第十一章 性能与可伸缩性

1. 线程的最主要目的是提高程序的运行性能。

（1）充分发挥可用处理器的能力。

（2）异步解耦，可以同时开启多个任务，提升响应性。

#### 一．对性能的思考

1. 资源密集型操作

操作性能受限于某种资源

2. 使用多线程的额外开销

（1）线程之间的协调（例如加锁、触发信号、内存同步）

（2）增加上下文的切换

（3）线程的创建和销毁

（4）线程的调度

注：如果过度的使用线程，那么这些开销甚至会超过由于提高吞吐量、响应性或者计算能力所带来的性能提升。

性能与可伸缩性

1. 可伸缩性

当增加计算资源时（如CPU、内存、存储容量、I/O带宽等），程序的吞吐量或者处理能力能相应的增加。

2. 大多数提高单线程程序的技术，往往都会破坏可伸缩性。

评估各种性能权衡因素

1. 小例子

快速排序算法在大规模数据集上的执行效率非常高。

但是对于小规模来说，冒泡排序效率更高。

2. 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。

3. 很多性能优化都是牺牲代码可读性或可维护性为代价。

4. 一些因素

（1）该方法何时运行的更快？

大数据集---小数据集

低负载---高负载

…

（2）性能提升时会不会付出隐含的代价，例如增加开发风险或维护开销。

5. 以测试为基准，不要猜测。

Amdahl定律

1. 如果要发挥多个处理器的能力，那么就必须对问题进行合理的分解，并使得程序能有效的使用这种潜在的并行能力。

http://hi.csdn.net/attachment/201004/22/0_1271944701C6d0.gif

如公式分析出，串行比例越低且处理器越多，加速比越高，程序优化效率越高。

如果串行比例占2/3，则无论处理器再多，最大加速比也只能达到1.5。

理想效果是，全部并行，最大加速比为 n。可以根据增加处理器无上限增强程序效率。

2. 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串行部分，那么可以再仔细检查一遍。

#### 二．线程引入的开销

1. 并行带来的性能提升必须超过并发导致的开销。

2. 上下文切换

（1）上下文切换的开销；

（2）首次运行时间长，上下文切换会导致一些缓存缺失；

3. 内存栅栏

内存栅栏可以刷新缓存，使缓存无效，刷新硬件的写缓冲，以及停止执行管道。

在内存栅栏中，大多数操作是不能重排序的。

4. 阻塞

（1）自旋锁

适用：等待时间短

（2）线程挂起

适用：等待时间长

（3）当线程因为阻塞被挂起，将包含两次额外的上下文切换，以及必要的操作系统操作和缓存操作。

（4）由于竞争而阻塞时，线程持有锁时将有一定的开销。

5. 锁消除优化

6. 把重点放在发生锁竞争的地方，而不是非竞争同步带来的开销。

#### 三．减少锁的竞争

1. 串行操作会降低可伸缩性，并且上下文切换也会降低性能，在锁上发生竞争将同时导致这两个问题，因此减少锁的竞争能够提高性能和可伸缩性。

2. 在并发程序中，对可伸缩性的最主要威胁就是独占方式的资源锁。

3. 导致在锁上发生竞争的可能性因素

（1）锁的请求频率

（2）每次持有该锁的时间

4. 降低锁的竞争程度

（1）减少锁的持有时间

（2）降低锁的请求频率（可以通过，锁分段、和锁分解来实现）

（3）使用带有协调机制的独占锁，这些机制允许更高的并发性

5. 减少锁的持有范围

（1）如果把一个同步代码块分解为多个同步代码块时，反而会对性能产生负面影响。

（2）如果JVM执行锁粒度粗化的操作，那么可能会将分解的同步块又重新合并起来。

6. 减少锁的粒度

（1）锁分解技术

将一个锁分解为多个锁。（竞争适中的情况下，可以提高性能和可伸缩性）

（2）锁分段技术

将锁分解利用到对象层次上，例如ConcurrentHashMap

缺点：获取独占的锁困难。例如，当ConcurrentHashMap需要扩容时，需要获得所有的锁。

7. 避免热点域

将一些反复计算的结果缓存起来。例如ConcurrentHashMap的size，往往热点域都会限制可伸缩性。

为了避免这个问题，ConcurrentHashMap为每个分段都维护了一个size，且都有该分段的锁来维护。

8. 一些替代独占锁的方法

（1）读写锁---ReadWriteLock

（2）并发容器---ThreadLocal

（3）原子变量---Atomic

9. 监测CPU利用率

CPU没有充分利用的几个原因

（1）负载不充足

（2）I/O密集

（3）锁竞争

10. 向对象池说不

通常，对象分配操作的开销比同步的开销更低。

#### 四．减少上下文切换的消耗

1. 锁获取操作上发生竞争时将导致更多的上下文切换。