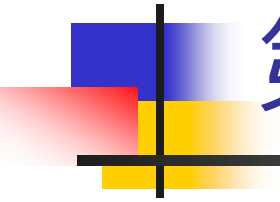
求實創新 勵志圖強

计算思维与计算科学

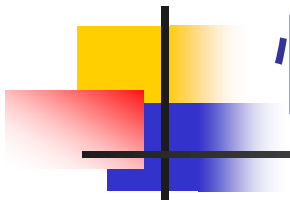
性能与优化

吉林大学 计算机科学与技术学院
College of Computer Science and Technology , Jilin University

吉林大学软件学院
国家示范性软件学院
College of Software , Jilin University



第十二章 性能与优化



性能与优化

一. 基本原则

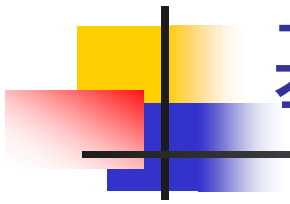
二. 基本路线和方法

  一. 代码级优化

  二. 算法级优化

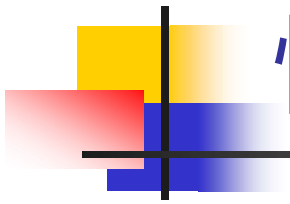
  三. 结构级优化

三. 系统架构优化



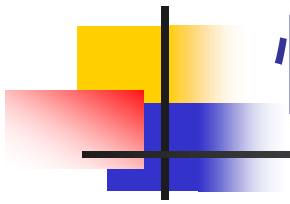
基本原则

- 过早的优化是罪恶之源——Donald Knuth
- 解决瓶颈问题
  - 性能评价与瓶颈发现
  - 数据量预测与评估
- 真正理解你要做的事情
- 优化不要破坏可读性和可维护性



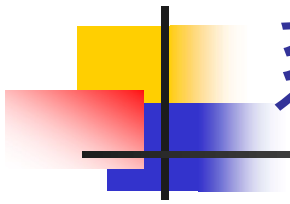
性能评估

- 外部性能指标
  - 响应时间：从提交请求到返回响应之间经过的时间
  - 吞吐量：对单位时间内完成工作的数量
  - 次要指标：并发量、资源耗用量、功率等
- 性能测试与分析
  - 负载测试与压力测试
  - 性能监控
  - 白盒测试与分析



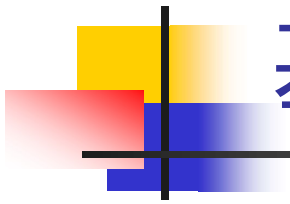
性能优化的阶段

- 开发时及上线前优化
- 上线运行后性能降低的优化
  - 性能与数据量之间存在线性（超线性）关系
  - 访问量增多带来的并发性能下降
- 性能告警，救火式优化
- 预防性优化



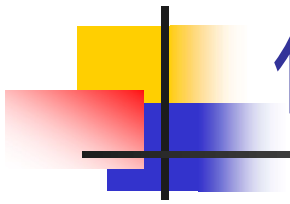
系统原理与优化

- 优化实质上可以分为三种情况
  - 真正减少工作量（算法与复杂度）
  - 使用更多的资源
  - 以更好的方式使用各种硬件资源和软件
- 在理解底层行为特性的技术上进行优化



基本路线

- 代码级优化
  - 对底层的深入理解
  - 和编译器合作
- 算法级优化
  - 算法复杂性
- 软件结构优化
- 系统架构优化

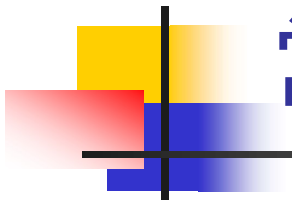


代码级优化

- 对相关原理有深入理解
  - 计算机原理（慢速内存、访问局部性、流水的限制）
  - 高级语言（复杂机制的实现，如多态函数）
  - 操作系统（操作系统调用是昂贵的、I/O机制、多任务）
  - 编译器（优化开关与选项）
  - 数据库管理系统(SQL优化)
  - 其他第三方库
  - 等等……

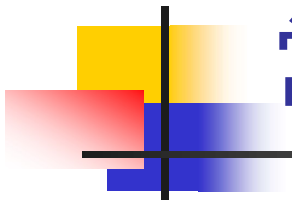
常见技巧-简单级

- 使用++，--
- 避免使用浮点数
- 多使用临时变量
- 使用移位代替乘除法
- 使用原生数组代替固定长度的数组类



常见技巧-语言相关（java）

- 创建Map等容器，使用初始化容量参数
- 多使用 final 修饰
- 尽早释放无用对象的引用

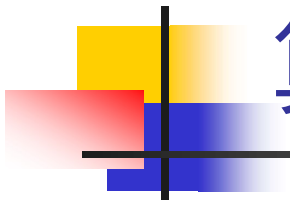


常见编译器优化技术

- 函数内联
- 常量预先计算
- 公共表达式提取
- 循环展开
- 循环内常量移除
- 表达式计算流水化
- 无用代码消除

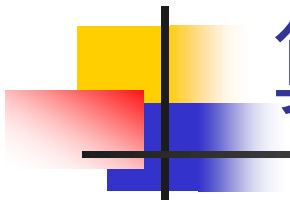
算法级优化基本路线

- 优化算法和数据结构
- 延迟计算
- 集中处理
  - 预处理、批处理
- 以空间换时间（存储中间结果）



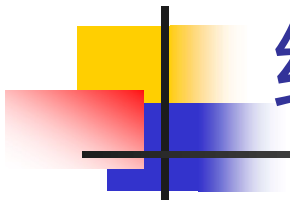
算法与数据结构优化

- 按照数据量和数据特征选择算法和数据结构
  - 例：图的最短路径
    - Floyd
    - Dijkstra
    - 基于堆的Dijkstra
    - 搜索与A*
  - 例：线段树（和查询）



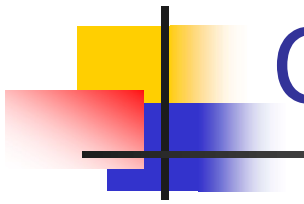
算法优化举例

- 延迟计算
  - 在首次使用前计算（singleton的初始化）
  - 字符串写时复制
- 预处理
  - 预先计算一些通用数据
- 批处理
  - 将请求累加到一定规模后批量处理
  - 如：数组删除元素，抢单排队处理，硬盘写文件



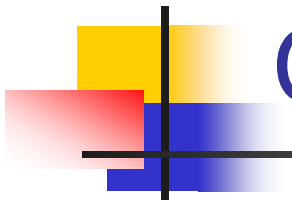
结构优化路线

- 缓存（Cache）
- 复用（Pool）
- 优化使用其他软件、模块的方法
- 分布式处理
  - 分离慢速部件（同步转异步）
  - 并行处理
  - 并行转串行



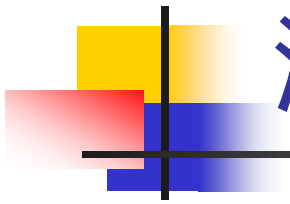
Cache

- 定义
  - 为了避免重复工作，匹配慢速部件和快速部件的运行效率，将结果保存在快速设备中
- 举例
  - 内存Cache（L2，虚拟内存）
  - 网络（DNS、HTTP客户端缓存）
  - 数据库Cache（Oracle）



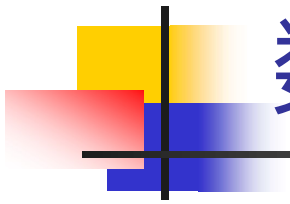
Cache

- 只读Cache
  - Cache加载策略
- 读写Cache
  - Cache写回策略
- Cache同步、更新与清除
- Cache造成的数据不一致
- 分布式Cache



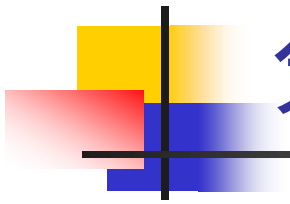
池化资源使用

- 预先申请一批资源，将其放入Pool中，使用者不直接申请资源，代之以对Pool的申请和释放
  - 避免了频繁创建、释放连接引起的大量性能开销，缩短响应时间
  - 避免资源泄露，避免并发量过大带来的系统颠簸
  - 引入一个抽象层，可以提供不同的具体实现，简化使用
  - 统一处理各类情况，提高稳定性和安全性



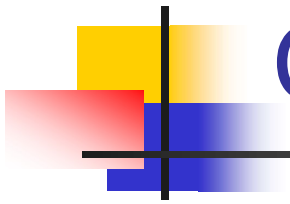
数据库优化

- 多次使用的SQL要预先编译
- 使用绑定参数，而不是拼接SQL
- 减少返回的数据量
- 数据库索引
  - 对常用字段建立索引（避免类型转换）
  - 字符串索引与前置模糊查询
  - 联合索引注意次序
  - 关注索引的利用率



分布式处理

- 方法
  - 多线程
  - 异构计算
  - 多机计算
- 关注点
  - 并发异常
  - 锁与死锁
  - 数据不一致



CAP理论

- 一个分布式系统最多只能同时满足一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三项中的两项
- 一致性：
  - 更新操作成功并返回客户端完成后，所有节点在同一时间的数据完全一致。
- 可用性：
  - 服务一直可用，而且是正常响应时间
- 分区容错性
  - 在遇到某节点或网络分区故障的时候，仍然能够对外提供满足一致性和可用性的服务

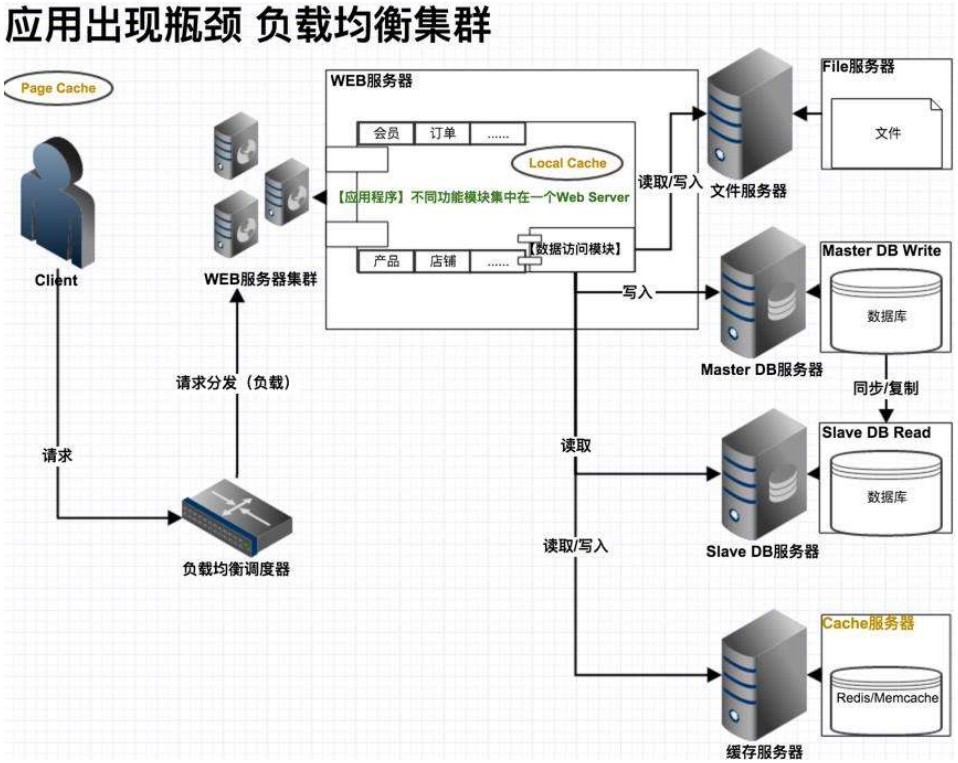
系统架构优化

- 垂直分割（业务模块拆分）
- 水平分割（业务分层、前后台分离）
- 水平扩展（可并行化）

实例系统-高业务量B/S系统

- 优化业务数据流（前置分流）
- 数据流中的缓存
- 高并发优化
- 数据库优化
- 架构优化

分布式信息处理系统


应用出现瓶颈 负载均衡集群
Page Cache
Client
WEB服务器
会员
订单
.....
Local Cache
【应用程序】不同功能模块集中在一个Web Server
【数据访问模块】
产品
店铺
.....
WEB服务器集群
请求分发（负载）
请求
负载均衡调度器
读取/写入
File服务器
文件
文件服务器
写入
Master DB Write
数据库
Master DB服务器
同步/复制
读取
Slave DB Read
数据库
Slave DB服务器
读取/写入
Cache服务器
Redis/Memcache
缓存服务器