云数据库管理系统及在互联网行业的应用

浙江大学计算机科学与技术学院
陈 刚

2015年4月16日

網易 NETEASE
www·163·com

浙江大学 ZHEJIANG UNIVERSITY

高校在大数据时代研究面临的挑战

- 不掌握大数据来源
- 很难得到真正的实际需求
- 建设大型试验环境比较困难
- 不能提供足够的人力资源

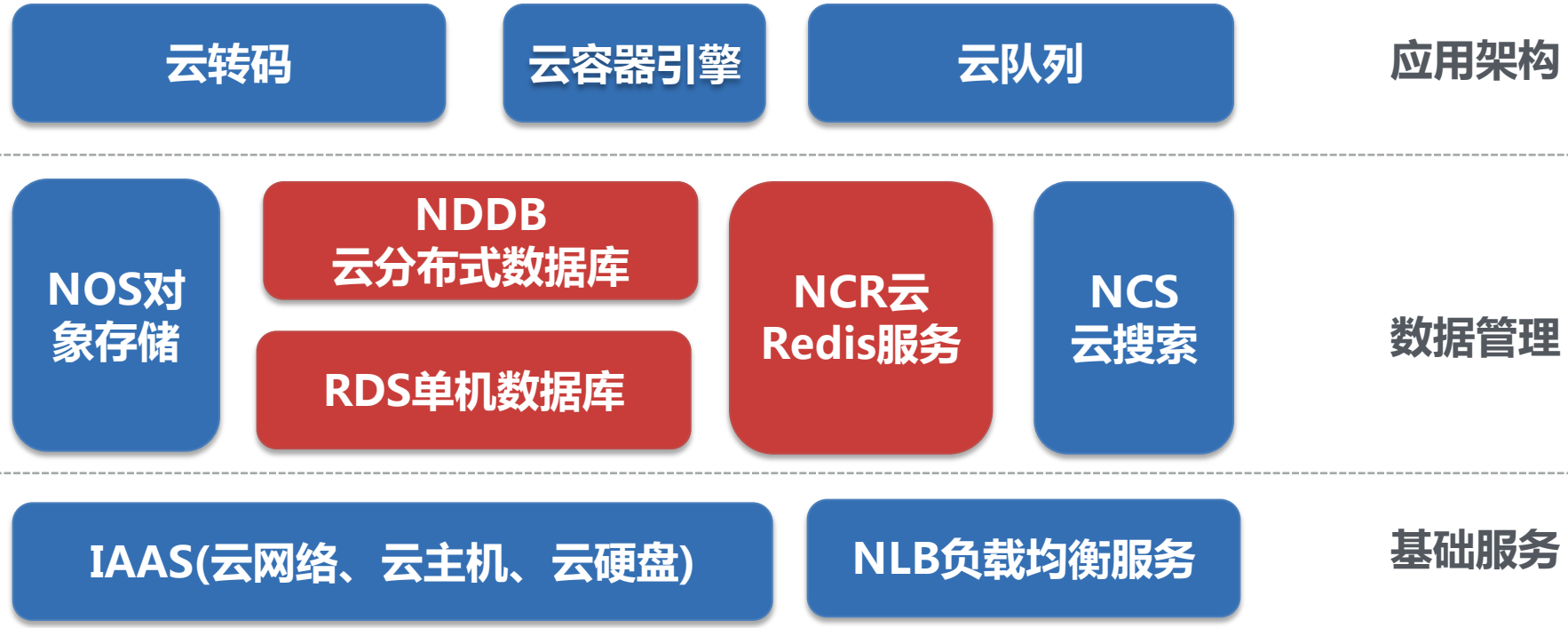
产学研结合

- 大目标统一，相互促进
- 松散耦合，各层关注自身核心利益
- 人的联系紧密（同一人在各层间完成身份切换）

浙大与网易的合作模式

- 学生学习研究期间即与企业研究团队紧密合作（非传统横向项目模式），毕业后选择去该企业就职，乃至担任高级职位
- 创建网易杭州研究院（浙大与网易两块牌子），目前工程研发人员已超过1000人，公共基础技术研究人员超过500人
- 半数以上的技术高级经理、总监毕业于浙大计算机学院
- 网易可以提供资金、大数据资源、用户实际需求提供、开发工程师等的帮助
- 浙大源源不断提供给网易高水平人才，深厚的前期技术积累

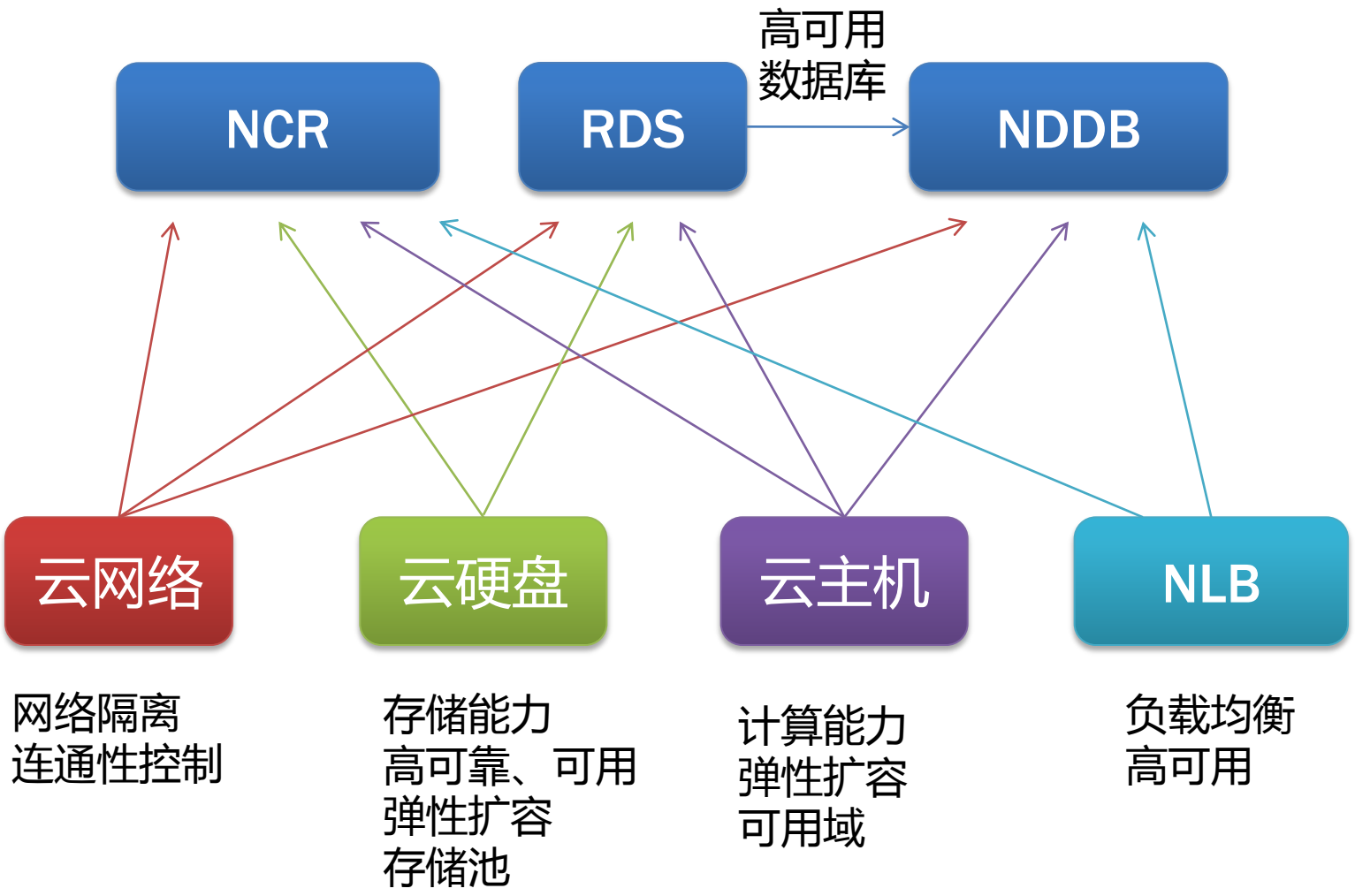
网易私有云的架构


云转码
云容器引擎
云队列
应用架构
NOS对象存储
NDDB
云分布式数据库
RDS单机数据库
NCR云
Redis服务
NCS
云搜索
数据管理
IAAS(云网络、云主机、云硬盘)
NLB负载均衡服务
基础服务

云数据库对云平台的需求

- 网络隔离：
  - 默认隔离不同租户的数据库
  - 灵活配置网络连通性，满足特殊需求
- 故障隔离：
  - 主备虚拟机不能位于同一个物理机、交换机
  - 主备数据不能落到同一个物理机
- 性能保障：
  - CPU、IO、网络性能有保障
- 高可靠：不能容忍数据丢失
- 弹性：　存储、计算规格的弹性伸缩

平台对数据库的支持


高可用
数据库
NCR
RDS
NDDB
云网络
云硬盘
云主机
NLB
网络隔离
连通性控制
存储能力
高可靠、可用
弹性扩容
存储池
计算能力
弹性扩容
可用域
负载均衡
高可用

网络隔离

- 私有网
  - 每个租户拥有一个独立的二层网络
  - 不同用户私有网络完全隔离
- 机房网
  - 不同租户虚拟机之间的互通
  - 租户虚拟机到机房中物理机的互通
  - ACL控制连通性
- 互联网
  - 链接到互联网（外网）

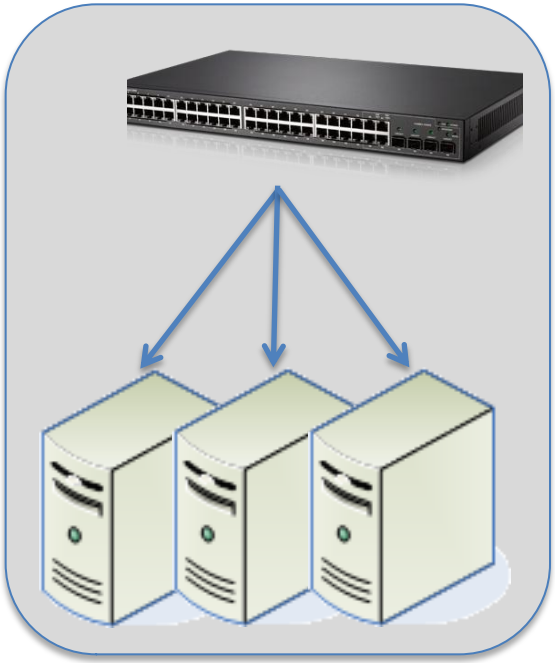
数据库默认运行于私有网络内，不同租户完全隔离。必要时通过机房网络互连

故障隔离风险

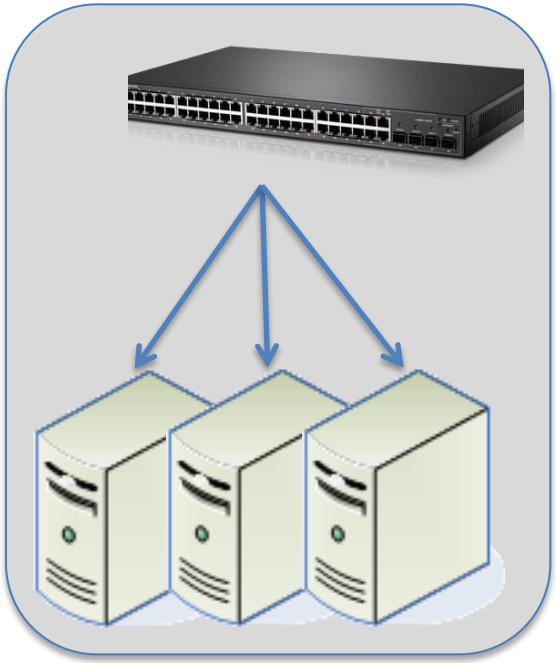
	资源物理位置失去控制	风险
硬盘	主从硬盘的数据可能落在同一物理机或物理硬盘	数据库数据丢失
主机	不同云主机可能位与同一物理机	数据库不可用
联网	不同云主机可能经由同一网卡或网线联网	数据库不可用
交换机	不用云主机可能接入到同一交换机	大量高可用数据库发生不可用

做冗余而不隔离故障， 是伪高可用、伪高可靠

故障隔离： 提供资源位置控制



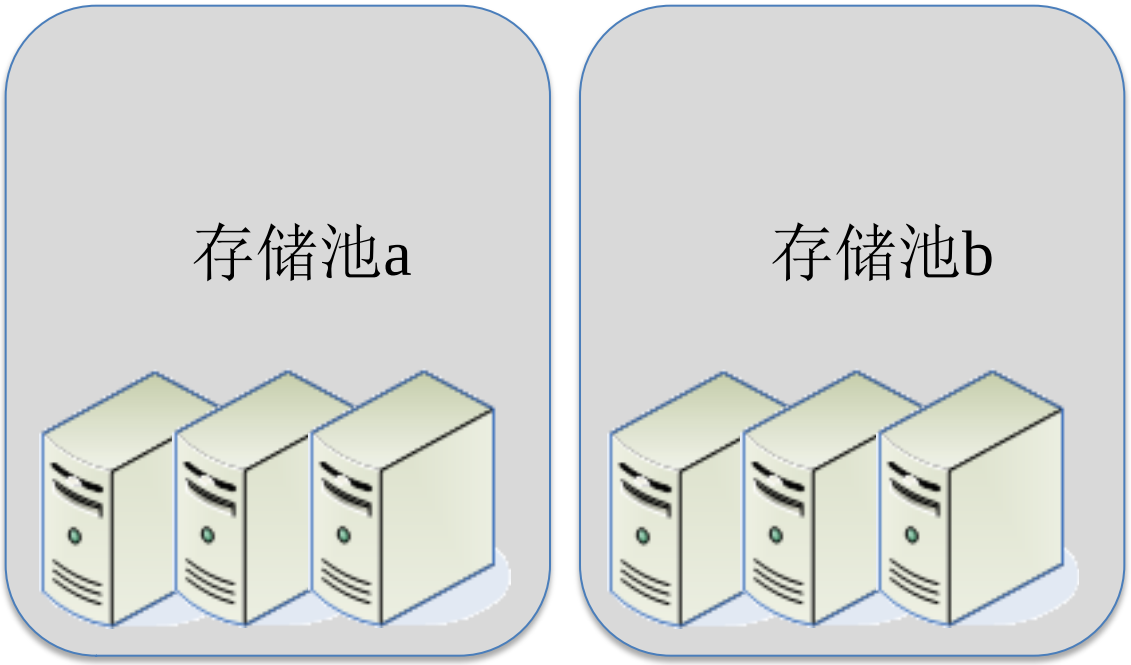
可用域a



可用域b

- 可用域
  - 控制云物理资源位置，确保故障隔离的机制
  - 不同可用域确保物理机、联网和交换机隔离
- 应用场景：主备隔离
  - 数据库主备从不同可用域分配
  - Redis主备
  - 云硬盘两副本

故障隔离： 提供资源位置控制（存储池）

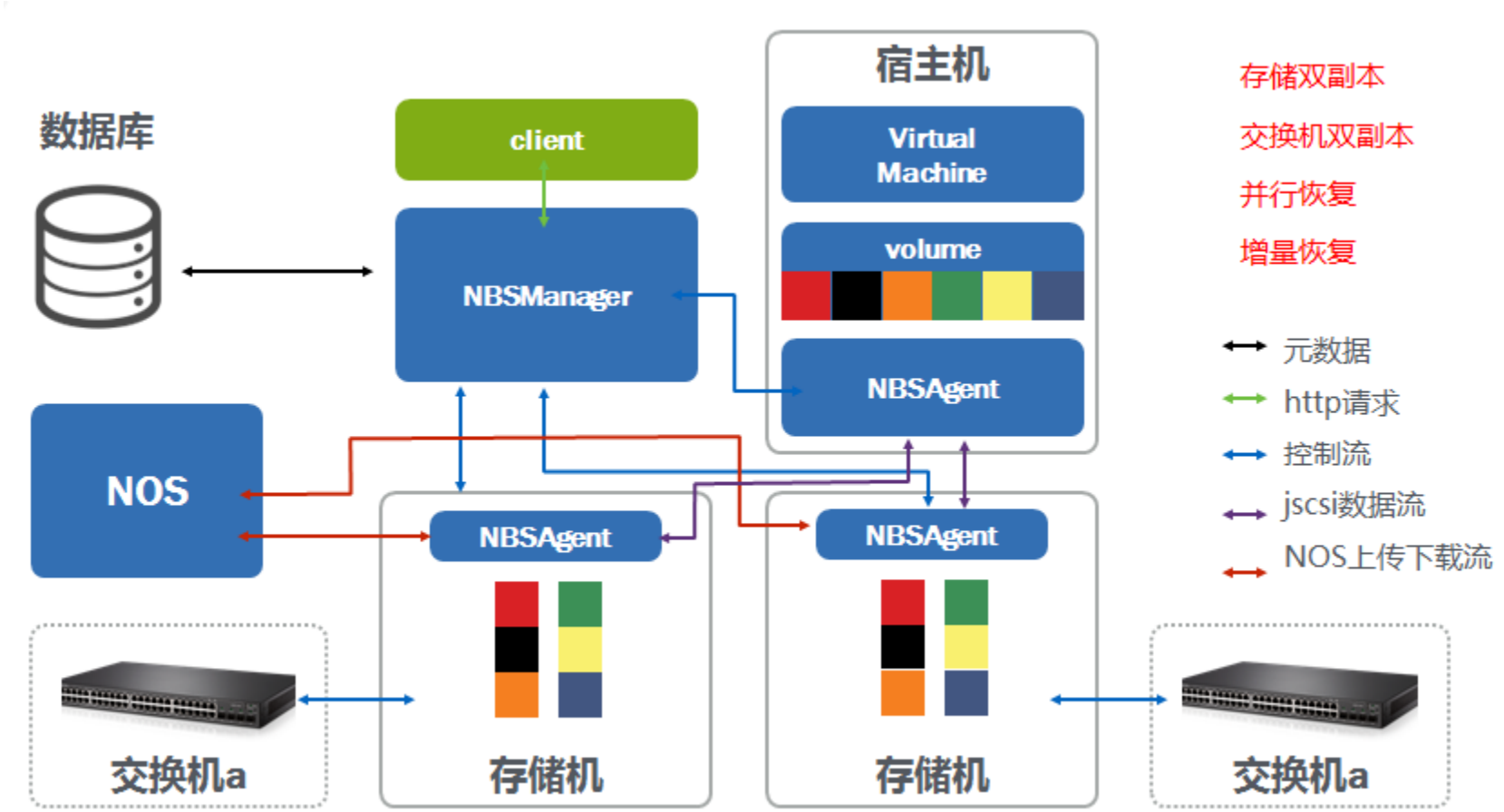


- 存储池
  - 云计算平台提供用于控制云硬盘物理资源位置，确保故障隔离的机制
  - 不同存储池确保物理机隔离
- 应用场景：主备隔离
  - 数据库主备两块云硬盘从不同存储池分配
  - 云Redis主备两块云硬盘从不同存储池分配

高可靠保证：云硬盘

需求：要求云硬盘可靠性大于本地RAID，不要求极高可靠性

因为数据库本身有主从备份



计算性能

容量预留

- 虚拟机计算性能随宿主机总体负载变化可能有20%左右的差异

  宿主机总体负载70%以内
- 虚拟机CPU负载建议不要超过70%
- 云计算平台运维保证宿主机总体负载不超过70%

热迁移

- 借助KVM提供的热迁移机制，可以在几分钟内将虚拟机无感知的迁移到另一台宿主机
- 监控总体负载超高的宿主机，热迁移其它高负载的云主机

逻辑隔离

- 网易云主机设计了可用域功能可以物理隔离不同类型业务的云主机
- 对外业务和内部服务隔离、线上业务和测试系统隔离、数据库与业务隔离

高可用数据库RDS

（新型数据库引擎NTSE +

对新型硬件的支持InnoSQL Flash Cache

+VSR技术）

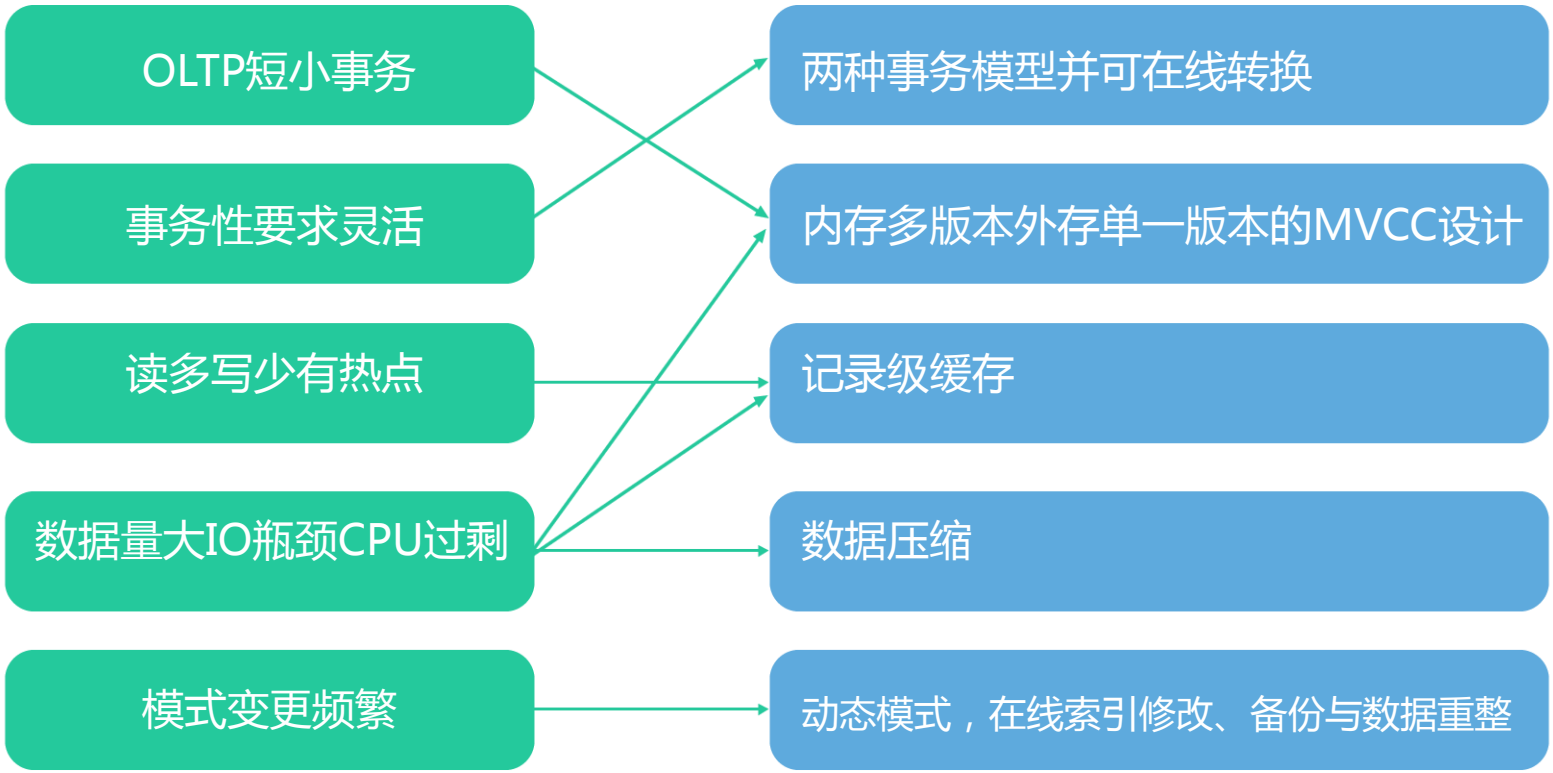
面向互联网的关系数据库需求

- OLTP，短小事务
- 事务性要求灵活
- 读多写少，数据热点明显，往往集中于数据集的10%以内
- 数据量大，IO为系统瓶颈，CPU能力通常过剩
- 产品升级较快，模式变更频繁

面向互联网的关系数据库需求

- 现有业内解决方案
  - 使用MYSQL+InnoDB+Memcached(Redis)
- 主要问题
  - 不必要的事务处理机制，造成性能处理的浪费
  - 页级缓存效率低
  - 空间利用率低下
  - 模式变更困难
  - Memcached难以降低更新负载
- 本质上并未针对互联网应用特点
- 为此，设计实现了一个新型数据库存储引擎NTSE，用MYSQL+NTSE来替代MYSQL+InnoDB+Memcached

关键技术的选择


OLTP短小事务
事务性要求灵活
读多写少有热点
数据量大IO瓶颈CPU过剩
模式变更频繁
两种事务模型并可在线转换
内存多版本外存单一版本的MVCC设计
记录级缓存
数据压缩
动态模式，在线索引修改、备份与数据重整

事务模型

- 两种事务模型
  - 传统ACID事务：保证多实体、多表操作通用事务ACID，支持分布式事务，通用性好
  - 单实体ACID事务：保证对单实体的单次操作微型事务的ACID，性能更优（锁定时间短并发度高、无需前向日志、无死锁）
- 事务模型在线切换
  - 单实体ACID事务表“升级”为传统ACID事务表：创建内存多版本数据结构，瞬时完成
  - 传统ACID事务表“降级”为单实体ACID事务表：purge少量内存多版本数据后销毁内存多版本数据结构，秒级完成

MVCC

- 多版本粒度
  - Oracle：页面级多版本，重建页面开销大
  - NTSE/InnoDB/PostgreSQL：**记录级多版本**，实现简单，效率高
- 版本记录存储
  - Oracle/InnoDB/PostgreSQL：外存存储，存储成本高，旧版本回收效率低，能良好支持超大超长事务；
  - NTSE引擎：**内存存储多版本记录**，外存单一版本记录。存储成本低，旧版本易回收，便于在线事务/非事务切换，但不能支持超大超长事务（应用不需要）；
- 历史记录存储
  - NTSE/Oracle/InnoDB：**独立存储**，方便旧版本回收，减少对应用的影响
  - PostgreSQL：数据文件内存储，旧版本回收特别困难；
- 在牺牲了支持超大超长事务（互联网应用一般无此要求）这一功能后，NTSE在存储、旧版本回收、旧版本操作等方面综合效率优于Oracle/ InnoDB/PostgreSQL

记录级缓存

- 使用记录级缓存，缓存频繁访问的行记录：
  - 内存利用率优于页面级缓存
  - 解决了目前使用Memcached存在的数据一致性和缓存效率问题
- 技术挑战
  - 内存分配：记录大小不一，使用类Linux SLAB分配器，根据记录大小，分为不同的级别；
  - 替换策略：简单LRU内存开销大，使用页内LRU组合页面级最小堆，将内存开销优化到每记录2字节
    - 同记录间的LRU替换算法
    - 根据访问热度及频率确定页面替换或记录替换
  - 缓存一致性：行级缓存修改，记录Redo日志

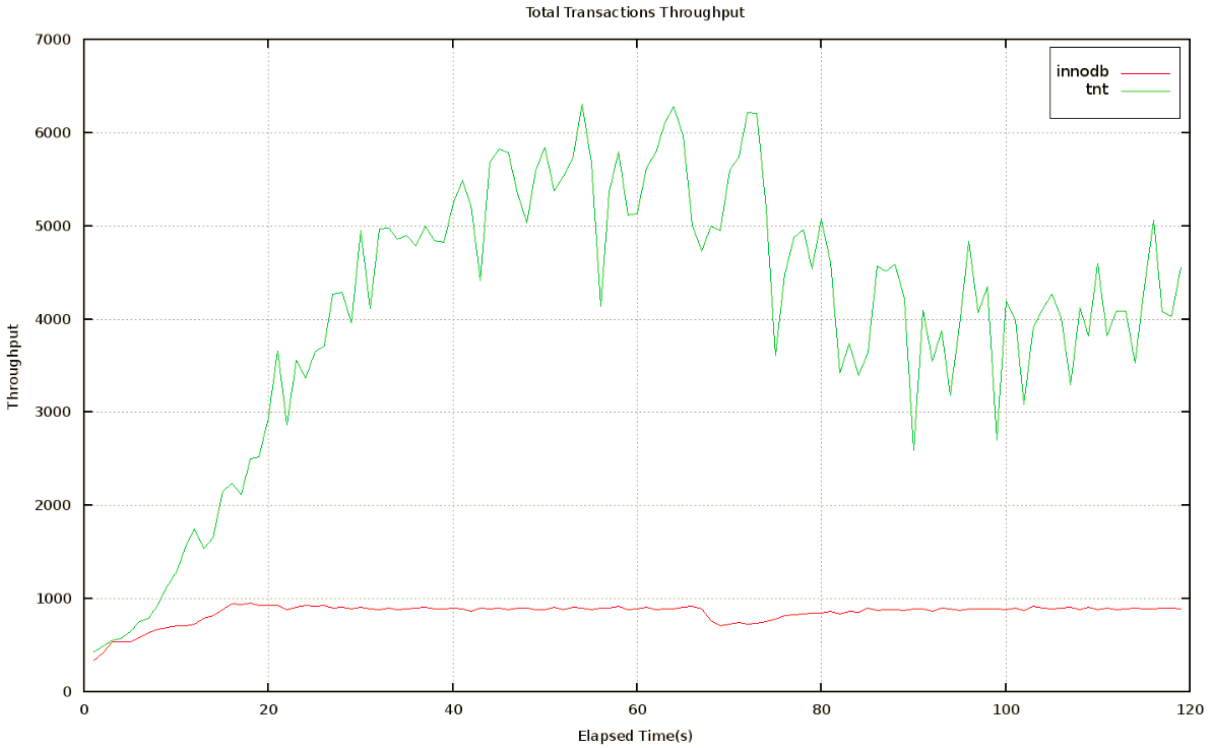
数据压缩

- 压缩算法
  - 表级混合行列压缩
  - 支持多种列内及列间规则压缩
  - 支持后端通用压缩算法
  - 支持多压缩级别
- 基于压缩数据的新型索引
  - 支持基本索引（MAX/MIN）和扩展索引（HASH）
  - 基本索引自动创建，对压缩率和装载速度无影响
  - 扩展索引手动创建，存储成本小于4字节/行，创建速度大大优于传统B+树索引
  - 所有索引对查询过程透明

高可用支持

- 动态模式（dynamic schema）支持，增加字段仅修改表定义，瞬时完成。
- 创建/删除索引，在线进行，不锁表，不影响应用
- 在线数据备份，备份过程不锁表，不锁数据库，不影响应用
- 数据重整（OPTIMIZE），在线进行，不锁表，不影响应用

性能对比（blogbench）



- blogbench，是模拟网易博客应用的基准测试，特点：SQL简单，以简单的查询、更新、插入为主
- 测试数据：1亿条记录
- 性能改善显著，达到InnoDB的4倍以上

InnoSQL Flash Cache设计思路

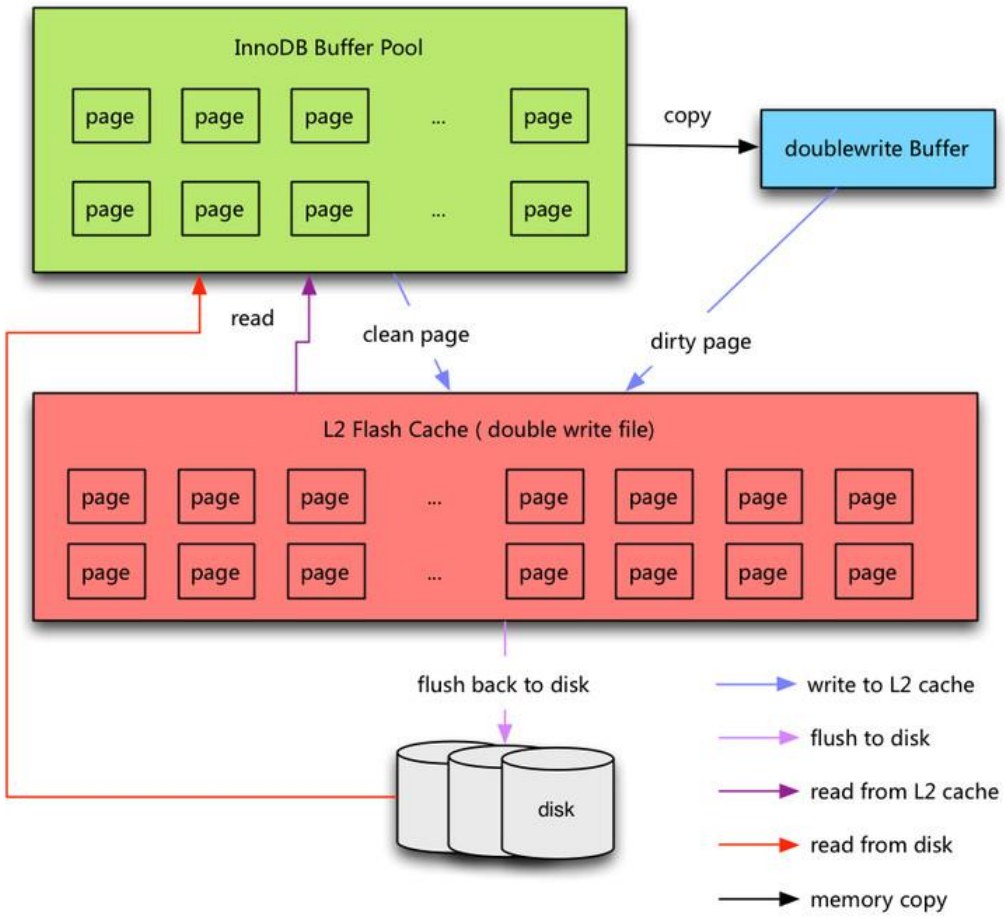
- 很多互联网应用按IOPS/GB计算介于SSD和SATA之间，适合使用SSD+SATA混合存储提升性能降低成本，但目前mysql无法有效利用SSD新型硬件优势

  设计实现了InnoSQL Flash Cache：SSD用作二级缓存

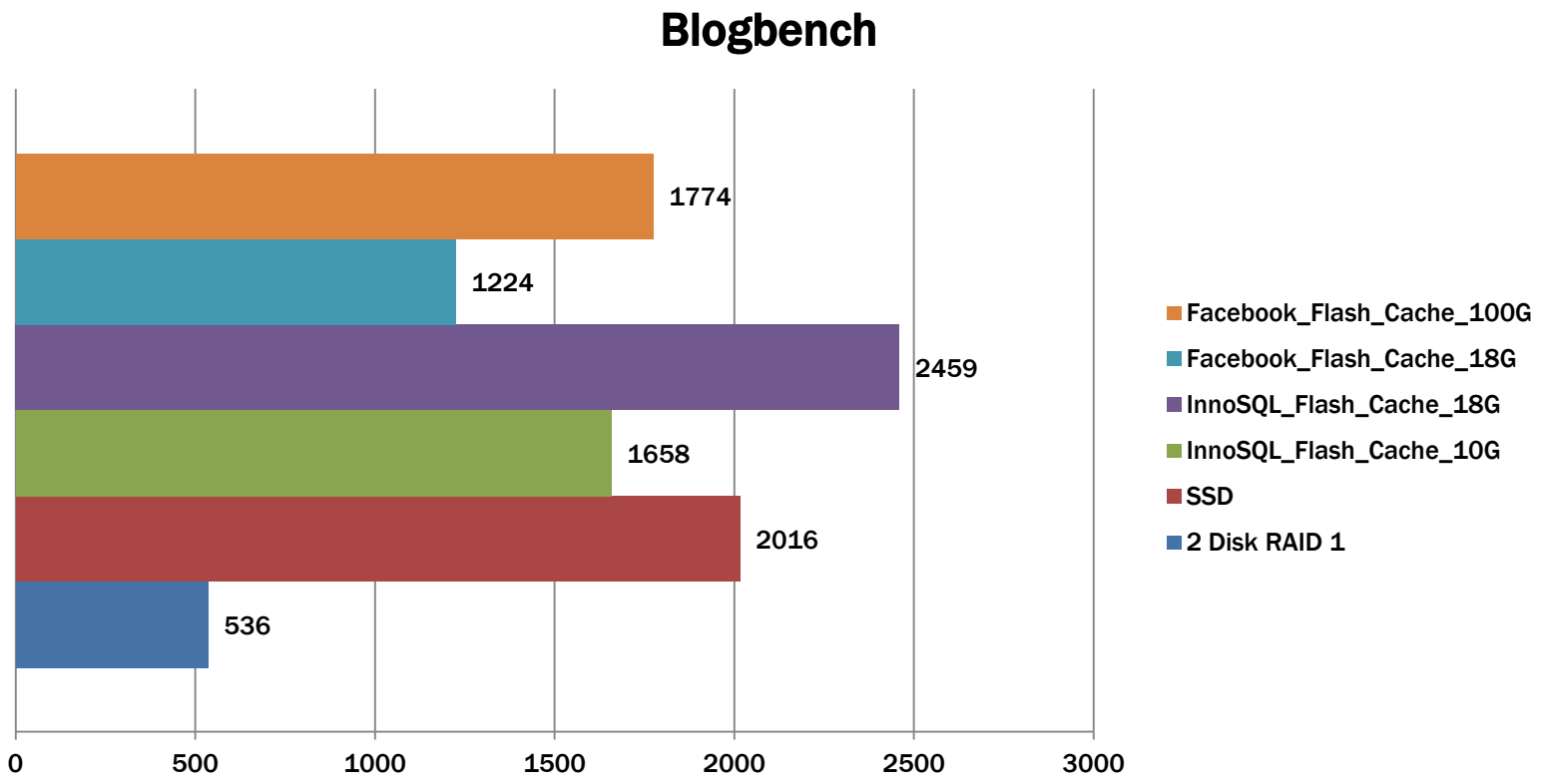
- 可靠性可用性增强
  - Virtual Synchronized Replication：虚拟同步复制，保证崩溃恢复但不保证Slave读取到实时数据
  - Full Synchronized Replication：全同步复制，保证Slave数据访问一致性
  - 缓冲池预热加速，缩短数据库重启预热时间：通过dump/load
  - 复制状态crash safe
  - 并行复制
- 云计算多租户环境需求
  - Profiler：限定数据库用户的IO访问次数和CPU使用时间

InnoSQL Flash Cache

- LRU替换下的页面才写到Flash Cache，避免Flash和内存重复缓存
- 在实现二级缓存的同时也实现了doublewrite功能
- 对SSD只进行顺序写，提升寿命和性能



InnoSQL Flash Cache：性能对比

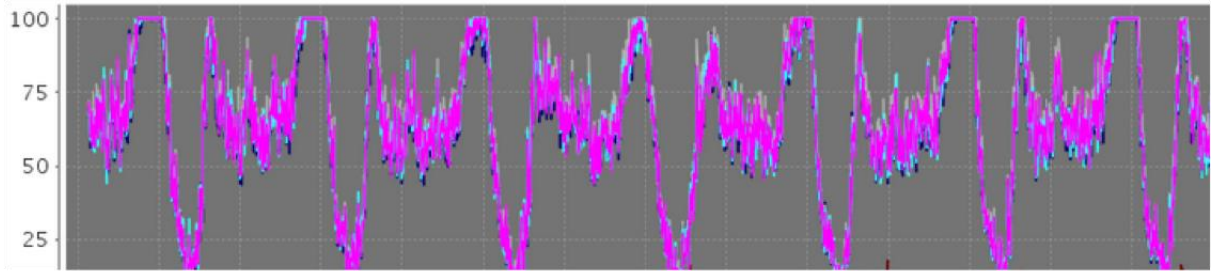


同等SSD缓存大小(18G)时InnoSQL Flash Cache性能是Facebook Flash Cache的2倍；由于InnoSQL Flash Cache对SSD只产生顺序写，性能甚至优于将所有数据存储于SSD时的原生MySQL。

云阅读应用Flash Cache案例

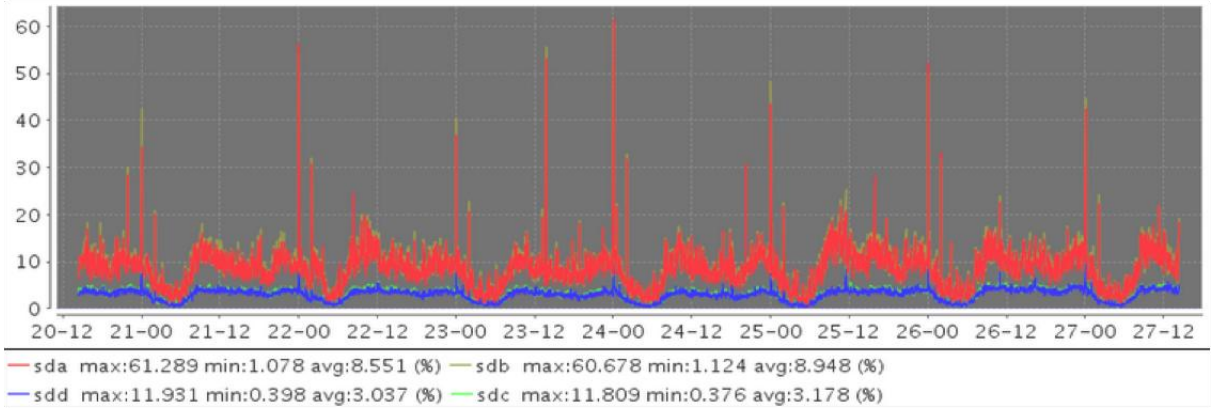
优化前硬件配置

600G SAS*4，
有效容量1.2T



优化后硬件配置

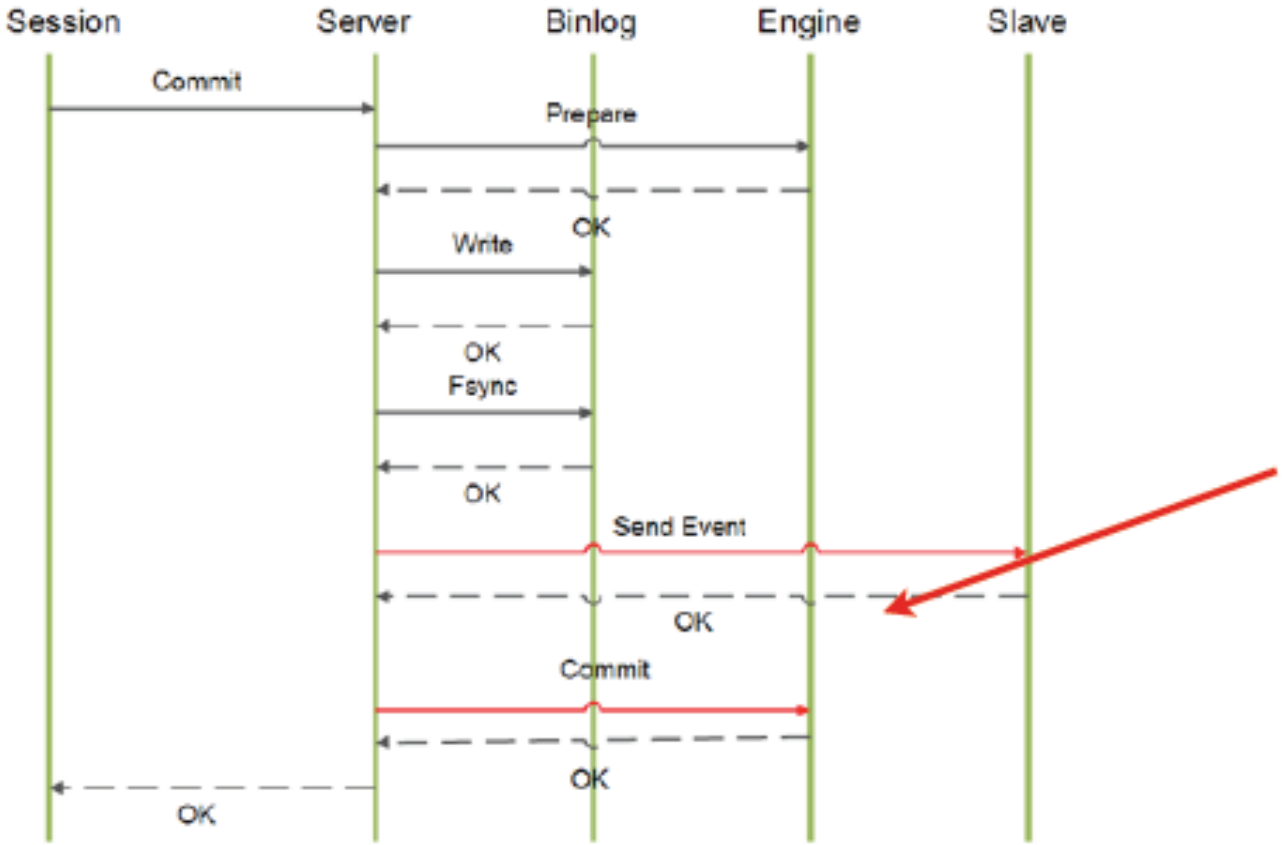
2T SATA*4+160G SSD*2，
有效容量4T
成本接近，容量大增，
IO负载骤降



RDS高可用技术

- VSR复制保证数据不丢
- 心跳和主动监测双重探活
- 并行复制保证主从无延迟
- 秒级切换
  - VIP切换（秒级）
  - Slave Apply完日志(并行复制保证零延迟）
- CAP选择C和A
  - P很少发生
  - 发生P时可能需要人工介入处理

虚拟同步复制：VSR

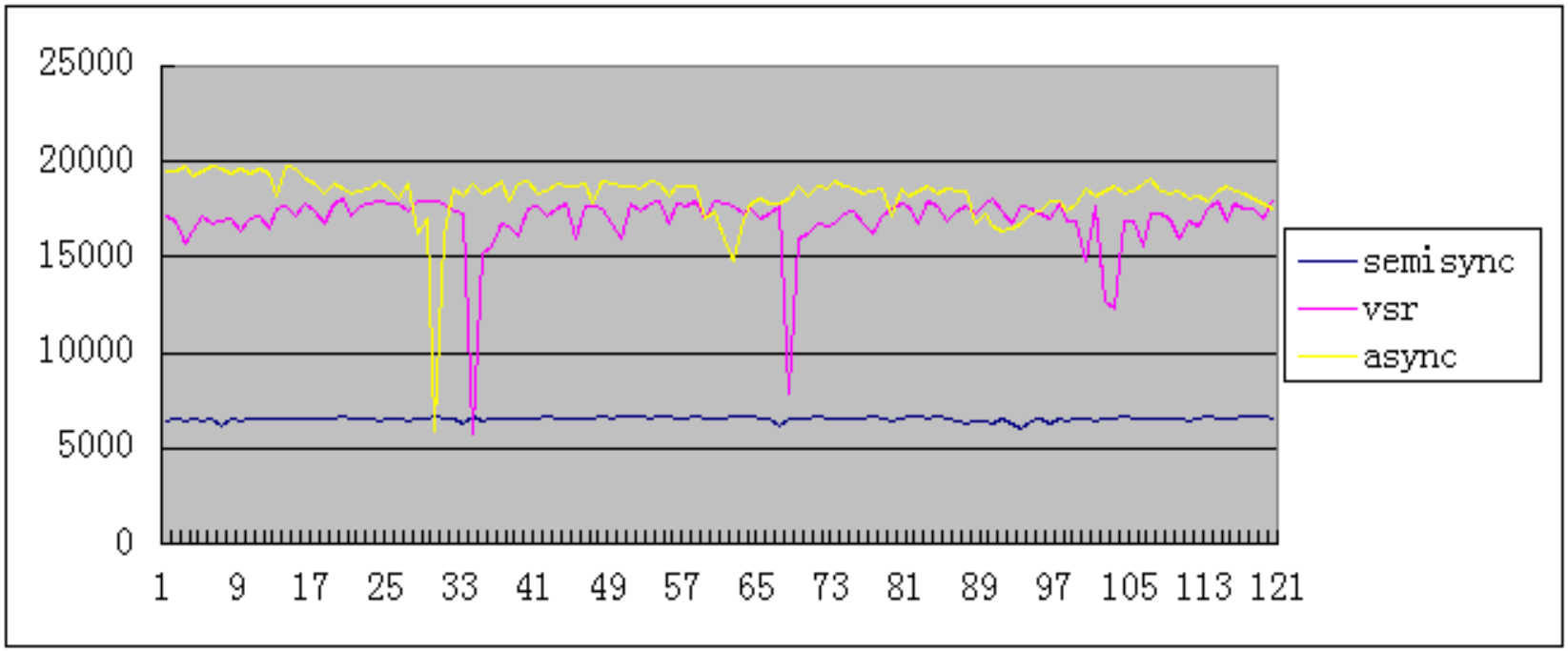


Binlog得到slave节点ack之后，再提交事务
故障时能保证持久性

VSR性能

VSR提供强同步的同时， 几乎没有性能损失

原因：增大了组提交比例， 减少了磁盘IO负载



高可用优化：并行复制

- 解决MySQL原生复制的痛点
  - 从串行执行binlog，主从数据延迟大
  - failover等待数据同步时间久， 可用性差
- 社区的方案
  - 基于表的并行复制：无法解决热点表的延迟问题
  - 基于批量提交的复制：无法充分利用磁盘带宽
  - 基于行的并行复制：并行检测实现难度巨大
- InnoSQL并行复制
  - 原理：一个组提交中的事务都可以并行执行
  - 效果：网易云监控数据库延迟从9个小时降低到无延迟、网易电商数据库延迟从5个小时降低到无延迟，可用性大幅提高

高可用方案对比

	数据丢失	性能	可维护性	failover速度
Replication	大量数据丢失	好	简单	快
Semi-sync Replication	小部分数据丢失	差	简单	快
MHA	数据不丢失，前提是Master服务器依然存活（可能性极低）	一般	简单	快
SAN	数据不丢失，前提是共享存储本身没有发生故障（可能性较高）	一般	数据库简单 SAN设备复杂	慢
Galera	数据不丢失，前提至少有1台服务器存活（可能性较高）	差	复杂	快
网易 RDS/NTSE	数据不丢失，前提至少有1台服务器存活（可能性较高）	好	简单	极快

应用总结

- 关注性能，事务需求不高的应用，可使用NTES提供的非事务功能
- 对事务要求较高的应用，可使用NTES提供的事务功能
- 数据热点集中的应用，可从NTES提供的行级缓存中获取优势
- 数据量大的应用，可使用数据压缩功能
- 提供很好的SSD支持
- 所有应用都能获得高可用支持

云分布式数据库NDDB

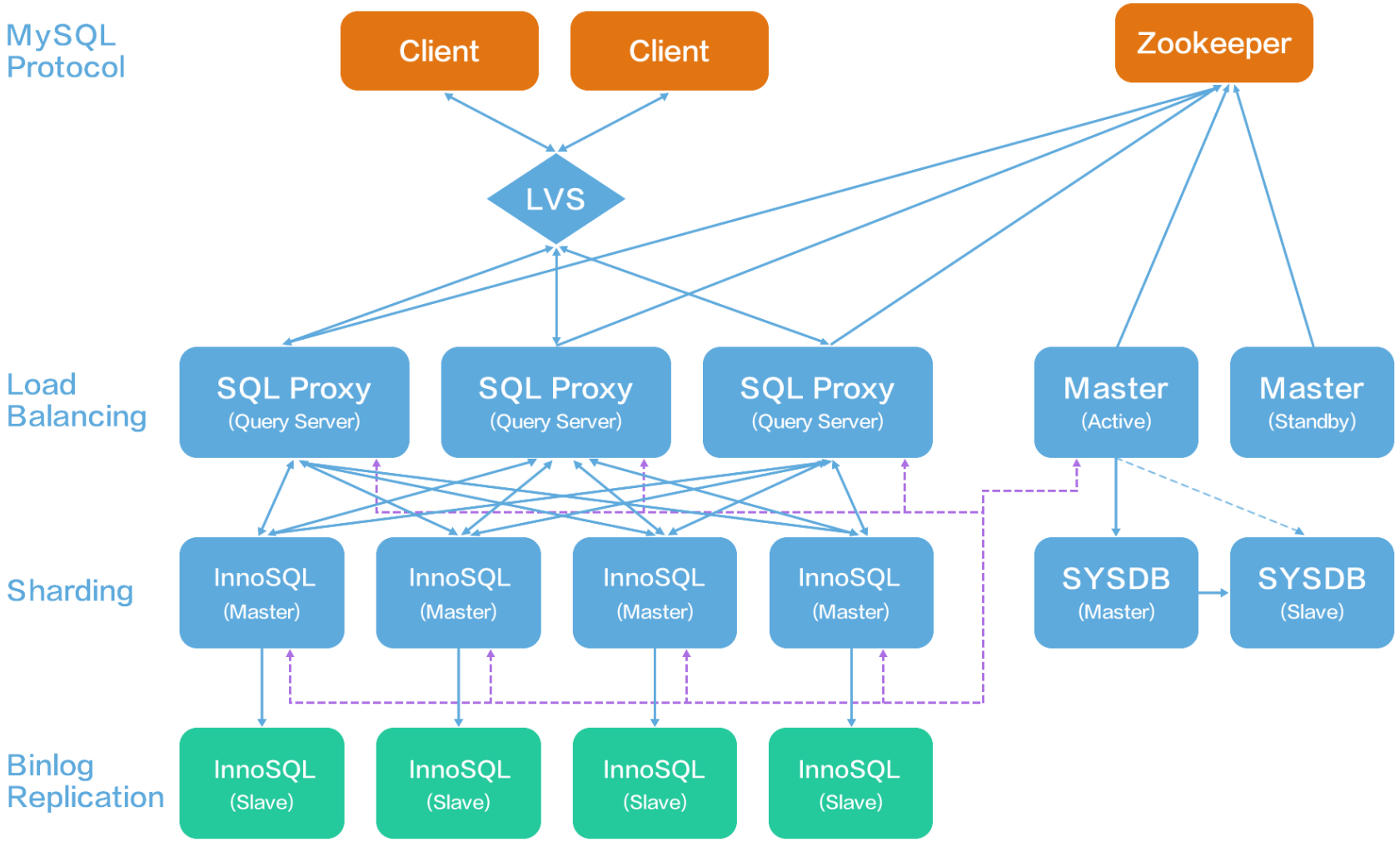
产品目标

- 大容量高并发， ~100TB
- 兼容标准MySQL协议
- SQL兼容度高，支持跨表JOIN，子查询等
- 高可用，高可靠，强一致（继承自RDS）
- 支持从RDS升级到NDDB
- 在线伸缩
- 对开发支持效率高
- 完善的管理特性：备份、统计、监控等

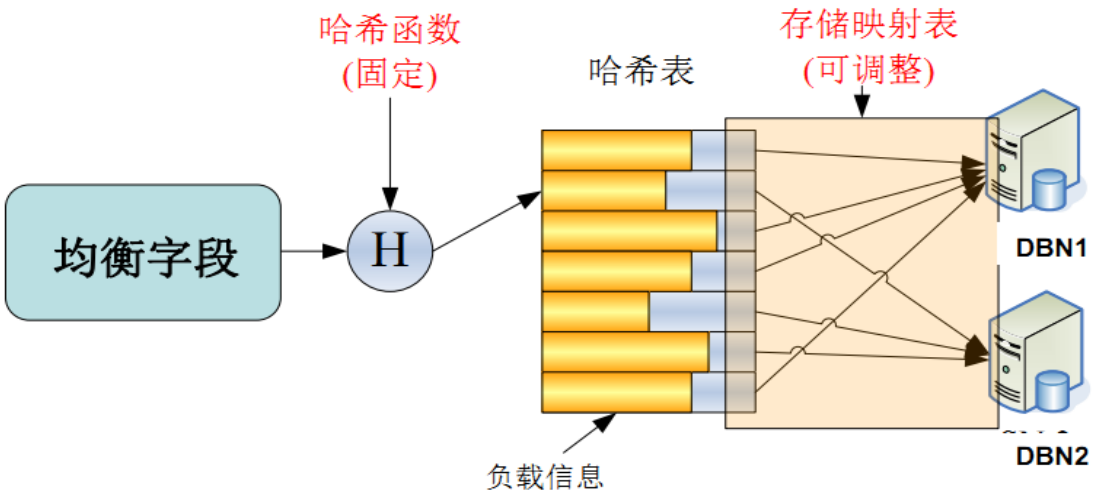
现有方案评估

- Oracle RAC
  - 成本高昂：商业产品；硬件需求(商业存储系统)
  - 可伸缩性风险：现有部署一般不超过20节点
- 中间件产品
  - 市场上缺乏成熟的解决方案
- NoSQL产品
  - 可用性、可靠性保障不足
  - 复杂查询、事务等功能支持不足
  - 缺乏类SQL标准接口，应用开发效率低

NDDB系统架构

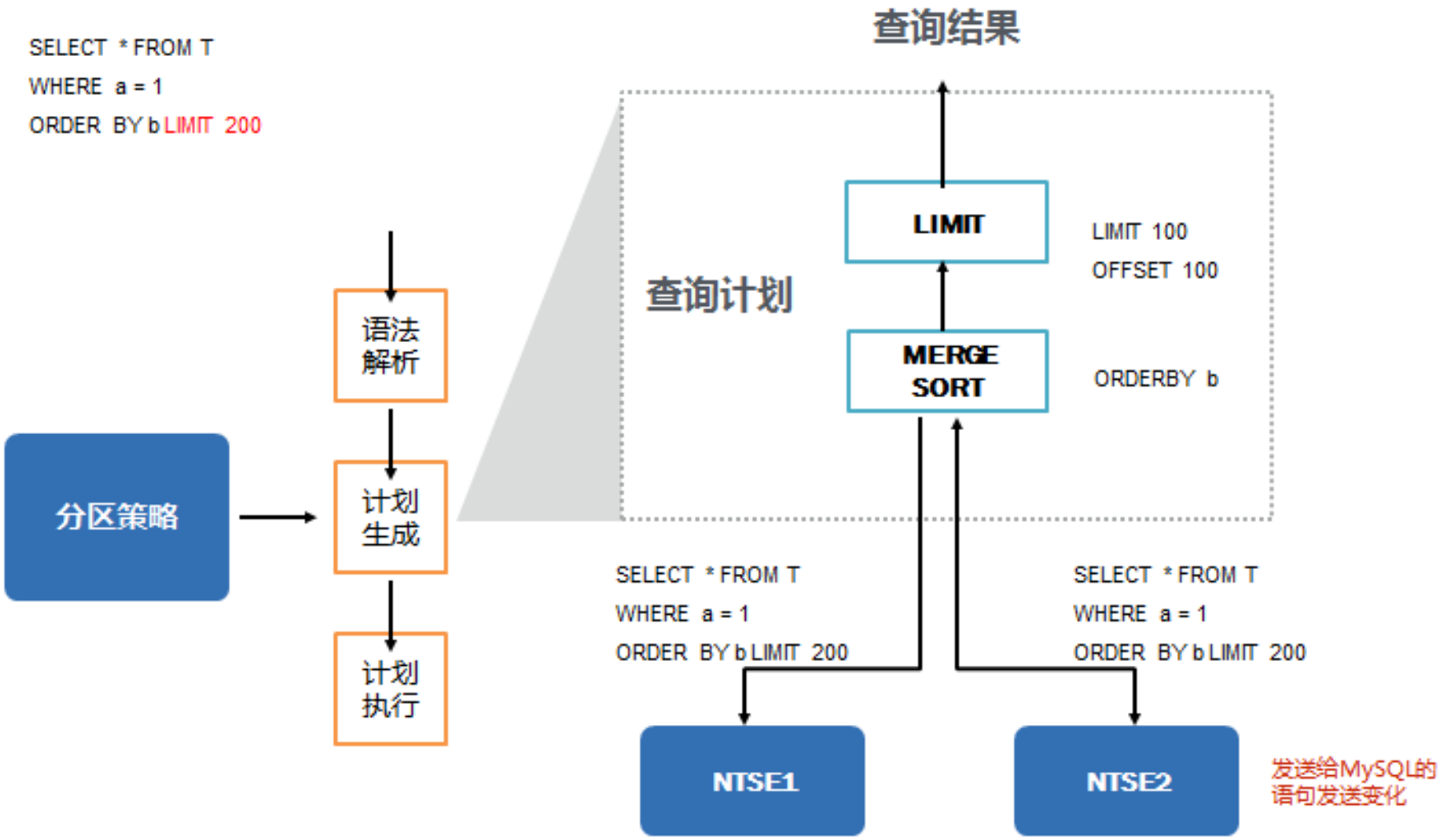

MySQL Protocol
Client
Client
Zookeeper
LVS
Load Balancing
SQL Proxy (Query Server)
SQL Proxy (Query Server)
SQL Proxy (Query Server)
Master (Active)
Master (Standby)
Sharding
InnoSQL (Master)
InnoSQL (Master)
InnoSQL (Master)
InnoSQL (Master)
SYSDB (Master)
SYSDB (Slave)
Binlog Replication
InnoSQL (Slave)
InnoSQL (Slave)
InnoSQL (Slave)
InnoSQL (Slave)

数据Sharding



- **核心一**：表级水平切分、两级映射方案
  - 保证一级Hash固定；方便后续的扩容与收缩
- **核心二**：关联表使用相同分区策略
  - 使用相同分区策略的关联表，可在分区内完成EQUAL-JOIN (Local Join)

查询处理流程


SELECT * FROM T
WHERE a = 1
ORDER BY b LIMIT 200
查询结果
查询计划
LIMIT
LIMIT 100
OFFSET 100
MERGE SORT
ORDERBY b
语法解析
计划生成
计划执行
分区策略
SELECT * FROM T
WHERE a = 1
ORDER BY b LIMIT 200
SELECT * FROM T
WHERE a = 1
ORDER BY b LIMIT 200
NTSE1
NTSE2
发送给MySQL的语句发送变化

在线扩容过程（1扩2为例）

- 全量复制：
  - 新建2个RDS作为目标RDS
  - 源RDS节点全量复制数据到目标RDS
  - 记录全量复制时日志位置
- 删除脏数据： 2个目标RDS各自删除一半无用数据，相当于1拆2
- 增量复制：
  - 从日志的全量复制点开始读取日志回放增量数据到2个目标RDS
  - 等待回访快要完成时，进行悬停切库
- 悬停切库：
  - 锁定所有查询服务器
  - 等待尾部少量日志回访完成
  - 查询服务器（QS）切换到目标RDS
  - 解锁查询服务器， 扩容完成

分布式事务处理

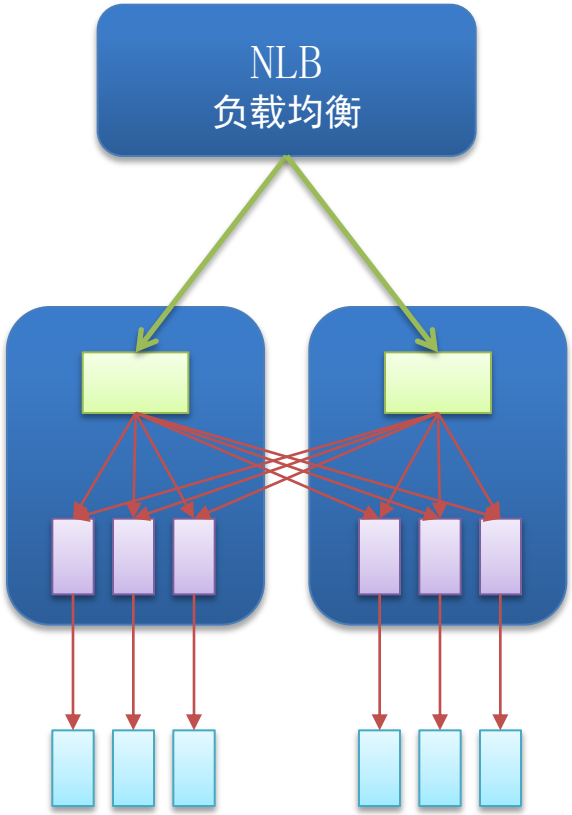
- **核心**：两阶段提交(2-Phase-Commit)支持
- SQL Proxy(Query Server)，作为分布式事务的发起者与协调者
  - 解析用户的SQL，若SQL跨多节点，开始一个分布式事务
  - 用户提交事务，QS作为协调者，发起Prepare请求，记录Prepare日志
  - 待所有参与者均Prepare成功，发起Commit请求，提交事务，记录Commit日志
- 异常情况一：Query Server崩溃并重启
  - 根据Query Server中记录的Prepare日志，处理悬挂事务
- 异常情况二：Query Server永久崩溃
  - 将分布式事务日志存储于高可靠的云硬盘系统保证XA事务ACID

NCR云Redis服务

NCR需求

- 兼容Redis协议
- 弹性伸缩
- 高可用
- 分钟级部署、完善的统计和运维工具

NCR架构

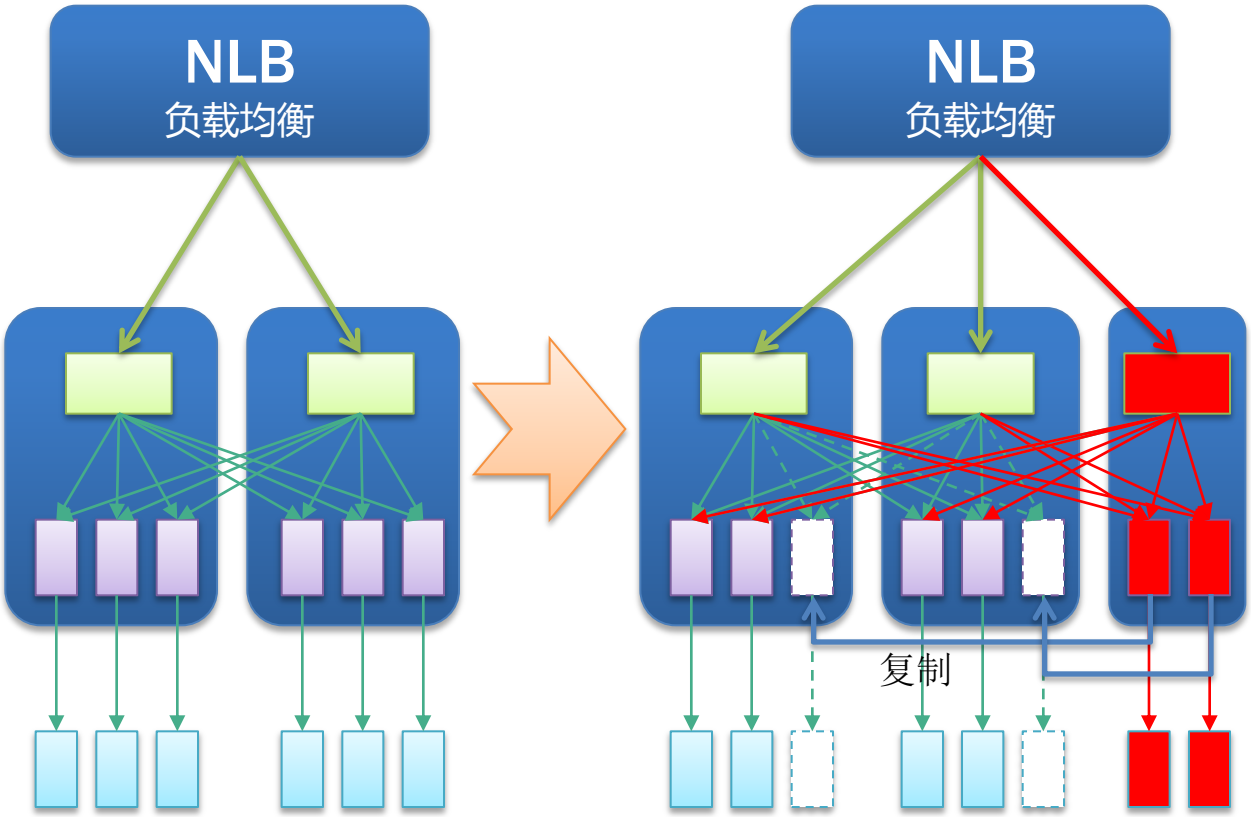


- 4层负载均衡
- 代理节点
- Redis进程

云主机

- 云硬盘

在线扩容



- Pre-sharding: 预先创建所有redis进程
- 通过redis成熟的一对一复制机制迁移部分redis进程实现伸缩
- 理解业务协议的代理节点延时请求消除节点路由切换影响
- 有状态系统实现自适应无缝伸缩的参考架构

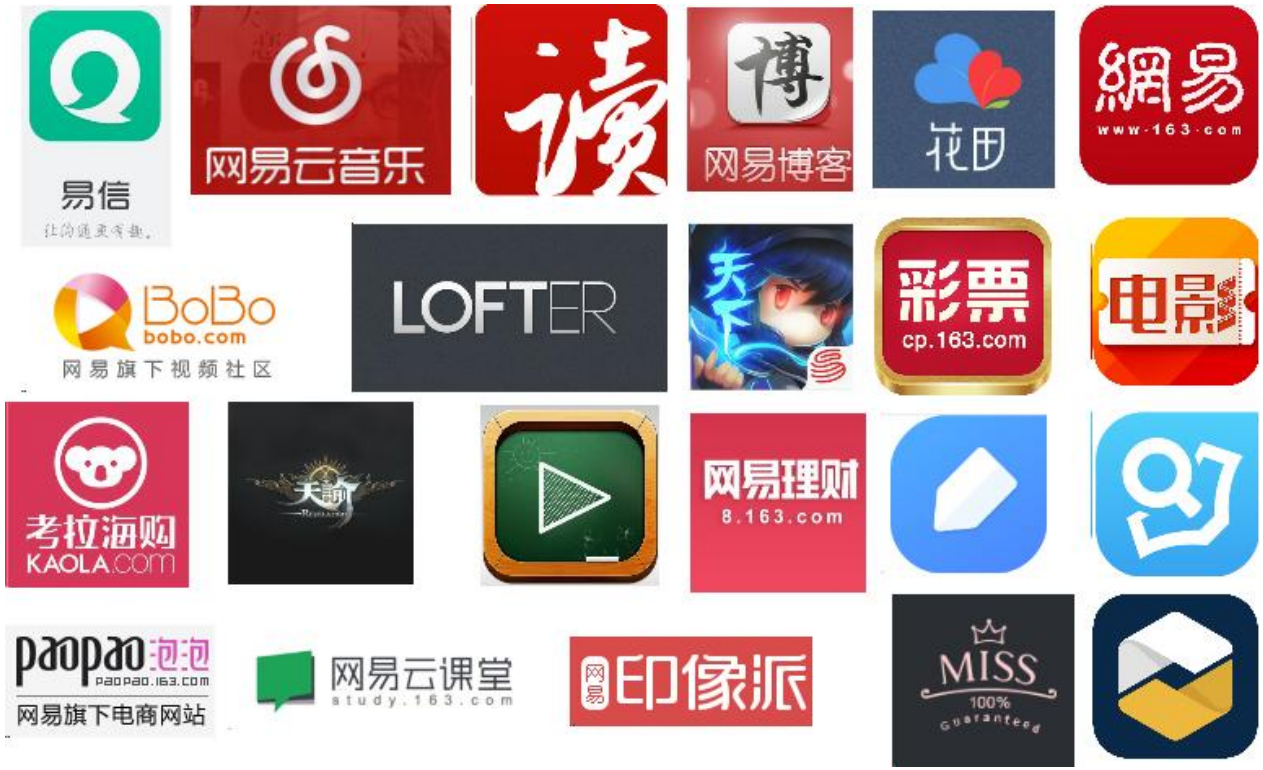
NCR特点

- 利用已有NLB、云主机、云硬盘组件
  - 实现代价小
  - SLA有保障

- Presharding + redis主从复制实现扩容
  - 优点：实现代价小，且满足应用需求
  - 缺点：运维需要有一定专业性，只适用私有云环境
  - 缺点：需要实现考虑集群容量，将来考虑采用REDIS CLUSTER

云数据库应用

- 覆盖网易互联网主要应用



小结

- 云平台和数据库一体化设计， 平台为数据库提供隔离性、性能保障、可靠性、资源弹性， 降低云数据库实现代价。
- 云数据库RDS，提供即开即用、稳定可靠、弹性伸缩的高可用数据库服务， RDS也是分布式数据库NDDB的基石。
- 分布式云数据库NDDB, 是兼容MySQL协议，具有横向扩容能力的分布式关系数据库平台， 提供大型应用无限伸缩能力。
- 云Redis服务NCR, 是高可用的托管式分布式Redis服务， 满足复杂内存数据结构存储和缓冲需求。

网易对浙大研究的帮助

- 问题来源（实际需求，用户奇思妙想）
- 大数据资源
- 设计师+工程师
- 资金

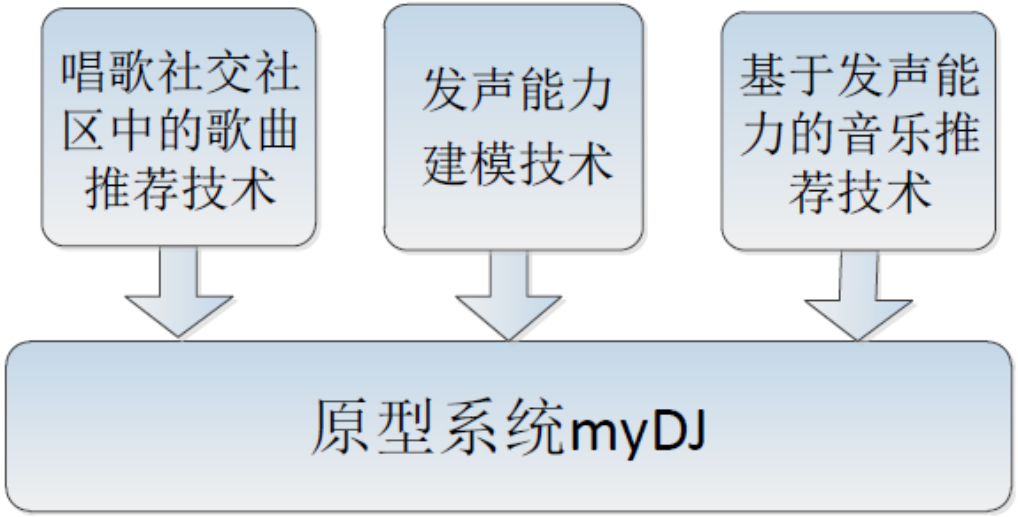
網易 NETEASE
www·163·com

浙江大学
ZHEJIANG UNIVERSITY

myDJ: 基于能力的音乐推荐（SIGIR/TKDE/ACM MM/TMM）

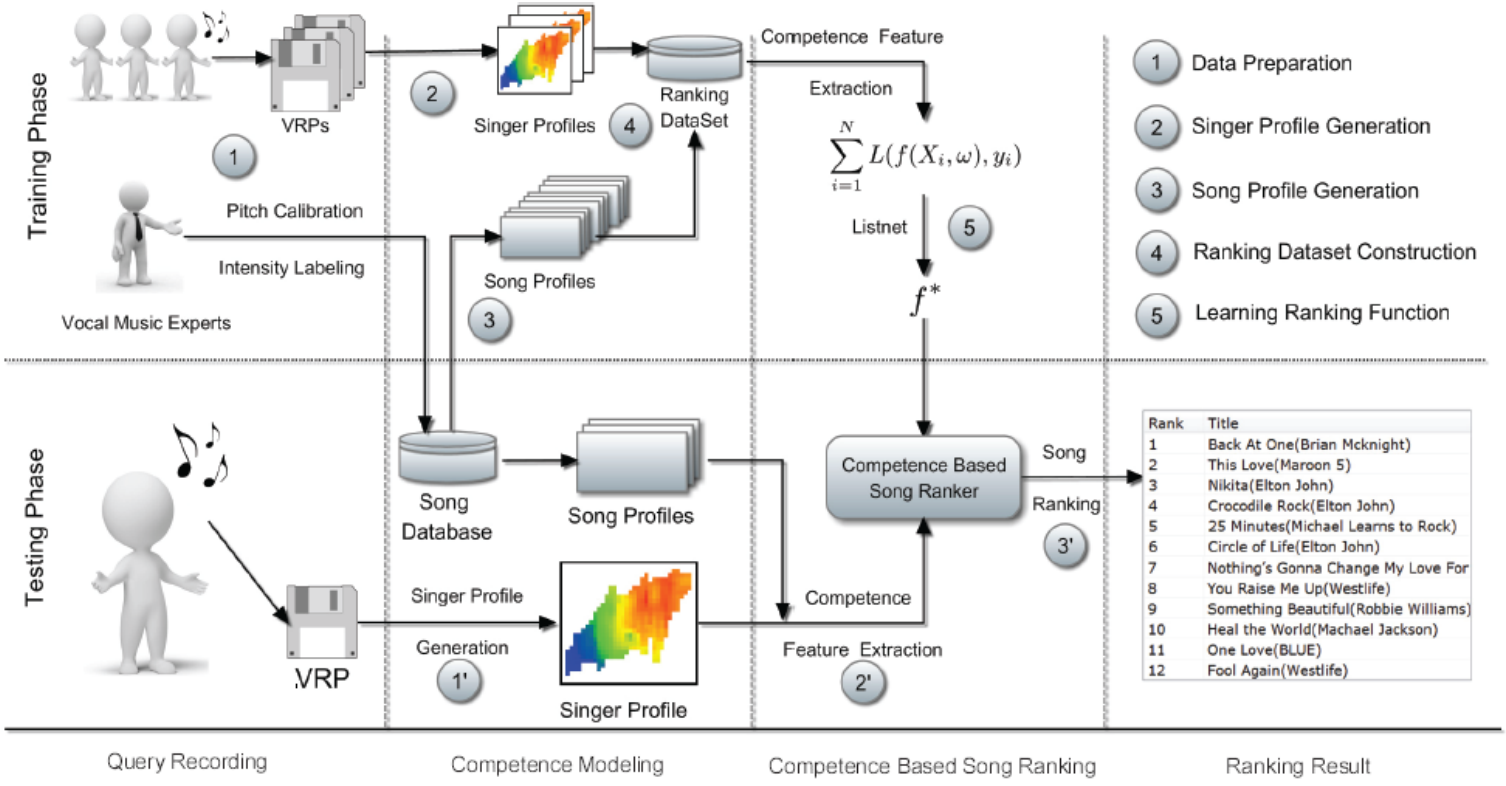
基于能力的音乐推荐

- 研究动机
  - 为了给用户推荐适合其自身演唱的歌曲
- 研究问题
  - 对唱歌社交社区中海量用户根据歌曲难度进行粗粒度的演唱歌曲推荐
  - 对有极高歌曲选择需求的用户根据其发声能力进行细粒度的演唱歌曲推荐
- 研究路线



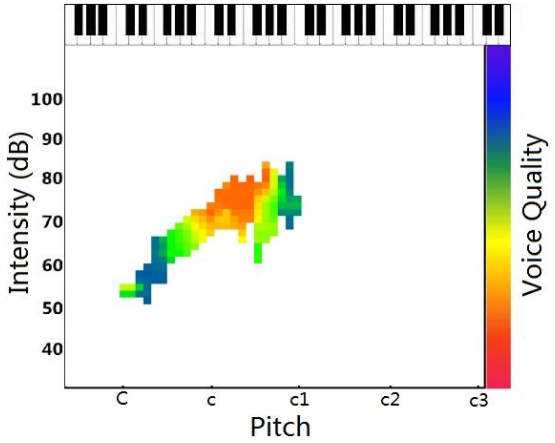
基于个体发声能力的音乐推荐

- 研究内容
  - 对个体发声能力进行建模
  - 对每首歌曲进行建模从而能用于推荐
  - 在发声能力以及歌曲之间建立查询推荐机制

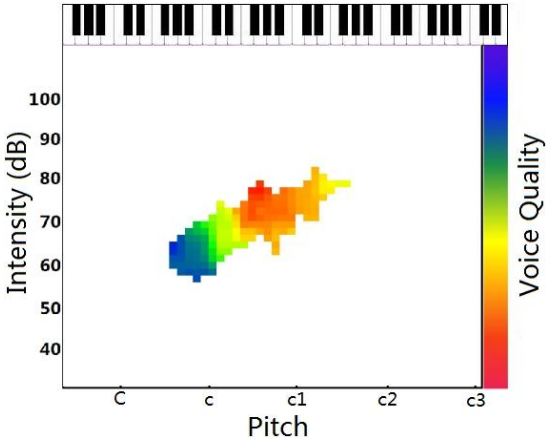


基于个体发声能力的音乐推荐

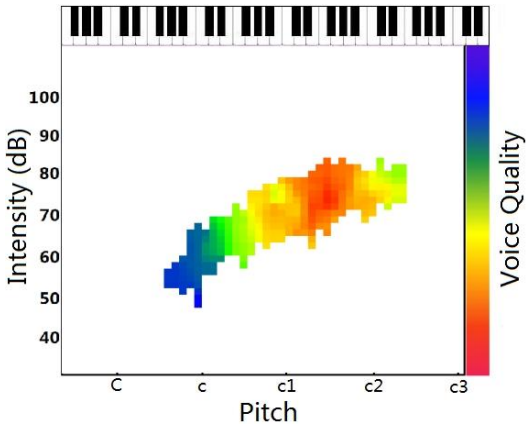
- 发声能力可视化



男低音



男中音



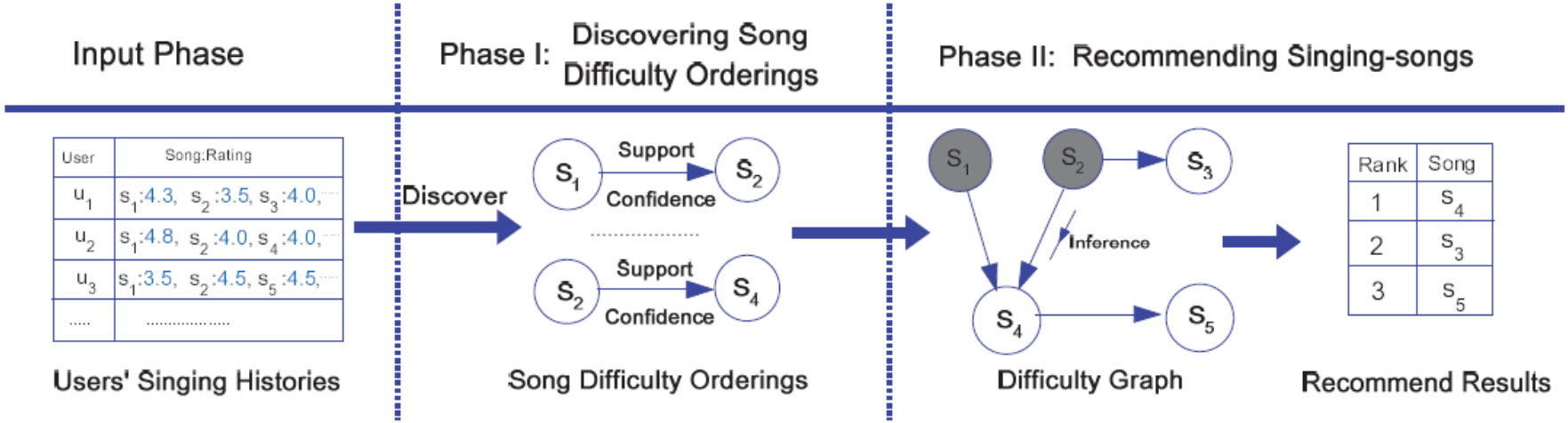
男高音

- 查询结果展示

Query	Top-5 CBSR Query Results			Top-5 PB Query Results		
	Song Name	Rank Value	Pitch/Intensity	Song Name	Rank Value	Pitch/Intensity
Male-Tenor	Never say goodbye	0.921	#c2/{2,3,4}	Apologize	0.986	#a1/{2,3}
	You and I	0.91	d2/{2,3,4}	My love	0.983	a1/{2,3}
	Guilty	0.909	#d2/{2,3,4}	My december	0.976	c1/{1,2,3}
	Tripping	0.905	#d2/{2,3}	I'll be there for you	0.971	d2/{2,3}
	Careless whisper	0.874	e2/{2}	As long as you love me	0.97	#g1/{3}
Female-Tenor	Memory	0.943	#f2/{1,2,3}	Listen	0.973	#f2/{2,3,4}
	I will always love you	0.938	#f2/{2,3}	Stay	0.972	d2/{2,3}
	Bleeding love	0.938	a2/{2,3}	Heartbeats	0.965	c2/{1,2}
	Time to say goodbye	0.921	a2/{2,3}	My heart will go on	0.953	#d2/{2,3,4}
	Hero	0.906	e2/{2,3}	Hero	0.952	e2/{2,3}

唱歌社交社区中的音乐推荐系统

技术路线

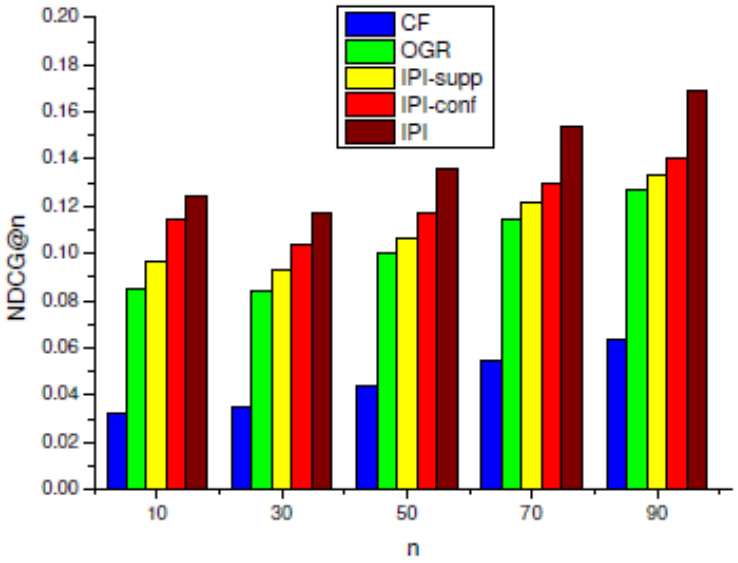


创新思想

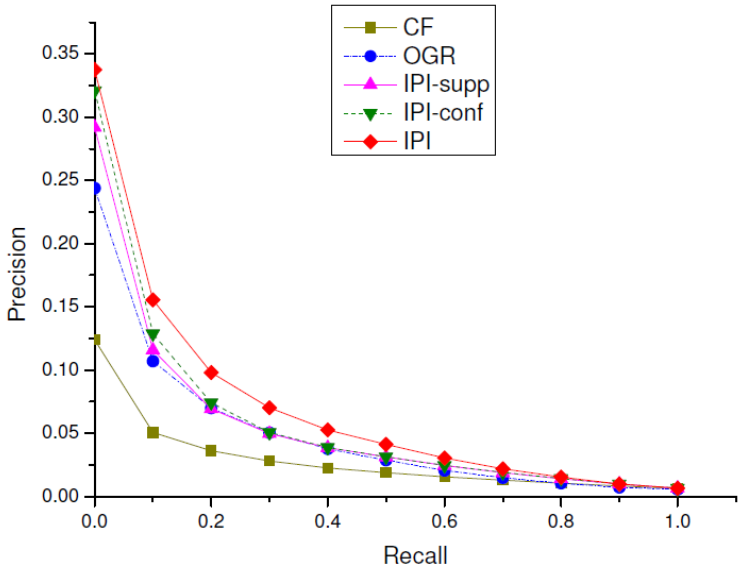
- 通过对真实用户数据进行挖掘发现歌曲难度序
- 提出歌曲难度图模型，对歌曲难度序进行概率建模
- 提出一种迭代概率推导的能力音乐推荐算法进行歌曲推荐

唱歌社交社区中的音乐推荐系统

- 实验结果
  - 推荐准确率与传统推荐算法相比提升180%以上，克服推荐系统冷启动问题
  - 推荐结果在实际用户使用中令人满意



(c) NDCG@n

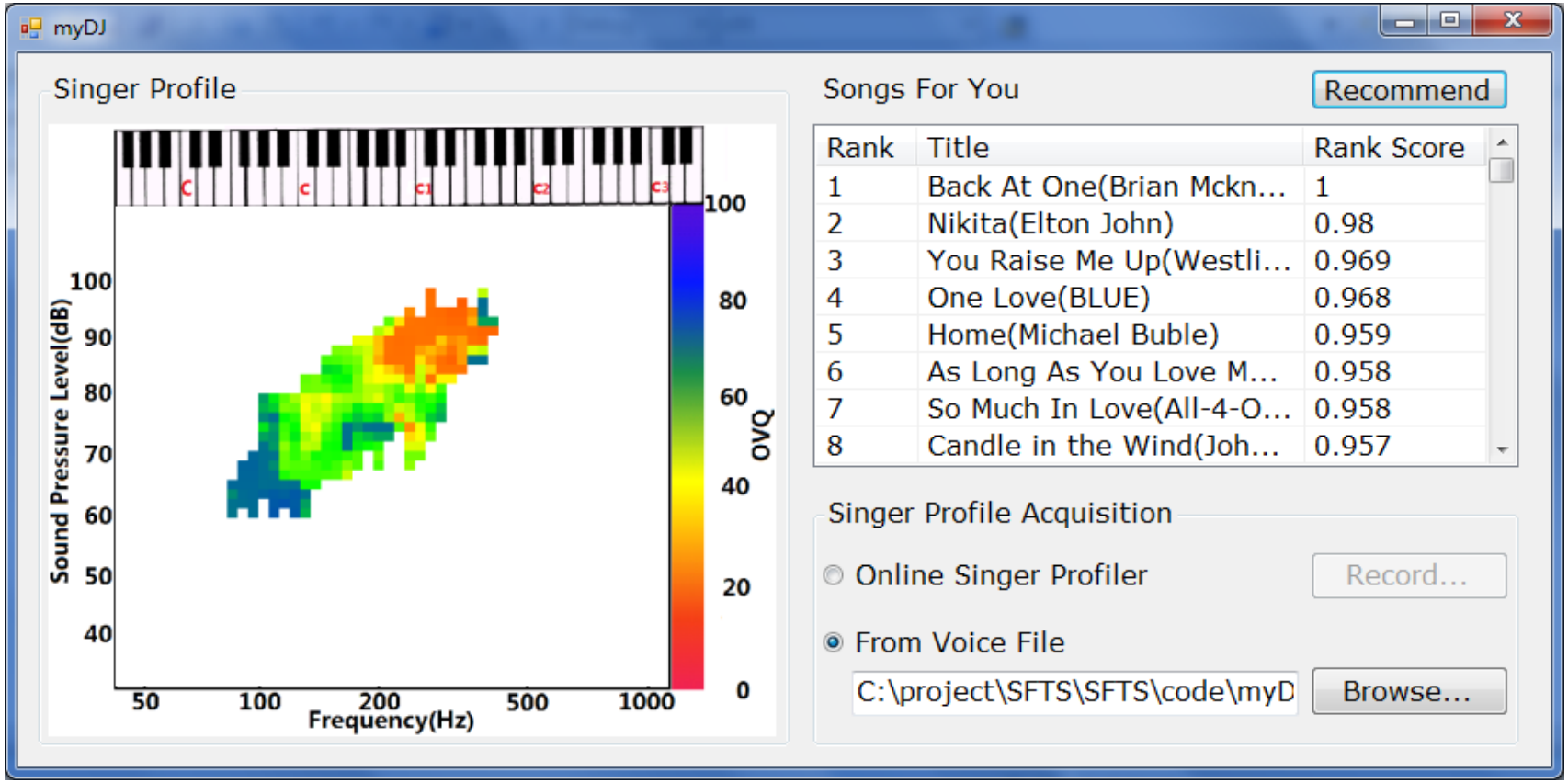


(d) Precision-recall Curve

算法准确率比较

myDJ原型系统

- 可以录制用户发声能力，实时的进行发声能力可视化以及演唱歌曲推荐



LogBase: 基于日志结构的弹性数据库（VLDB/ICDE/TKDE）

背景介绍

- 传统的数据库
  - 保留两份实际上等同的数据：数据库表、日志
  - 数据库表用来快速回答查询 （查询效率高）
  - 日志用来进行恢复处理 （更新效率高）
  - 日志和数据需要同步 （代价高）
- 然而，有一类大数据应用传统数据库不能很好的支持
  - 高速更新而查询分布不均匀：高速电商交易（每秒10万笔）、互联网有害信息甄别（数据出入速度极快、分析查询为主）

背景介绍

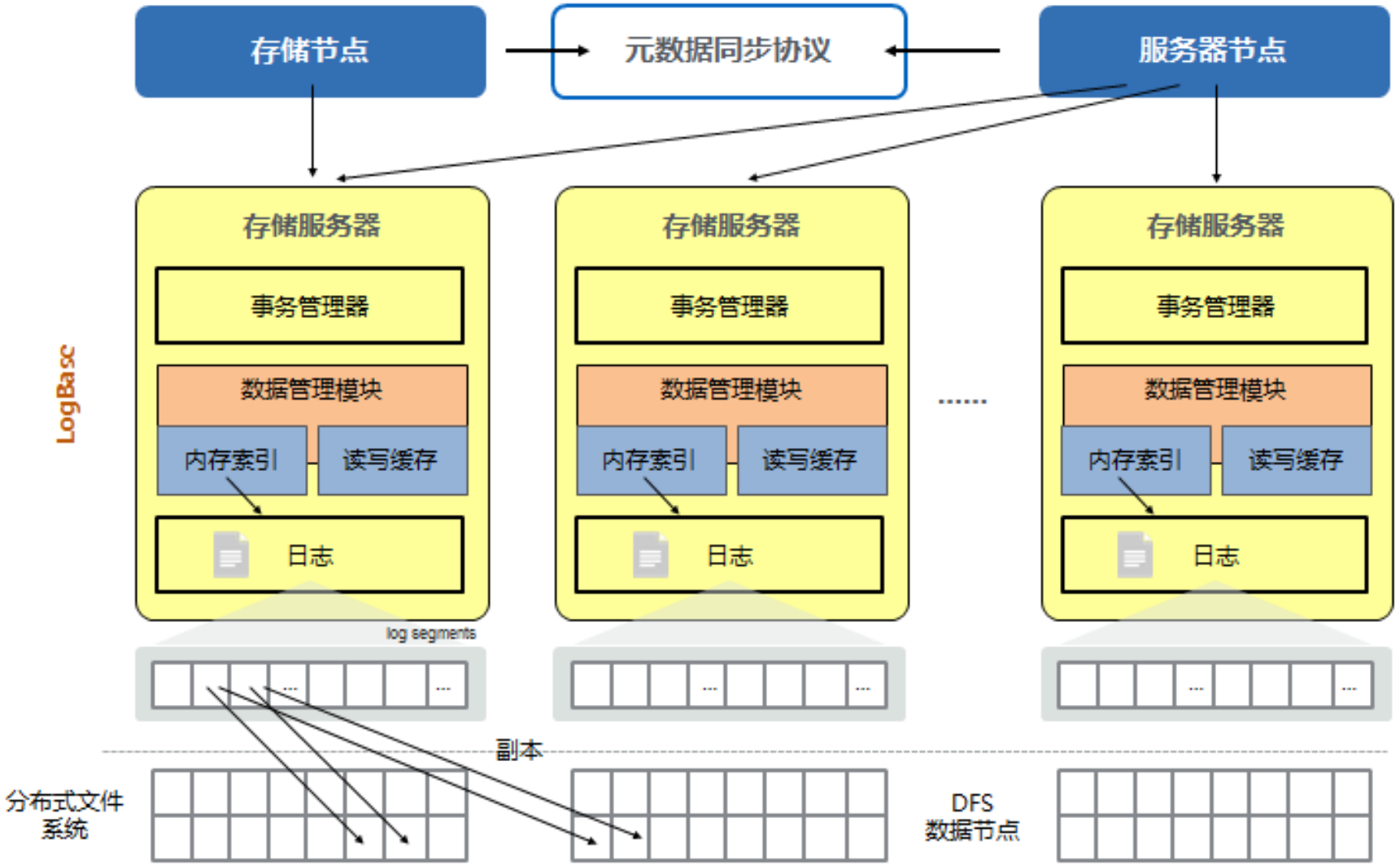
- 基于日志的恢复
  - 当数据库发生故障时，可以通过日志来进行恢复。利用日志里面记录的事务开始、结束和期间完成的修改操作，我们可以将数据库恢复到其发生故障前的状态
  - 那么也就是说，可以通过搜索日志来得到数据库中数据
- 那么可以通过日志来回答用户的查询吗？
  - 理论上完全可行
  - 实际系统中速度太慢

Log	数据库表
$<T_0$ start>	
$<T_0$, A, 100>	
$<T_0$, B, 200>	
	A=100
	B=200
$<T_0$ commit>	

基于日志结构的数据库

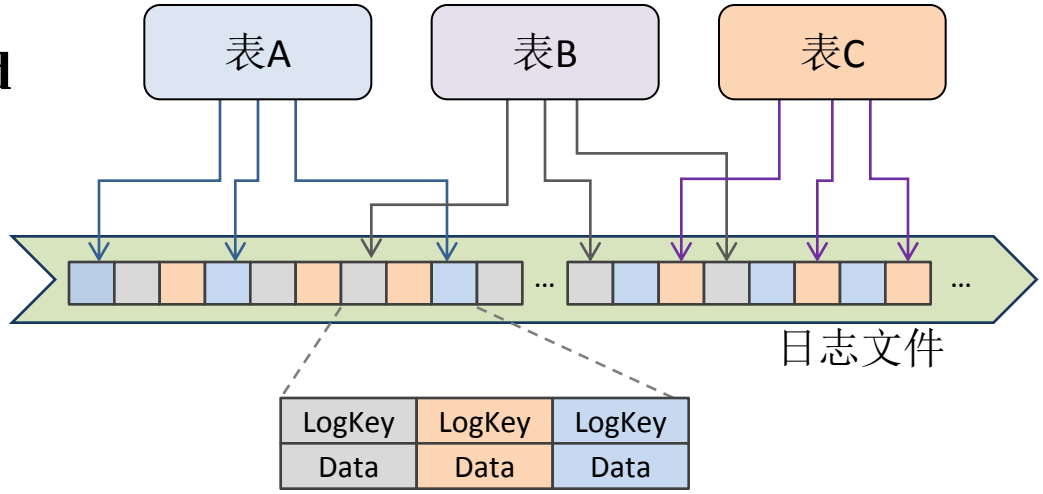
- 提出基于单一日志结构的数据库
  - 颠覆了传统数据库（日志＋数据库表）存储结构，仅仅维护一个日志文件
  - 大大降低了存储代价（不再需要数据表）
  - 大大提升了更新速度，单节点每秒10万条，多节点弹性增长
  - 新的硬件（如PCM和SSD)进一步提高了日志文件的读写性能
- 主要的挑战：如何在得到以上优点的同时，保证高效的查询处理能力？

日志结构数据库架构


存储节点
元数据同步协议
服务器节点
LogBase
存储服务器
事务管理器
数据管理模块
内存索引
读写缓存
日志
log segments
存储服务器
事务管理器
数据管理模块
内存索引
读写缓存
日志
.....
存储服务器
事务管理器
数据管理模块
内存索引
读写缓存
日志
副本
分布式文件系统
DFS数据节点

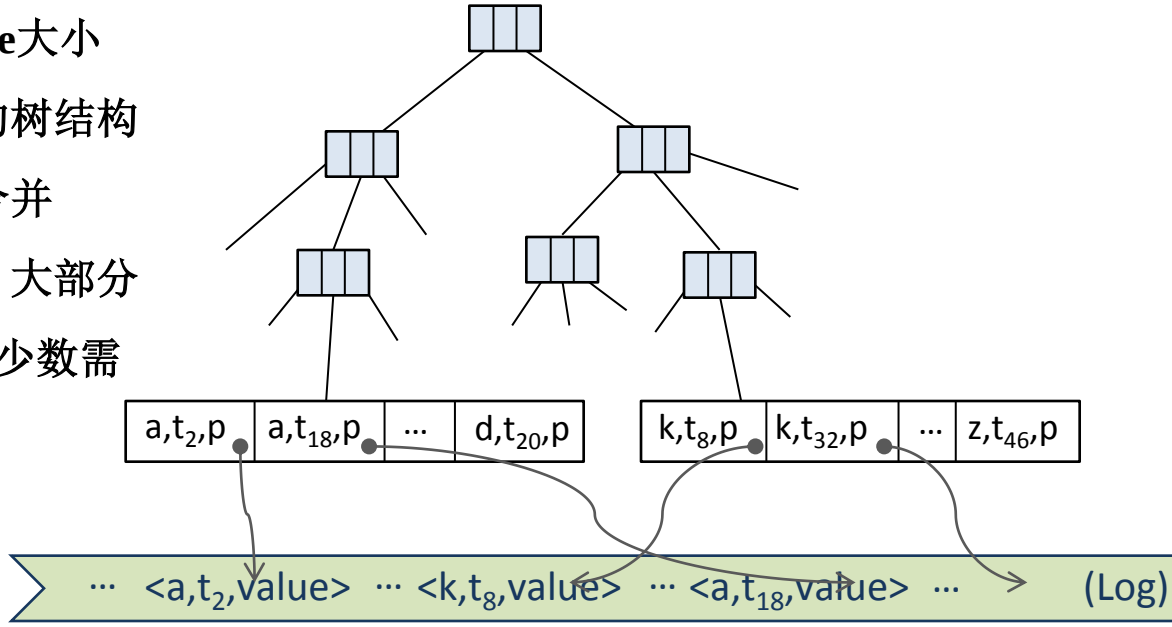
日志结构数据库架构（续）

- 数据表（日志格式）被平均分割到不同的服务器节点
- 每个服务器节点对每一个表都维护一个指针，指向日志文件的相关位置：每个表的日志先存储在内存缓存中，当达到一定大小(4M)，将被使用append的方式写入日志文件，而服务器则记录该块在日志文件的起始位置
- 每一条日志记录使用特殊的键值进行区别， 所有的表数据都存储在一个统一的日志文件
- 唯一的插入操作：append



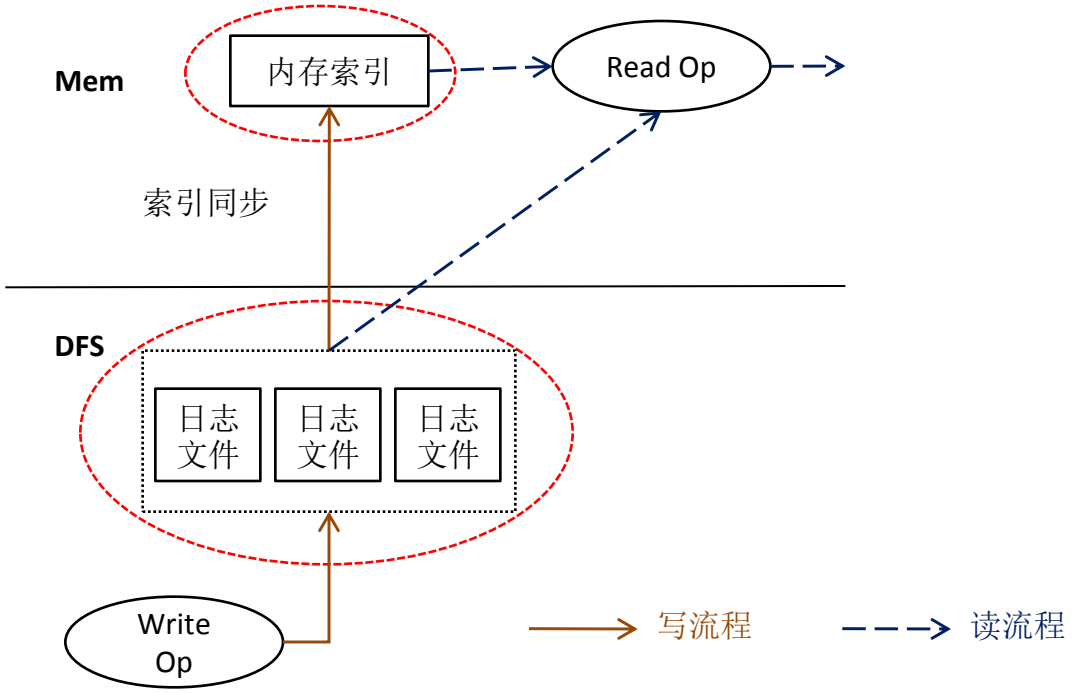
LogBase的查询机制

- 在LogBase中，为了支持高性能的查询，使用了创新的
  - 内存硬盘混合索引技术
  - 分布式日志查询技术
- 内存硬盘混合索引技术
  - 内存保留类似B树的索引结构
    - 每个叶子节点＝cache line大小
  - 硬盘维护基于日志文件结构的树结构
  - 内存索引满了和硬盘树进行合并
  - 查询时，根据查询热点模式，大部分查询可以在内存索引得到结果，少数需要搜索硬盘索引



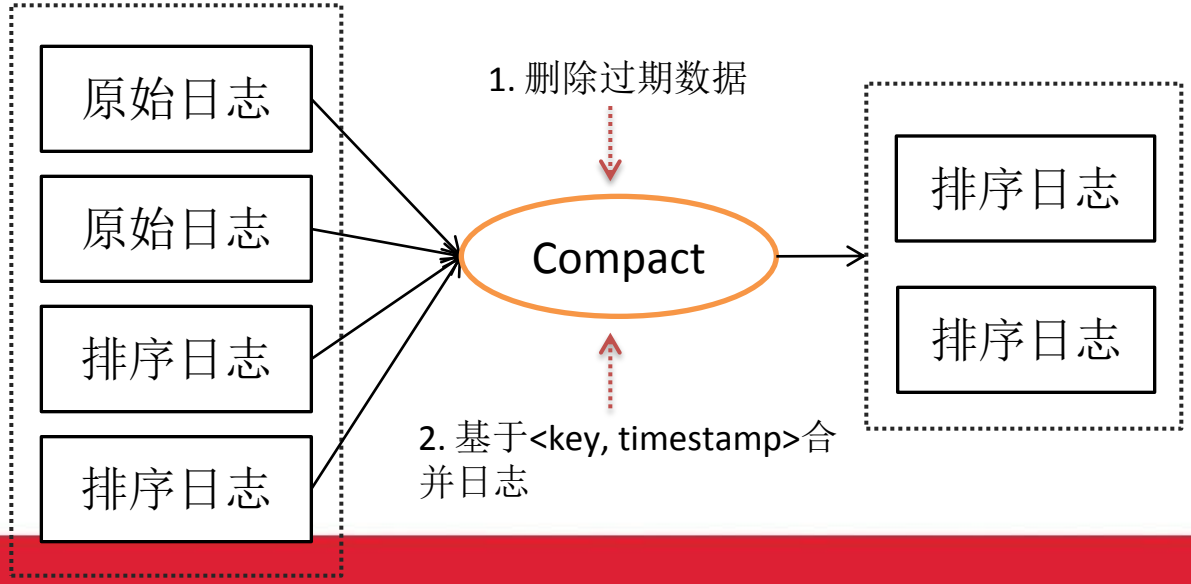
LogBase的查询机制（续）

- 分布式日志查询技术
  - 将查询按照键值分割到不同的服务器节点
  - 每个节点并行查找
  - 结果异步的返回，降低同步代价
  - 相比类似的Hbase模型，因为LogBase仅仅维护一份日志文件，大大降低了读写的I/O代价



日志压缩算法

- 日志文件大小会随着时间增长，而很多日志记录的数据实际已经被后面日志的新数据所替代，成为过时数据
- 定期调用特定的日志压缩算法来合并日志，压缩比达到1/10
  - 日志首先根据键值和时间进行排序
  - 同一键值、不同时间的日志进行合并
  - 最终产生新的按照键值排序的日志文件



谢 谢

敬请批评指正