

⊙注意只读库的可用性设计



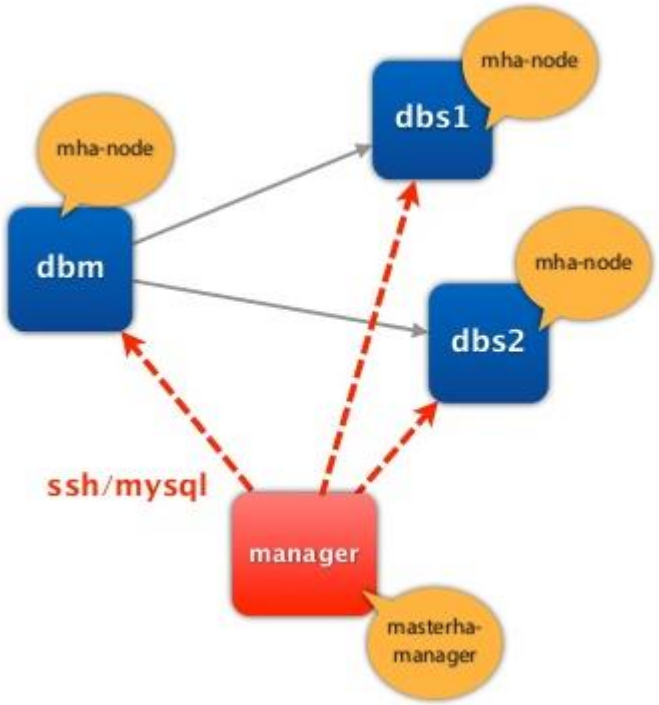


荐



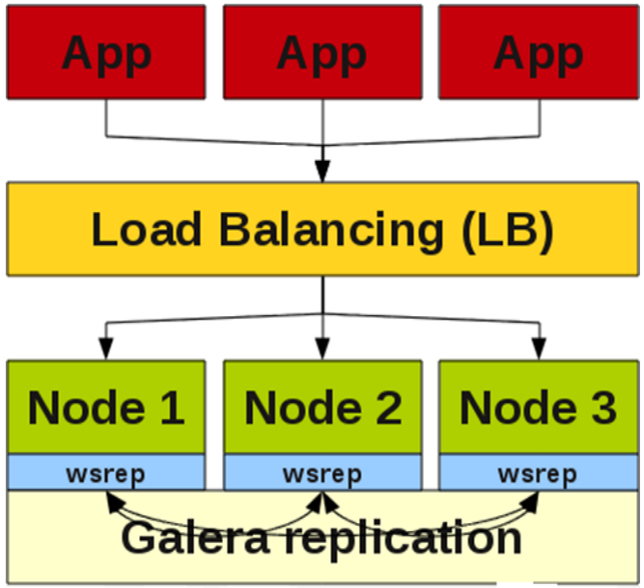
MySQL高可用架构选型-MHA

- MHA
  - ⊙故障切换能自动补齐binlog，最大程度保证数据一致性
  - ⊙Perl语言，可以进行二次开发
  - ⊙从服务器自动切换，无需人工干预
  - ⊙需要ssh互信，内网安全要做到位
  - ⊙建议搭配binlog server使用
  - ⊙至少3个节点，适合读写比较高的应用
  - ⊙适合业务单一但数据库压力大的公司



MySQL高可用架构选型-PXC

- PXC
  - ⊙同步复制，解决了传统架构复制延迟的问题
  - ⊙数据一致性优先，多点并发写时锁冲突、死锁问题多
  - ⊙多主复制，每个节点都可以读写数据，但写压力仍会同步到所有节点，并且无法解决热点更新问题
  - ⊙数据强一致
  - ⊙不要有大事务
  - ⊙木桶效应
  - ⊙并发效率有损失
  - ⊙网络要求较高，建议万兆网络
  - ⊙适合数据强一致的业务



MySQL高可用架构选型

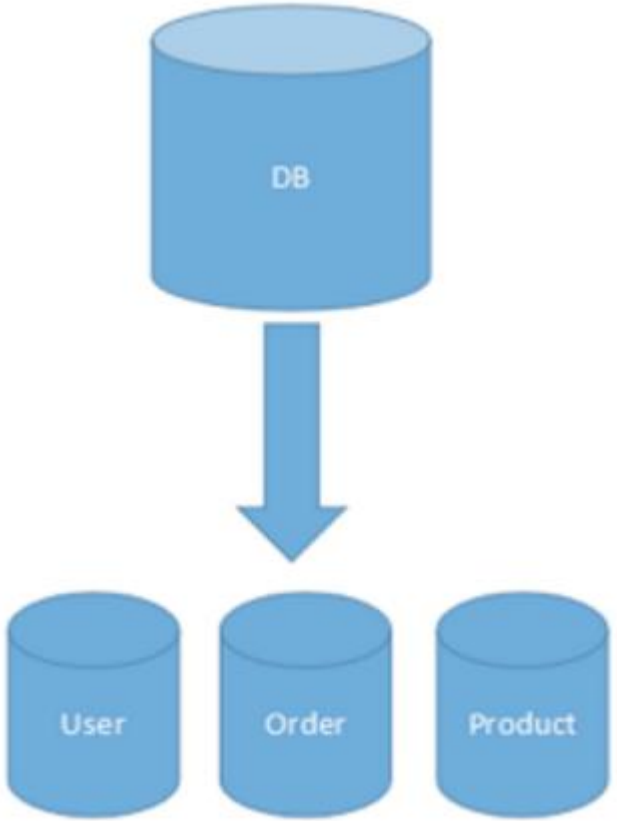
没有最好的架构，只有最适合的架构

MySQL拆分原则和分库分表设计

- 为什么要拆？
  - ⊙单库并发较大
  - ⊙单库物理文件太大
  - ⊙单表过大，DDL无法接受
  - ⊙防止出现性能瓶颈，提升性能
  - ⊙防止出现抖动不稳定现象

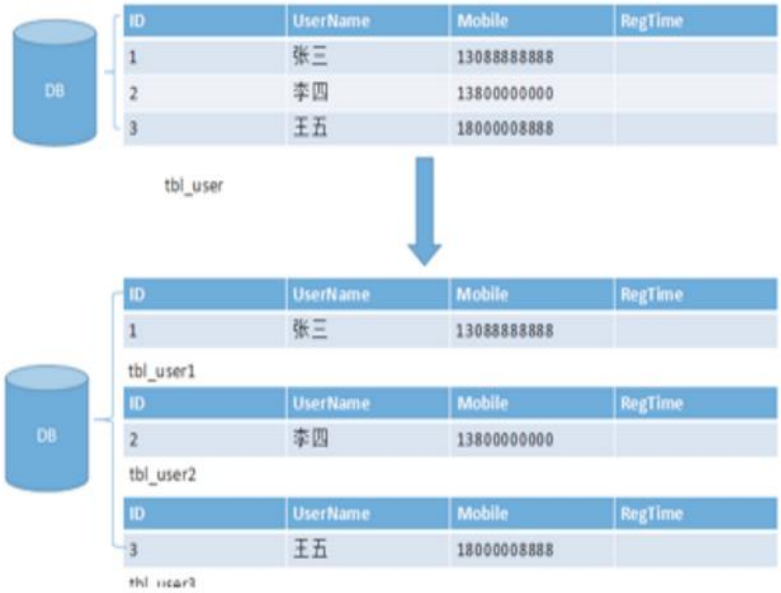
MySQL拆分原则和分库分表设计

- 怎么拆？
- 垂直拆分
- 优点
  - ⊙ 拆分简单明了，拆分规则明确
  - ⊙ 应用程序模块清晰，整合容易
  - ⊙ 数据维护方便易行，容易定位
- 缺点
  - ⊙ 表关联需要改到程序中完成
  - ⊙ 事务处理变的复杂
  - ⊙ 热点表还有可能存在性能瓶颈
  - ⊙ 过度拆分会造成管理复杂



MySQL拆分原则和分库分表设计

- 水平拆分
- 优点
  - ⊙ 不会影响表关联、事务操作
  - ⊙ 超大规模的表和高负载的表可以打散
  - ⊙ 应用程序端改动比较小
  - ⊙ 拆分能提升性能，也比较易扩展
- 缺点
  - ⊙ 数据分散，影响聚集函数的使用
  - ⊙ 切分规则复杂，维护难度增加
  - ⊙ 后期迁移较复杂



MySQL拆分原则和分库分表设计

- 先分库还是先分表？
- 目标还是为了扩展
- 根据具体的场景来决定是先分库还是先分表

MySQL拆分原则和分库分表设计

- MySQL拆分中间件

MySQL拆分原则和分库分表设计

- 但，最好用的中间件还是要自己动手

MySQL拆分原则和分库分表设计

- 我们怎么拆
  - ⊙根据业务的具体情况选择垂直拆分或水平拆分
  - ⊙先剥离活动、后台统计等业务
  - ⊙消息类数据，基于时间维度进行拆分，动态拼出表名
    单表5G-10G
    行数500-1000w
  - ⊙用户类数据，按照HASH或RANGE进行拆分。
    比如好友关系的拆分，分库分表都可以。
- 并发仍然比较高怎么办？
  - ⊙在时间维度拆分的基础上再按RANGE或HASH进行拆分。
- 不要过度拆分

MySQL拆分原则和分库分表设计

- 实践：SNS用户消息表拆分
- 业务描述：

  ⊙5000万用户，每个用户平均30条消息，总数据量15亿左右
- 拆分方法：

  单表1000w数据量，拆分成150张表，按消息ID进行拆分

  不分库：

  msg_id%150=[N] -> [0,150] -> msgdb.msg_N

  逐步打散：

  msg_id%150=N->[0,75] ->msgdb.msg_N

  msg_id%150=N->[76,150] -> msgdb1.msg_N

  极限扩容：

  msg_id %150 -> 1 -> msgdb1.msg_1

  .....

  msg_id %150 -> 150 -> msgdb150.msg_150

  或

  (1,3,5,7) ->db1

MySQL拆分原则和分库分表设计

能不拆就不拆，10亿数据量也跑的好好的。
做个“懒”DBA。

MySQL拆分原则和分库分表设计

- 不关键的应用，分区表也能扛

```
PARTITION p20160905 VALUES LESS THAN (1473091200) ENGINE = InnoDB,
PARTITION p20160906 VALUES LESS THAN (1473177600) ENGINE = InnoDB,
PARTITION p20160907 VALUES LESS THAN (1473264000) ENGINE = InnoDB,
PARTITION p20160908 VALUES LESS THAN (1473350400) ENGINE = InnoDB,
PARTITION p20160909 VALUES LESS THAN (1473436800) ENGINE = InnoDB,
PARTITION p20160910 VALUES LESS THAN (1473523200) ENGINE = InnoDB,
PARTITION p20160911 VALUES LESS THAN (1473609600) ENGINE = InnoDB,
PARTITION p20160912 VALUES LESS THAN (1473696000) ENGINE = InnoDB,
PARTITION p20160913 VALUES LESS THAN (1473782400) ENGINE = InnoDB,
PARTITION p20160914 VALUES LESS THAN (1473868800) ENGINE = InnoDB,
PARTITION p20160915 VALUES LESS THAN (1473955200) ENGINE = InnoDB,
PARTITION p20160916 VALUES LESS THAN (1474041600) ENGINE = InnoDB,
PARTITION p20160917 VALUES LESS THAN (1474128000) ENGINE = InnoDB,
PARTITION p20160918 VALUES LESS THAN (1474214400) ENGINE = InnoDB,
PARTITION p20160919 VALUES LESS THAN (1474300800) ENGINE = InnoDB,
PARTITION p20160920 VALUES LESS THAN (1474387200) ENGINE = InnoDB,
PARTITION p20160921 VALUES LESS THAN (1474473600) ENGINE = InnoDB) */
```

利用NoSQL为MySQL加速

- 为什么要使用NoSQL？
  - ⊙数据存储在内存中，访问速度快
  - ⊙能支持大批量操作及爆发性负载
  - ⊙数据结构丰富，有效缓解MySQL压力
  - ⊙协议简单，支持各种语言的API
  - ⊙存储大量数据无需担心性能

利用NoSQL为MySQL加速

- 使用redis的注意事项
  - ⊙不要依赖redis的持久化
  - ⊙不要求事务、join
  - ⊙注意数据的大小

利用NoSQL为MySQL加速

- 如何利用redis给MySQL加速
  - ⊙ 利用K/V结构，缓存结果
    - ◎ 用户信息
    - ◎全局排行
  - ⊙ 利用其丰富的数据结构为MySQL减压
    - ◎ 计数器update+1
    - ◎ 排序
    - ◎ Hash（把表映射到Redis中）
    - ◎ 消息队列

Gdevops

全球敏捷运维峰会

THANK YOU！