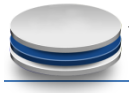
SACC

2010系统架构师大会

百度分布式数据库

Baidu 百度



简介

- ✓ 谁使用MySQL?
- ✓ 为什么使用MySQL?
- ✓ 问题
  - 性能
  - 数据规模伸缩
  - 功能特性
  - 服务化
  - 自动化

单节点

目标：

- 响应时间
- 吞吐
- 解放大部分产品线
- 节约资源
- 分布式数据库需求

功能：

- Fulltext
- Snapshot
- Optimized alter
- 其他

单节点性能

- ✓ 性能
  - – QPS (读 / 写)
  - – 响应时间 (平均 / 长耗时)
  - – 数据规模
- ✓ 问题
  - – 随机读
    - • 存储引擎cache & 系统cache
  - – 随机写 (LRU / checkpoint ...)
    - • buffered write
    - • ordered write
    - • 长耗时绝大部分的请求响应时间在1ms以内
- ✓ IOPS是读操作和写操作的瓶颈!

我们的优化结果

- ✓ Vs 硬盘 (sas 10k)
  - QPS 提升 700%
- ✓ Vs SSD (FTL Optimized)
  - QPS 提升 250% ，长耗时减少 95%
  - 可用空间增多 & 使用寿命增加
  - 通用型优化，读为主应用及写为主应用均适合

- ✓ 对应用完全透明，使用方式和以前一样

- ✓ 2007年百度尝试Flash， 2008年百度网页搜索全面使用Flash
- ✓ 2008年MySQL尝试使用Flash，2011年百度MySQL全面使用SSD

IO设备特性

- ✓ IO 设备 (硬盘 & SSD & 内存)
  - 顺序写、顺序读、随机写、随机读
  - 响应时间
  - 带宽
  - 访问密度
  - 价格

- ✓ Tape is dead, disk is tape, flash is disk, ram locality is king.

—— Jim Gray

SSD Vs 硬盘

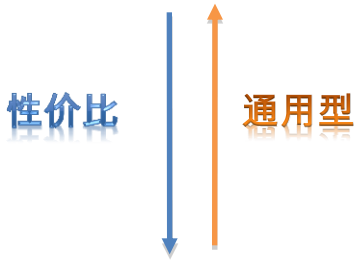
	16K随机读	16K随机写	16K顺序写	1M顺序写	1M顺序读
SSD	0.3ms	0.54ms	0.1ms	3.8ms	1.77ms
硬盘	5.9ms	1.08ms	0.15ms		

SSD 16K随机读比硬盘提升 1860%
SSD 16K随机写比硬盘提升 100%
SSD 16K顺序写比硬盘提升 50%
SSD 16K顺序写比其随机写提升 440%
SSD 1M顺序写比64次16K随机写提升 800%
SSD 1M顺序写比64次16K顺序写提升 68%
SSD 1M顺序写比1次16K顺序写提升 3700%

如何针对这些数据来设计系统 ？

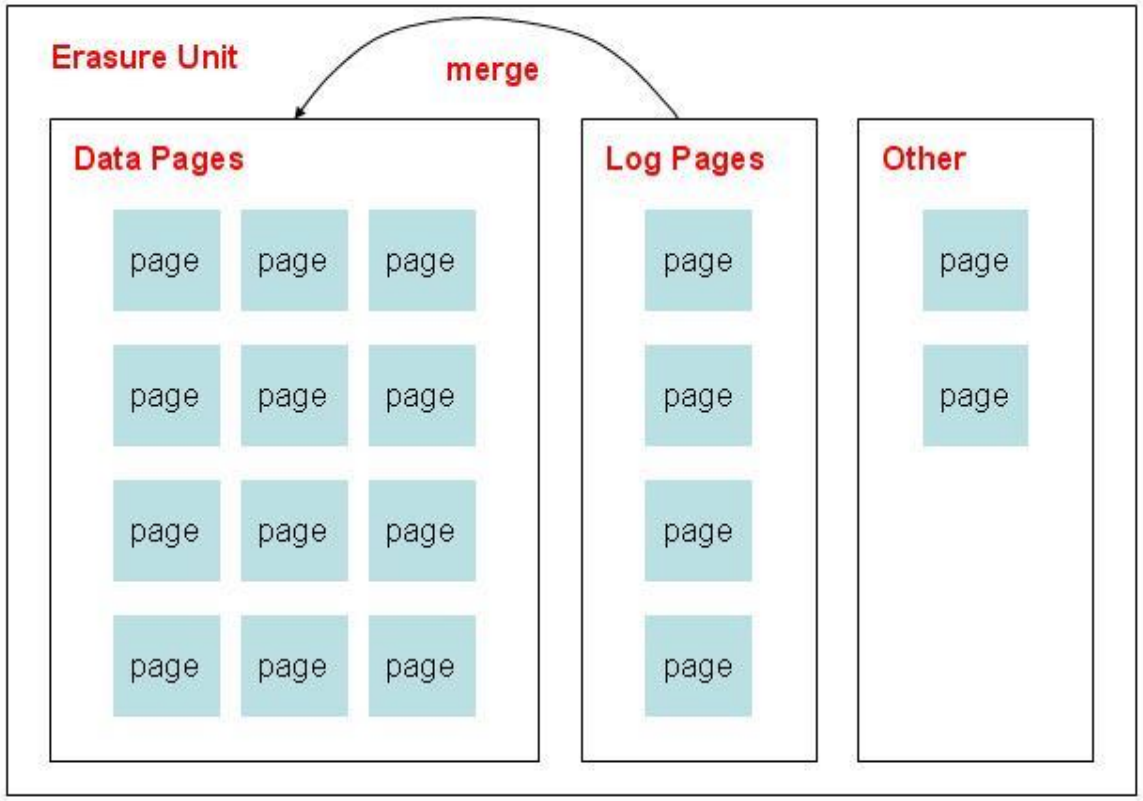
优化手段

- ✓ FTL
  - in-page logging
  - 其他
- ✓ 文件系统
  - l2fs,btrfs, zfs ...
  - BFTL
- ✓ Kernel
  - flashcache
- ✓ 存储系统逻辑
  - append write
  - random read
  - merge



FTL

- ✓ IO 模型
  - 随机写
  - 随机读
- ✓ In-page logging
  - 20% log 空间
  - 75% raid5
  - 60% 使用率

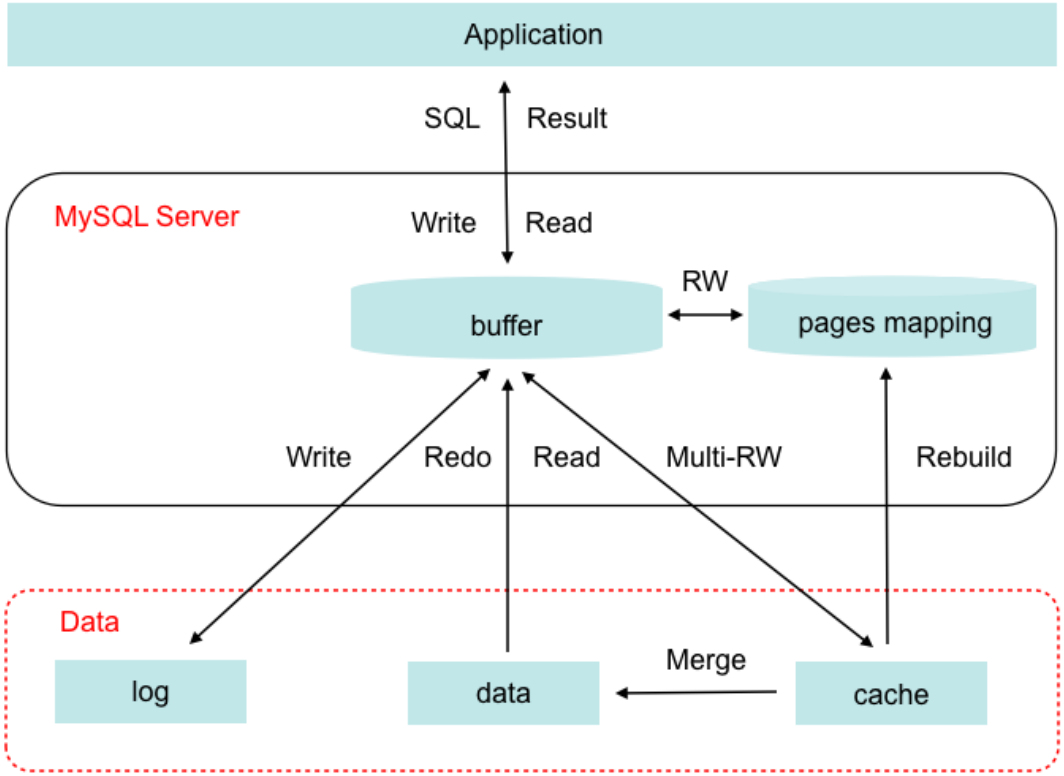


存储系统逻辑

- ✓ SSD/硬盘作为SSD/硬盘的写cache
- ✓ SSD作为硬盘的读cache
  - – SSD作为innodb buffer pool的二级读cache
- ✓ 远程memory作为innodb buffer pool的二级读cache
- ✓ 不同IO模型分离
  - – 文件 / 设备 / IO模型转化 / 分离

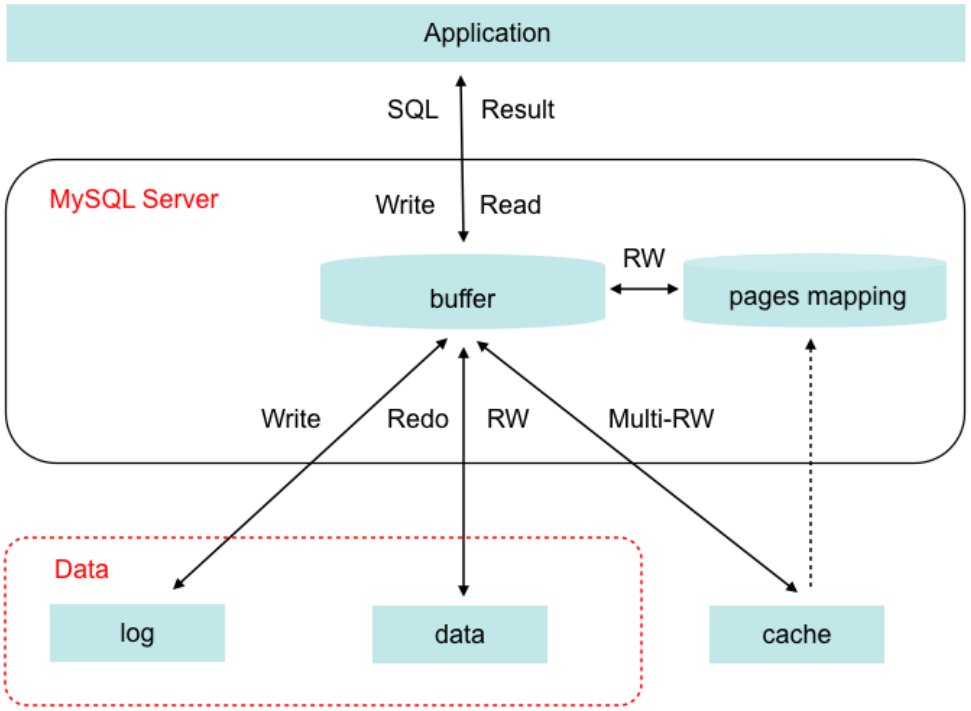
写cache

- ✓ IO 模型
  - 顺序写 (提升800%)
  - 随机读
- ✓ Merge
- ✓ Pages mapping
  - mem: ssd = 1 : 350
- ✓ Multi-Write
  - 提升68%
- ✓ 写瓶颈
  - iops -> 吞吐
- ✓ 读瓶颈
  - iops -> iops



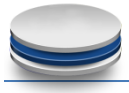
写cache & 读cache

- ✓ IOPS Vs 吞吐
- ✓ 读Cache Vs 写Cache
- ✓ 性价比
- ✓ 预热
- ✓ 可维护性
- ✓ 数据完整性 & 一致性
- ✓ 透明 & 通用
- ✓ Nand flash Vs Nor flash
  - 100ns、写性能、价格、容量、直接寻址
- ✓ Snapshot (Redirect write)
- ✓ Btree (log-based 38x?) / Btree patch compaction



其他

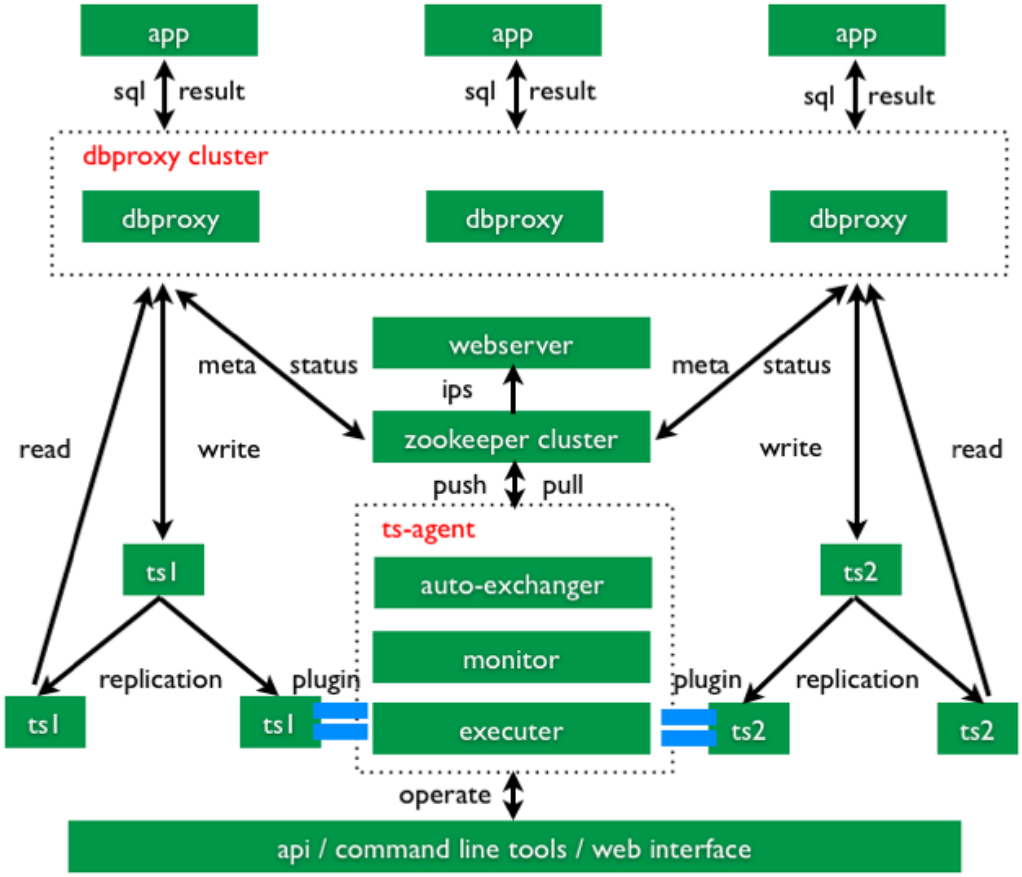
- ✓ 故障
  - ECC
  - SLC
  - Raid / Rebuild
  - 架构
- ✓ 继续优化该版本
  - read cache / btree patch compaction
- ✓ 单节点
  - 500G ~ 1T
  - 功能特性增强
    - snapshot
    - online alter table



分布式

- ✓ 产品定位
  - 尽量保证数据库特性，提升数据规模
  - 线上低延迟的访问
  - 满足具有一定复杂关系的数据操作
- ✓ 设计原则
  - 应用访问方式不变
  - 应用知道数据逻辑分布
  - 不同访问模式提供的功能不同
  - 自动发现/人工决定/自动处理

总体架构


app
app
app
sql
result
sql
result
sql
result
dbproxy cluster
dbproxy
dbproxy
dbproxy
webserver
ips
zookeeper cluster
meta
status
meta
status
read
write
write
read
push
pull
ts-agent
auto-exchanger
monitor
executer
ts1
ts2
replication
plugin
plugin
replication
ts1
ts1
ts2
ts2
operate
api / command line tools / web interface

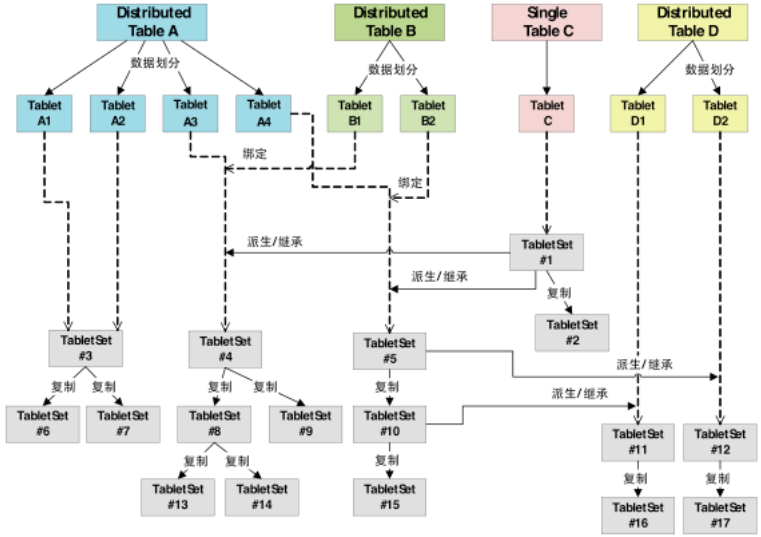
访问模式

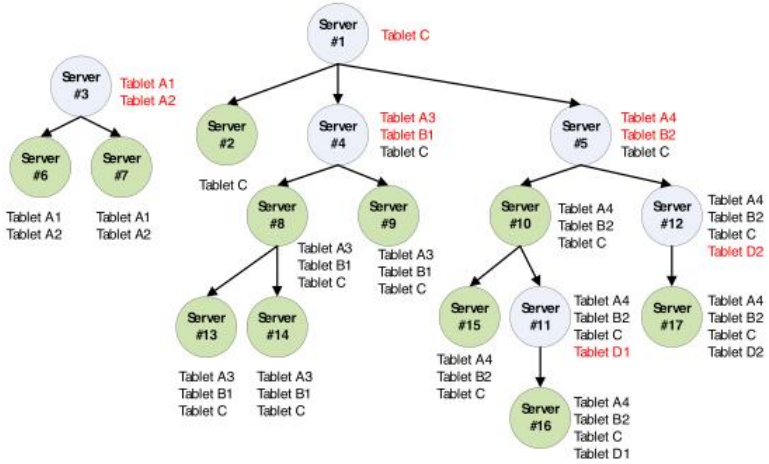
- ✓ Scan & Search
- ✓ 基于Partition Key
  - 单表单机
  - 单表多机
  - 多表单机
  - 多表多机
- ✓ 不基于Partition Key
  - 单表
  - 多表

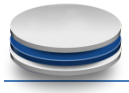
数据划分

- ✓ 范围划分
- ✓ 散列取模划分
- ✓ 枚举划分
- ✓ 时间划分
- ✓ 组合划分

- ✓ Binding
- ✓ 继承







负载均衡 & 数据迁移

- ✓ 负载均衡
  - 目标
  - 衡量标准
  - 定期汇报
- ✓ 数据迁移
  - 负载均衡
  - 高可用

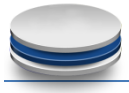
数据一致性

- ✓ dbproxy与zookeeper
- ✓ zookeeper内部数据一致性
- ✓ 同一tablet不同副本之间的数据一致性(异步/半同步)
  - 最终一致性
  - 会话一致性
- ✓ 不同tablet之间的数据一致性
- ✓ 分布式事务
  - 单机事务 + 最终一致性

系统可用性 & 可靠性

- ✓ 多副本
- ✓ 部署
- ✓ 切换

- ✓ dbproxy
- ✓ zookeeper
- ✓ ts
  - slave ts down / master ts down / tablet down / all tablet down
  - auto-exchanger / 盘柜
  - mq

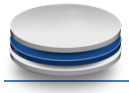


可扩展性

- ✓ dbproxy
- ✓ zookeeper
- ✓ table
  - 预防扩容
  - 读性能引起 (QPS / Latency)
  - 写性能引起
  - 自动扩容
  - 半自动扩容
  - 合并、分裂

其他 & 开源

- ✓ 其他
  - 接口 / 权限
  - 备份
  - 监控
  - 混合运维
  - 计算
  - 工具
- ✓ 开源
  - 单机性能优化
  - dbproxy



Thanks!

Q & A