了解性能优化的发展历史及优化目标

性能优化

随着数据量、访问量的增加

- 硬件
- 操作系统
- 运行环境
- 应用本身

如何定位性能瓶颈

> 现象

根据资源消耗情况来分析

CPU 、 磁盘IO、 网络IO、内存 [监控系统]

CPU的消耗分析

用户空间线程

内核空间线程

硬件中断

CPU资源消耗来源：

1. 上下文切换
2. 运行队列

3. CPU利用率（死循环的线程）

定位工具：

> pidstat
>
> top
>
> vmstat

磁盘IO

- pidstat
- jstack

内存消耗

定位工具

1. vmstat
2. free -m
3. top
4. pidstat

应用程序运行较慢的问题

> 系统资源消耗不高，但是程序效率很低

锁竞争

- 线程锁
- 死锁
- 分布式锁

> 无锁化设计 或者 降低锁的粒度

硬件资源利用率不够

CPU（增加线程数量）、内存（内存缓存）

数据库/存储设备的性能较低

优化数据库的io效率。

第三方服务

如果第三方服务请求较慢

- 对接多个三方服务
- 异步化

执行调优策略

性能调优的目标， 在不影响程序正确性的情况下，去提高程序的运行效率。

1. 降低成本的优化。
2. 产品体验，商业价值

用户视角： 请求很慢,怎么这么慢

应用开发： 系统、软件的问题

宏观视角

架构重构

1. 硬件资源
    1. 垂直扩容
    2. 水平扩容
2. 软件架构
    1. 应用架构： 单体、分布式....
    2. 异步化架构
    3. 存储调优
    4. 集群架构

5. 增加缓存
    1. 客户端缓存（网页缓存）
    2. 代理缓存
    3. 数据缓存
    4. 内存缓存
    5. cpu缓存
    6. ....
6. 批处理/预处理
    1. seata -> 批量发送请求
    2. kafka -> 批量发送消息
    3. 预处理 [cpu的缓存行 、 mysql -Page Cache 预读取机制（Buffer Pool） ..]

局部视角

算法、编译、代码调优..

> 代码层面以及中间件的调优

- 中间件调优（标准化的能力）
- 多核cpu资源的利用（多线程）
- 锁优化（synchronized 、无锁化设计（CAS自旋）、LongAddr、ConcurrentHashMap（分段锁））
- 缓存的使用， 缓存预热
- 批处理（减少网络包的传输次数）
- 数据压缩（减少网络包传输的大小） -> 序列化算法 protobuf
- 使用合适的数据结构算法

  数组、链表、树、图、栈、Hash表、跳跃表

  不同的数据结构涉及到不同时间复杂度

  O(n)

  O(1)

  ConcurrrentHashMap 1.8 ， 链表升级成红黑树.
- 池化技术（对象池、内存池、连接池）
- JVM（运行环境）参数调优
- Tomcat 、 参数调优
- Mysql 、 参数调优

- 网络IO 、 NIO、BIO。。
- ....