性能与可伸缩性

- 对性能的思考
  - 性能与可伸缩性
    - 可伸缩性指的是:当增加计算资源时(如cpu、内存、存储容量或者IO带宽),程序的吞吐量或者处理能力相应的增加
    - 在性能调优时,其目的通常是用更小的代价完成相同的工作
    - 在进行可伸缩性调优时,其目的是设法将问题的计算并行化,从而利用更多的计算资源来完成更多的工作
  - 评估各种性能权衡因素
    - 避免不成熟的优化.首先使程序正确,然后再提高运行速度--如果它还运行的不够快
    - 在使某个方案比其他方案更快之前,首先问自己几个问题
      - 更快的含义是什么
      - 该方法在什么条件下运行的更快?在低负载还是高负载?大数据集还是小数据集?能否通过测试结果来验证你的答案?
      - 这些条件在运行环境中发生的频率?能否通过测试结果来验证你的答案?
      - 在其他不同条件的环境中能否使用这里的代码?
      - 在实现这种性能提升时需要付出哪些隐含的代价,例如增加开发风险或者维护开销?这种权衡是否合适?
    - 以测试为基准,不要猜测
- Amdahl定律
  - Speedup<=1/(F+(1-F)/N),F:必须被串行执行的部分,N:处理器个数
  - 在所有并发程序中都包含一些串行部分.如果你认为在你的程序中不存在那部分,那么可以再仔细检查一遍
  - 内存同步
    - 内存栅栏
    - 同步操作
      - 有竞争
      - 无竞争
- 减少锁的竞争

- 在并发程序中,对伸缩性的最主要威胁就是独占方式的资源锁
- 有3种方式可以降低锁的竞争程度
    - 减少锁的持有时间
    - 降低锁的请求频率
    - 使用带有协调机制的独占锁,这些机制允许更高的并发性
- 缩小锁的范围
- 减小锁的粒度
    - 锁分解
    - 锁分段
    - 避免热点域
- 替代独占锁的方法
    - ReadWriteLock
    - 原子变量
- 检测CPU利用率
- 向对象池说“不”
    - 通常,对象分配操作开销比同步的开销更低