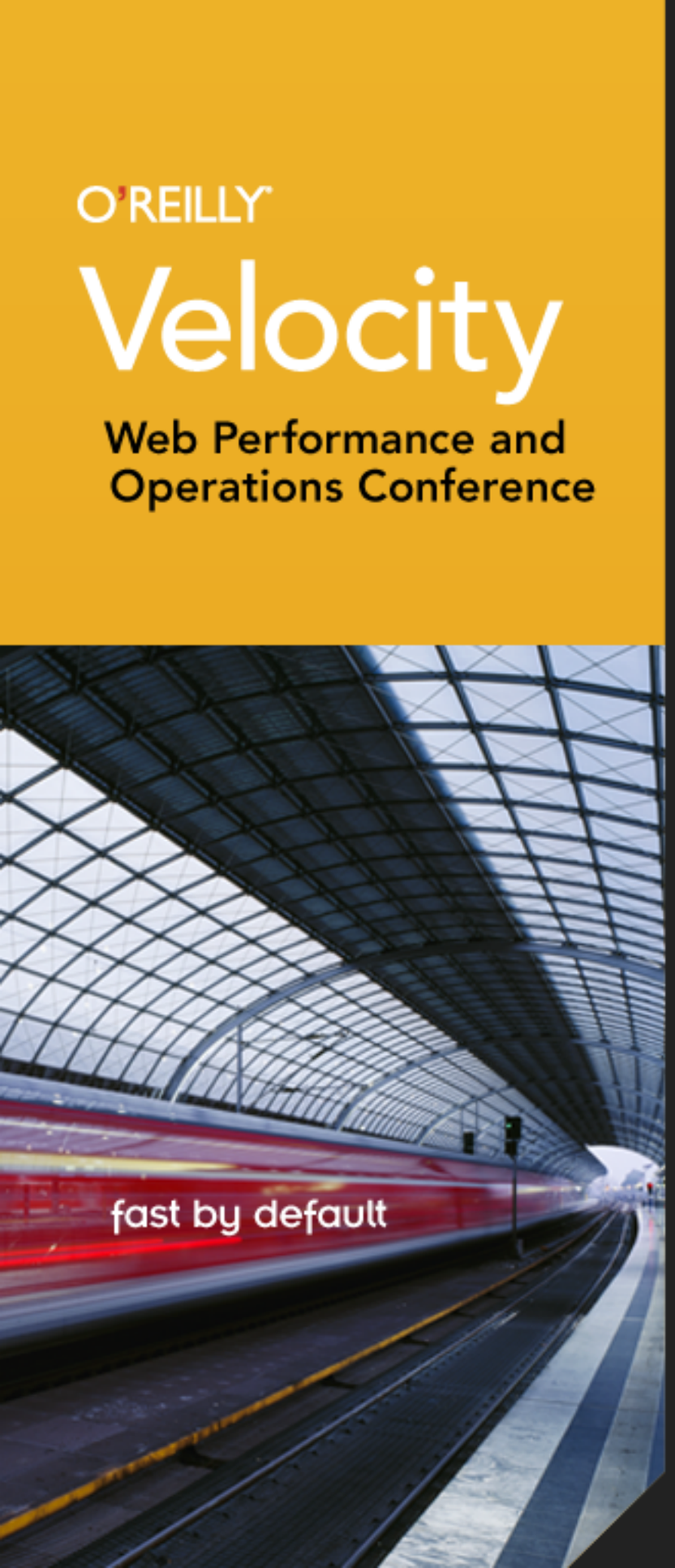


NTSE
面向Web应用特征优化的MySQL存储引擎

汪源 (Breezes/风轻扬)
网易.杭州研究院.副院长
http://wangyuanzju.blog.163.com
wangyuan@corp.netease.com

O'REILLY Velocity Web Performance and Operations Conference

简介

- NTSE: Non-Transactional Storage Engine，为一个非事务型高性能MySQL存储引擎
- 主要设计目标是高性能与高可用性
- 博客模拟基准测试性能达到纯InnoDB 10倍以上，InnoDB +Memcached的5倍以上，磁盘空间占用节约50%
- 支持在线增加/删除索引与字段
- 2008.6月开始开发，2010.5月开始在部分产品用试运行
- 90K行C++实现代码

O'REILLY
Velocity
Web Performance and Operations Conference

提纲

- 项目背景
- 系统特色与优势
- 性能基准测试
- 关键设计
- 发展方向

O'REILLY Velocity
Web Performance and Operations Conference

为什么要开发自己的存储引擎？

- 数据库是Web应用的核心组件，硬件投入不小
  - 以UGC为特征的各类Web 2.0应用需要海量的数据库存储
    - 以网易博客为例，数据库超过20T且增长迅速
  - 年数据库服务器硬件采购费用超百万，长期积累的运营费用更高
  - 由于数据库性能不足，需要额外部署大量Memcached服务器，进一步增加了硬件投入
    - 当然，不部署Memcached的硬件成本将更高
- 存储引擎是决定数据库性能的关键
  - 几乎所有数据库应用的性能瓶颈在于IO（即使部署了Memcached且命中率达90%以上）
  - 存储引擎是决定IO性能的主要组件
    - SQL解析、查询优化的CPU开销无足轻重
- 现有MySQL数据库的可用性不满意
  - 增加/删除索引与字段时表是只读的，导致表模式修改时停服务，而Web应用变更频繁

我们能做得更好吗？能！

- 我们知道现有存储引擎的不足并清楚解决之道
  - 我们主要用MySQL+InnoDB
  - 问题：不必要的事务处理机制；解决方案：将事务作为可选机制
  - 问题：页级缓存效率低；解决方案：集成行级缓存
  - 问题：空间效率低；解决方案：紧凑的记录存储格式、记录压缩、索引前缀压缩等
  - 问题：Memcached难以减轻更新负载；解决方案：集成行级缓存，通过日志保证可靠性
  - 问题：无法在线修改表模式；解决方案：借助于日志同步
- 幸运的，项目组有良好的技术储备
  - 项目组有多人（包括我自己）来自于浙江大学数据库课题组，参加过国产数据库OSCAR核心组件研发，对数据库实现技术有深入理解
- 更幸运的是，MySQL提供了插件式的存储引擎开发机制

有针对性才能做得更好

- 我们的Web应用的典型特征
  - 数据量大IO负载高而CPU负载轻：以UGC为特征的应用数据量超大但查询较简单，导致IO负载高而CPU负载轻→存储引擎应采用各类数据精简与压缩技术，用CPU换IO
  - 事务要求不高：数据通常只有拥有者自己才能修改，可多人修改的通常是访问计数等精确性要求不高的数据，对事务的要求不高。→存储引擎应将事务作为事选机制，并对访问计数等精确性要求不高的数据进行特殊优化处理
  - 数据热点明显：数据访问存在较明显热点（如据统计一天访问的博客文章只占不到5%）→存储引擎应设计行级缓存使得同样的内存能缓存更多有效记录数据
  - 模式变更频繁：应用多采用敏捷迭代开发模式，更新上线频繁，导致经常需要进行表模式修改→存储引擎应能支持在线的表模式修改

O'REILLY
Velocity
Web Performance and Operations Conference

提纲

- 项目背景
- 系统特色与优势
- 性能基准测试
- 关键设计
- 发展方向

O'REILLY Velocity
Web Performance and Operations Conference

高性能

- 集成行级缓存
  - 与页面级缓存相比，行级缓存只有真正需要的记录才会占用缓存空间，可大幅提高缓存命中率，降低IO
    - NTSE也包含页面级缓存
  - 与外置的Memcached相比，集成的行级缓存也能有效消除更新带来的负载，且无额外编码、部署开销（后将详述）
- 数据压缩
  - 大对象数据压缩：文章数据压缩到30-40%
  - 索引前缀压缩：索引大小减至30%
  - 记录数据压缩：压缩到40%左右
  - 紧凑的记录模式：InnoDB每条记录开销19字节，NTSE为0或4字节
- 超高性能UPDATE
  - 更新如计数等不要求精确的字段时只作内存更新，不产生IO
  - 定期写出

O'REILLY Velocity Web Performance and Operations Conference

高可用性

- 索引修改
  - 两种索引创建方法
    - 基于外排序的快速索引创建，表只读
    - 基于临时索引的在线索引创建，表可读写
  - 通过元信息标识的方法快速完成索引删除
- 在线字段增删
- 在线碎片整理
  - 在线记录数据碎片整理
  - 在线大对象数据碎片整理

O'REILLY Velocity
Web Performance and Operations Conference

实体ACID vs 事务ACID

- 实体：一条记录，包括记录中包含的所有大对象字段及记录所对应的所有索引项
- NTSE目前不支持事务，但对一个实体的修改保证原子性、隔离性和持久性
  - 完善的日志机制
  - 完善的并发控制机制
- 实体ACID带来
  - 极短的加锁时间，极高的并发度
  - 无死锁
- 后续版本将在实体ACID基础之上增加可选的事务支持

O'REILLY
Velocity
Web Performance and Operations Conference

提纲

- 项目背景
- 系统特色与优势
- 性能基准测试
- 关键设计
- 发展方向

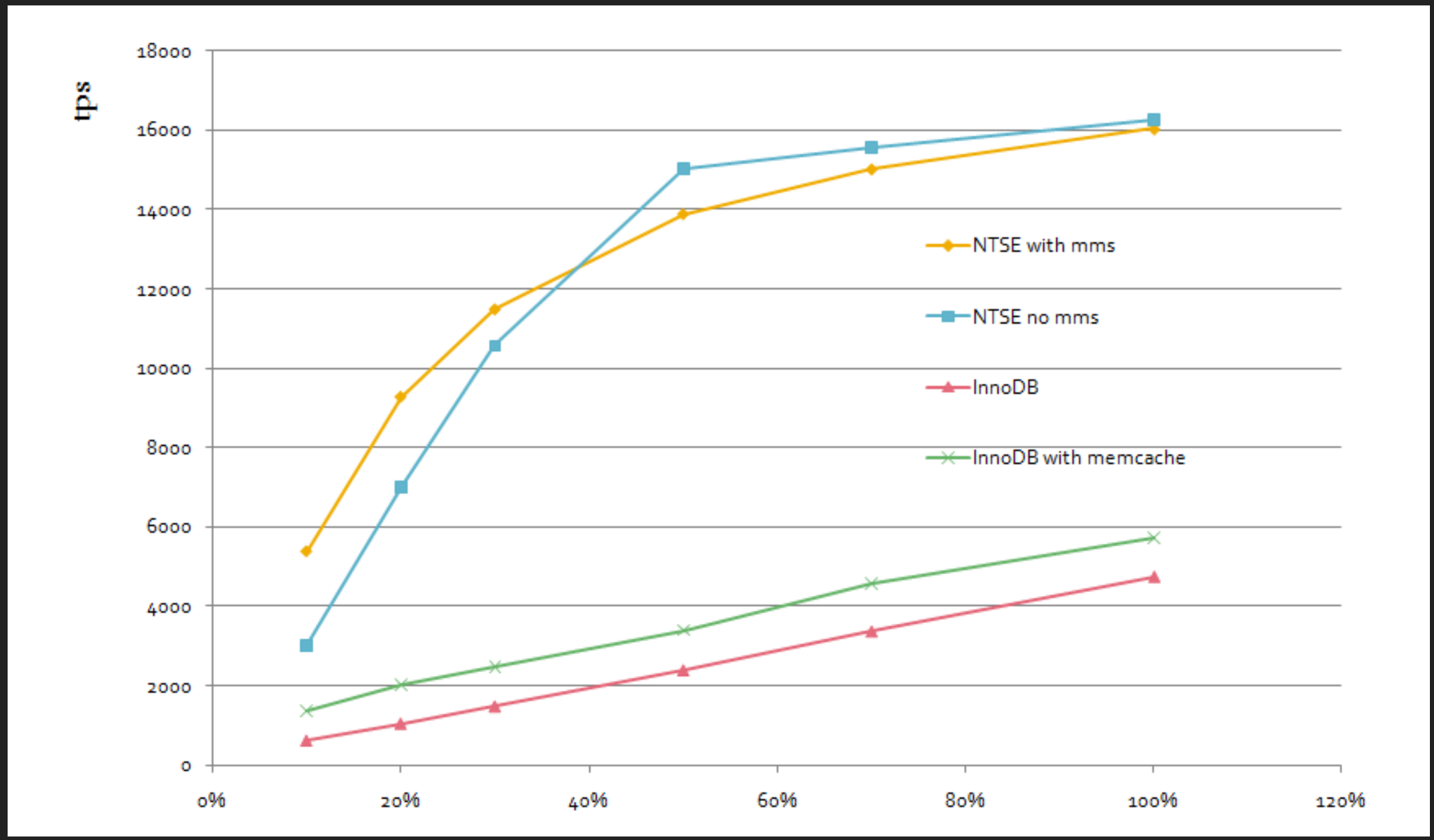
O'REILLY Velocity
Web Performance and Operations Conference

Blogbench基准测试程序简介

- 真实的模拟博客日志应用，包括表模式，表中数据的特征，操作类型，数据分布等都根据博客真实应用的统计结果设定；
- 一张表：Blog
- 七类事务：发表与修改日志、显示日志列表与日志内容、更新访问与评论计数等；
- 支持数据库与Memcached联合使用；
- 计划开源。

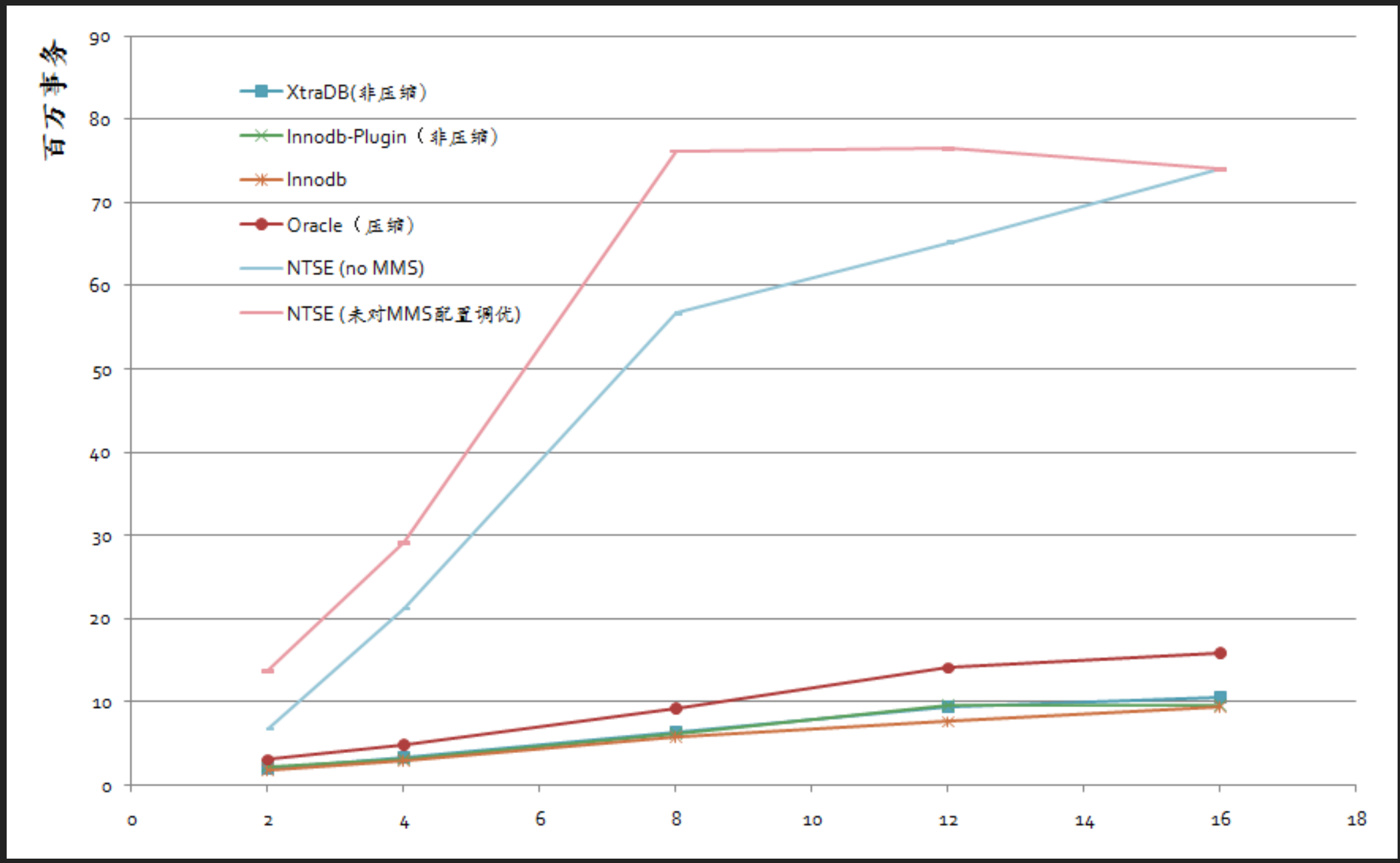
NTSE vs InnoDB

- 内存量为数据量10%的常见配置下，NTSE的性能达到InnoDB的7-13倍，为InnoDB+Memcache的3-5倍
- 内存较少时行级缓存对性能的提升作用明显



NTSE vs Oracle etc

- 同样的，NTSE的性能也远超XtraDB、InnoDB Plugin、Oracle



O'REILLY
Velocity
Web Performance and Operations Conference

提纲

- 项目背景
- 系统特色与优势
- 性能基准测试
- 关键设计
- 发展方向

O'REILLY Velocity
Web Performance and Operations Conference

MMS：记录缓存

- MMS：Main Memory Storage
- 缓存经常访问的记录或小型大对象数据
- 索引扫描访问的记录自动加载到MMS
- MMS vs 传统数据库页面缓存
  - 页面缓存空间开销小，但无法精确缓存真正需要的记录数据
  - MMS空间开销大，但只有真正经常访问的记录才占用空间
- 何时应使用MMS
  - 内存量远小于数据量（<30%）
  - 数据访问存储较明显的热点
- 记录级或页面级替换，取决于二者的访问频率

MMS更新缓存：超高性能UPDATE

- 只更新指定的某些属性时，只更新内存值不写日志，定期写出日志
- 对精确性要求不高且频繁UPDATE的数据（如访问计数），可开启MMS更新缓存功能提高UPDATE性能
- 适用条件
  - 必须启用MMS
  - 不能在任何一个索引中
  - 表必须要有主键
  - 不能是大对象类型

O'REILLY Velocity Web Performance and Operations Conference

MMS vs Memcached

- MMS类似于与数据库记录对应的Memcached
- Memcached
  - -：并发UPDATE时无法保证缓存数据与数据库相一致（除非使用数据库加锁）
  - -：不能用于缓存UPDATE，记录被UPDATE➔缓存失效
  - -：更高的开发、部署与管理成本
  - +：使用灵活，可以缓存任意数据
  - +：应用可精确控制缓存策略
- MMS
  - +：可以缓存UPDATE（有日志保证不会丢失）
  - +：不会出现数据不一致问题
  - +：与数据库集成紧密使得更高效
  - +：简化开发、部署与管理
  - -：只能缓存与记录对应的数据
  - -：只能统一使用LRU替换策略

O'REILLY Velocity Web Performance and Operations Conference

MMS的一些设计经验（1）

- 设计一个性能优异的记录级缓存系统并不容易
- MMS的一些设计经验
  - 缓存更新操作
    - 基本策略：更新命中缓存时只记LOG，并进行定期检查点保证恢复时间
    - 主要挑战：定期检查点时产生大量IO
      - 基本方法
        - 脏记录按RID排序之后再写出，脏页面将PID排序之后再写出将“乱序”IO转化为“单向跳跃性”IO
  - 刷脏记录改进策略
    - 不好：限制脏记录数→最优值难以确定且导致增加无谓的IO
    - 好：只写出对应页面在页面缓存中的脏记录，其余脏记录只写日志以保证能恢复→刷脏记录只带来对日志的顺序IO
  - 刷脏页改进策略
    - Smart sleep：刷脏页时增加一些sleep操作，使得系统恰好能够在指定的检查点周期完成检查点操作，不要写得太快

MMS的一些设计经验（2）

- MMS的一些设计经验（续）
  - 支持不同大小对象的替换算法：根据最近访问时间戳估计访问频率，替换频率低的记录或页面
    - 适用于页级缓存的LRU和CLOCK算法无效
    - 尽量尽少替换算法带来的空间开销
      - 2字节页内LRU+页级堆→同级别记录全局LRU
  - 记录缓存与页面缓存的空间配置
    - 人工难以确定，需可在线调整
    - 页面缓存的“热身”速度远高于记录缓存，需可临时“借用”记录缓存空间
  - 某些操作只作尝试不强制要求成功
    - 如将被访问记录插入到记录缓存中的操作在尝试一段时间不成功时放弃努力
    - 一种便于实现且能防止操作由于并发原因被长时间阻塞的方法

O'REILLY Velocity Web Performance and Operations Conference

数据压缩

- 整型数据压缩
  - 转化为大端优先字节序并去除前导0/0xFF（对负数）字节
  - 压缩后可使用memcmp快速排序以优化索引搜索效率
    - 即便对于多个属性和负数
- 索引前缀压缩：压缩比30%
  - 索引页划分为多个MiniPage，最多15项键值为一个MiniPage，每个MiniPage中只有第一个键值是完整的，其余键值只存储与前一键值的不同后缀
- 大对象压缩：压缩比35%
  - 使用LZO压缩库
- 记录数据压缩：压缩比40%
  - 表全局字典，最多64K个字典项
  - 字典通过采样的方式构建
  - 针对字节流，不考虑属性边界，因此可压缩多属性组合及属性部分值，压缩比高
  - 以可配置的属性组为单位，CPU与压缩比的折衷

O'REILLY Velocity
Web Performance and Operations Conference

并发控制

- 如何提高并发度
  - 减少加锁操作
  - 缩短锁持有时间
  - 减小加锁粒度
- 一般数据库
  - 对记录加锁，持有至语句或事务结束
- NTSE
  - 对记录加锁，持有至单条记录处理结束
  - 索引需要额外的页级锁，但持有时间极短
    - 先操作所有唯一性索引，再操作非唯一性索引
    - 唯一性索引的页级锁持有至所有唯一性索引操作结束
      - 过多的唯一性索引可能降低并发度，幸运的是通常只有一个
    - 非唯一性索引的页级锁持有至操作阶段结束
      - INSERT/DELETE只有一个阶段
      - UPDATE有DELETE和INSERT两个阶段
    - 页级锁是在读入页面之后再加，因此持有期间通常无IO操作

O'REILLY Velocity Web Performance and Operations Conference

其它

- 索引智能分裂系数
  - 插入操作在页面中的平均插入位置决定分裂时的左右比例
  - 很好的适应从纯增/降序到纯随机之间的任何一种有序性特征，既提高了索引页面利用率，又不会导致页面分裂大量增加
- 在线模式修改
  - 一般流程
    - 对表进行“模糊”的表扫描，拷贝到临时表，建立新老RID的对应关系（表可读写）
    - 回放拷贝开始后的日志，根据RID对应关系应用于临时表
      - 进行多轮直到只剩很少量日志为止
    - 短期锁定表进行最后一轮回放并替换
  - 不能读写时间通常<10秒
- 日志机制
  - 一般数据库：需要记录前后像（可能会UNDO和REDO）
  - NTSE：只需要记录后像及部分索引的前像（绝大多数情况下只需要作REDO）

O'REILLY
Velocity
Web Performance and Operations Conference

提纲

- 项目背景
- 系统特色与优势
- 性能基准测试
- 关键设计
- 发展方向

O'REILLY Velocity
Web Performance and Operations Conference

NTSE 2.0→TNT

- NTSE 2.0将重命名为TNT，代表Transactional or Non-Transactional，核心是增加可配置的事务支持
- 功能简述
  - 每张表可单独配置为启用或不启用事务
  - 可在线修改表是否启用事务，且不会导致数据拷贝
  - 事务大小受制于内存，面向事务普遍短小的在线应用设计
- 实现思路
  - 多版本技术
  - 记录：新版本存储于堆中，旧版本统一存储在版本池
  - 索引：索引的最新修改缓存于内存，定期合并到外存索引
  - 根据索引数据即可确定版本可见性，支持Index only
- 原NTSE模块基本不需要修改

O'REILLY Velocity Web Performance and Operations Conference

结语及一些相关的想法

- 细粒度缓存非常有效，但Memcached并不是一种最有效的机制
- 压缩非常有效
- 事务应作为一种可选的而非必需的机制
  - 有大量应用不需要事务，又有大量应用要借助于事务
  - 无论是强制使用事务还是根本不提供事务都不是好的方案
  - 最佳方案是可配置且可灵活转换
- 实体原子性是比事务ACID更基础的有用性质
  - 在一个支持实体原子性的系统之上，借助于多版本技术实现事务并且实现两者的共存与相互转换是可行的
  - 如同在CAS指令基础之上使用各类锁机制
- 需要放松对Consistency的要求吗？
  - 我们喜欢Consistency带来的开发便利
  - Eventual Consistency
    - EC将增加开发工作量，导致其在组织内难以推广
  - CAP
    - 在单个数据中心中，不超过1000台服务器时，P出现问题的概率极低，没有必要为此牺牲C
    - 在不超过100台服务器时，A出现问题的概率也极低（远低于应用升级带来的计划内的A问题），没有必要为此牺牲C