Performance: tips and traps

Wenliang ZHANG
https://zedware.github.io

Outline

- 引言
- 性能改进的原因
- 针对特定场景的优化
- 针对通用场景的优化
- 如何识别陷阱
- 性能调优
- 实际例子

引言

“性能就像个小姑娘，想怎么打扮就怎么打扮”

Anonymous

引言

- 厂商发布新产品时的大多会有性能提升的宣传
  - Intel发布新款CPU
  - Intel发布新款SSD
  - Oracle发布数据库：10倍提升
  - MySQL发布新版本：百万QPS
  - Hadoop、JDK、手机等都有类似的宣传
- 哪些是货真价实的提升，哪些有很多水分？
  - 外行看热闹，内行看门道
  - 不要被公关稿蒙蔽

In spite of earlier, more modest statements, Intel's 8th-generation Coffee Lake processors are up to 45% faster than their Kaby Lake predecessors. This explains the requirement for users to upgrade to a new motherboard – even though some clever modders found a way around that.

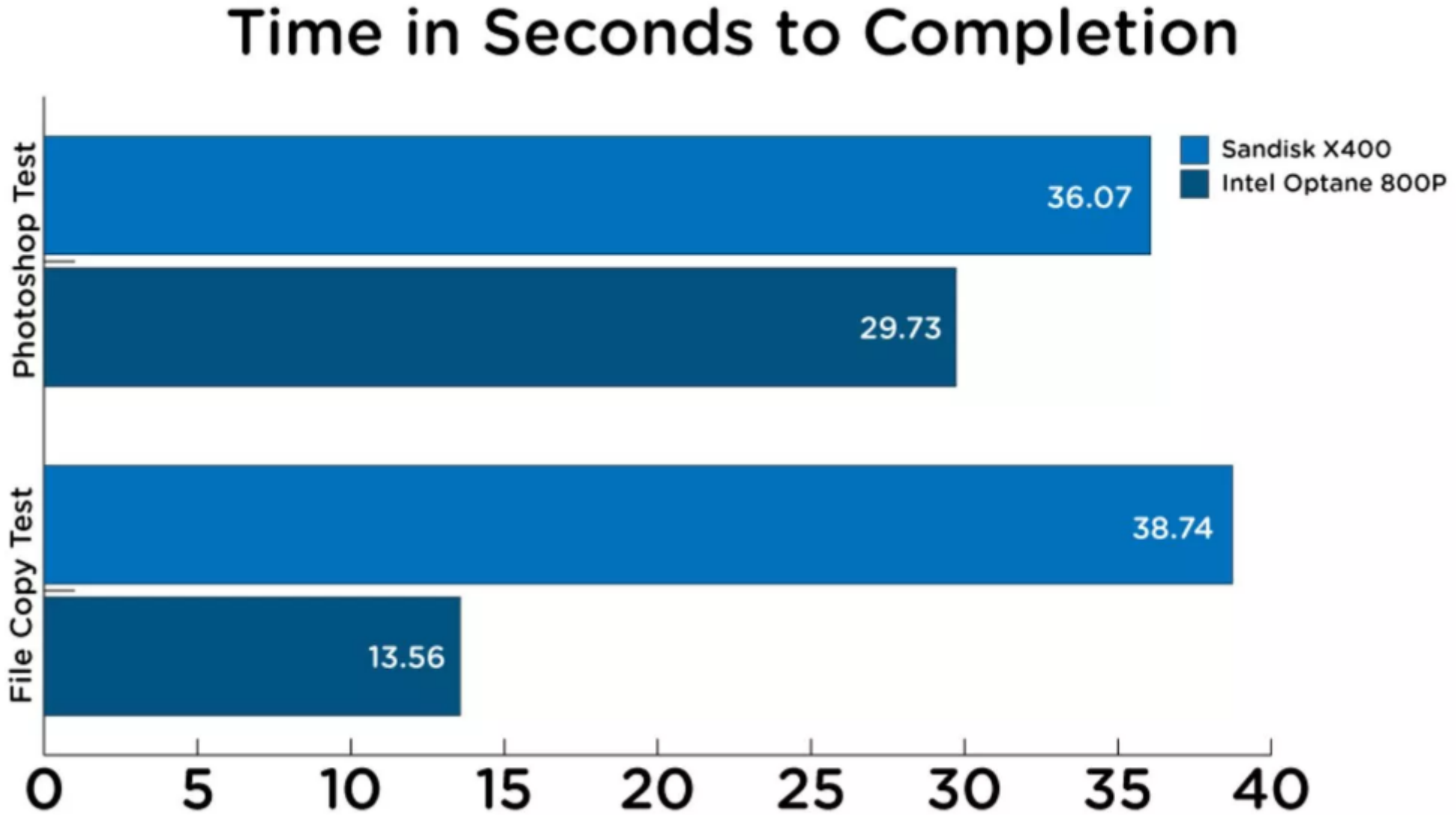

Image Source: Gizmodo

ORACLE STRATEGY

Complete Stack

- Best-of-breed
- Open
- Vertical Integration
- Extreme Performance
- Engineered Systems

Complete Customer Choice

- On-premise
- Private Cloud
- Public Cloud
- Hybrid Cloud

10x the Performance.
10x the Value.

ORACLE

引言

- 厂商发布的可能是特定环境下的最佳结果
- 厂商发布的业务负载与客户的未必一致
- 底层硬件的能力可能会被上层软件限制
- 硬件的提升并不会成比例带来软件的提升
  - IOPS: HDD:SATA-SSD:NVMe-SSD=200:70K:750K
- 硬件可能只适合特定场景，需要软件改造

引言-性能指标

- 吞吐量-越大越好，波动小
  - 读写带宽
  - IOPS
  - TPS
  - QPS
- 延迟-越小越好，波动小
  - 最小
  - 最大
  - 平均
  - 99%

Outline

- 引言
- 性能改进的原因
- 针对特定场景的优化
- 针对通用场景的优化
- 如何识别陷阱
- 性能调优
- 实际例子

性能改进的原因

- 硬件本身的提升
  - 更快的存储介质、更高的CPU主频等
- 并发能力的提高
  - 多存储通道、更多的CPU内核等
- 算法的优化改进
  - 新算法、并行化等
- 对比方式的投机取巧 ☹
  - 放大了性能改进的效果

Amdahl's law – 完成任务所需时间减少

- 考虑特定的计算任务并行化的加速

$$\text{加速比} = \frac{\text{一个处理器串行执行的时间}}{\text{多个处理器并行执行的时间}}$$

- f表示串行部分占的比例，p表示处理器个数

$$S_{(p)} = \frac{1}{f + \frac{1-f}{p}}$$

- 当p趋向于无穷大时，$S_{(p)}$趋向于1/f，这也是性能改善的上限。
- 增加处理器个数的边际效益递减。

Amdahl’s law的优缺点

- 优点
  - 可以反映特定计算任务的并行加速比上限
  - 可以用来估计某个瓶颈被优化后的上限
- 缺点
  - 仅考虑了处理器的能力（时代局限）
  - 没有考虑处理器多了之后可以做更多事情
  - 没有考虑Cache的影响
  - 没有考虑并行化后程序之间的通讯开销

Gustafson's law – 相同时间内做更多事情

- 考虑机器的处理能力

$$\text{加速比} = \frac{\text{多个处理器的计算量}}{\text{单个处理器的计算量}}$$

- W_s表示串行部分的负载，W_p表示并行部分的负载，W表示总负载，f表示串行负载所占比例

$$S_{(p)} = \frac{W_s + p \times W_p}{W_s + W_p} = \frac{f \times W + p \times (1-f) \times W}{W} = f + p \times (1-f)$$

- p越大，计算量增加越大

Outline

- 引言
- 性能改进的原因
- 针对特定场景的优化
- 针对通用场景的优化
- 如何识别陷阱
- 性能调优
- 实际例子

针对特定场景的优化

- 场景是否代表了实际情况？
  - MySQL经常用的RO场景，跑出百万甚至更高的QPS
  - MySQL混合读写的场景，但是关闭了BINLOG和REDO
- 优化的位置和副作用
  - 服务器端优化、客户端优化、文件系统优化？
  - 某些语句可以在客户端优化，直接返回结果。例如：SELECT 1
    如果应用依赖它来检测服务器是否活着，就会破坏语义

针对特定场景的优化

- 热点行更新
  - 针对秒杀场景非常有效
- 查询计划缓存
  - 只针对特定查询还是基本无限制？
- 查询结果集缓存
  - 同上
- 减少元信息
  - 非常适合读比例特别高并且结果集很小的场景

Outline

- 引言
- 性能改进的原因
- 针对特定场景的优化
- 针对通用场景的优化
- 如何识别陷阱
- 性能调优
- 实际例子

通用类型的优化

- 只读事务优化
  - 大多数场景下读多写少
- 针对高速硬件的优化
  - SSD等已经普及
- 支持PREPARE STATEMENT
  - 数据库的经典优化
- 线程池
  - 多核处理器
  - 短事务

通用类型的优化

- 支持存储过程
  - 减少数据传输
  - 减少语句交互
- 优化器的改进
  - 规则优化 vs 代价优化
  - SYSTEM R vs CASCADE
- 执行算法优化
  - HASH JOIN/AGGR
  - 并行算法

Outline

- 引言
- 性能改进的原因
- 针对特定场景的优化
- 针对通用场景的优化
- 如何识别陷阱
- 性能调优
- 实际例子

如何识别陷阱

- 关闭REDO日志，MySQL设置非双1
- 针对测试负载直接返回结果（作弊跑分）
- 针对测试负载做各种预处理（作弊跑分）
- 多台服务器的QPS累计到一起
- 拿新版本和竞品的旧版本对比
- 拿调优的版本和竞品的默认设置对比
- 拿原型系统与实际产品对比
- 拿自己的优势对比竞品的劣势

如何识别陷阱

- 采用标准化、综合性测试集
  - TPC系列
  - 崩溃恢复，测试ACID
- 随机测试集
- 归一化处理
- 性能估算，确定理论上下限
- 积累经验，知己知彼

Outline

- 引言
- 性能改进的原因
- 针对特定场景的优化
- 针对通用场景的优化
- 如何识别陷阱
- 性能调优
- 实际例子

调优

- 应用系统调优
  - 效益可能最大，但用户不配合
- 数据库本身的调优
  - 依赖产品特性以及专业经验
- 文件系统调优
- 存储设备调优
- 网卡、CPU调优

调优

- 让专业的人或工具干专业的事情
  - MySQL/Oracle 数据库调优指南
  - 借助AI或机器学习算法的工具
  - http://www.brendangregg.com/linuxperf.html