漫谈多核应用服务器中线程池设置

2016-01-28 13:21 feiying 0 阅读 73

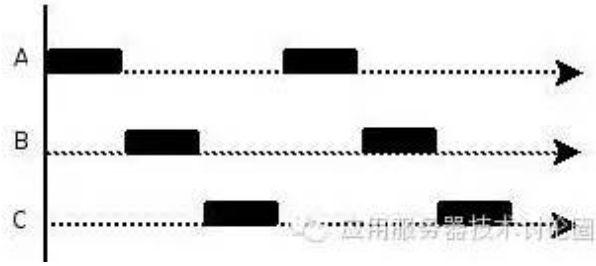
在讲解线程池设置之前，我们需要区分两个概念，一个是并发，一个是并行。

什么是并发？并发是在网站技术上经常使用的术语，例如说xxx网站可以支撑10亿并发，经常有在互联网公司面试的小伙会问你，

你做过多少并发的网站，说来听听，一般假如你说我也不知道，那基本上服务器端的职位直接印象就不是很好，感觉貌似你没有什么经验。

其实，这个并发数从技术角度来讲，是指一个物理的cpu，或者是多个，在各个线程/进程之间的进行切换任务，达到共享利用有限的物理资源来

解决多任务的问题，让更多的任务能进行执行，提高了工作的效率，如下图所示：



可以分析得出来，上述的服务器就是一个cpu，并且是1个核，因为A，B，C三个线程/进程执行任务，没有办法一直占据cpu执行，cpu会将自身

计算能力切分到多个任务中，因为切分的次数非常的频繁，几乎能达到微秒级别，导致根本感觉不出来，cpu是在切换，那么这种cpu多路执行

的思路就是并发。值得说的一点是，上述的cpu分配有点武断，其实运行一个操作系统，cpu还需要维护内核，核心服务等很多耗费cpu的地方，

并且在运行态到内核态的切换也会占用一部分时间，但大多数的时间还是切分到用户态的任务上。

对于并行来说，在一定的时间内，是指A，B，C三个任务同时进行，三个任务没有共享cpu的核的一说，如下图所示：

从这张图明显看出，并行实际上是在三个cpu核数上运行，每个核类似于一个处理中心，单独进行处理任务，互相不干扰，这种模式就是并行。

总结一下，其实并发大多数指的是一个CPU的多路任务复用，多个任务共享一个CPU的时间，这种模式和并行的模式不一样，并行是每一个任务

对应一个cpu核，自己干自己的，互相不干扰。

并行编程是近年来的一个趋势，因为cpu的工艺已经产生了瓶颈，导致现在cpu越做核数越多，因而现在基本上每个人的计算机上，虽然就一个cpu模组，

但是其中可能有n个核，这些核虽然有共享的资源，但是大多数都是独立运行的，因此在这样的前提下，基于并行的写代码方式是比较

关键的，而对应多任务来讲，一个任务可以对应为一个进程，也可以对应为一个线程。因此，原来的单核的时代，一般线程池的调整，通常没有

太多可以参考的依据，通常的做法就是在Jprofile，nmon，top等一些监控工具中，压测出一个波峰和拐点，这个拐点处一般的线程池的设置就是对应的数字。

而基于并行时代，线程池的设置还是有一些说法的，基于核的数量还有你的应用的类型。

1.计算密集型：

计算密集型，就是cpu发挥巨大作用的地方，IO瓶颈几乎没有，举例来讲，就是在内存之中倒数据，这种就是cpu和内存之间的交互，效率极高，这种就是

计算密集型。应用服务器就是典型的计算密集型，虽然网络IO有延迟，但是应用服务器前段请求解析，这仅仅是一个模块而已，对于后端的还有各种功能

，这些功能都是指令级，内存，寄存器之间的交互，cpu计算量比较大。

对于这种类型，**当有N个处理器的系统中，设置线程池的大小为N+1，是最优的利用率。**

为什么？可以理解因为大多数都是CPU计算，因此一个任务一个CPU，最优的配置自然靠近CPU的核数，但是，多出来的1是什么呢？这个1就是对应的页

确实或者页没有命中。在现如今的操作系统中物理内存和虚拟内存是不一样的，也就是说一个进程的数据和执行指令是放到虚拟内存中的，虚拟内存是物理

内存的一个映射，其地址空间划分为好多的页，因为物理内存小而硬盘空间大，为了在内存里放置更多的进程，操作系统的设计者们决定把页映射到内存帧

或硬盘上的虚拟内存文件中。进程的可视范围是它自己的地址空间，它并不知道某一页映射到内存里还是硬盘上，进程只管自己运行。当进程需要访问某一页时，

操作系统通过查看分页表，得知被请求的页在内存里还是硬盘里。若在内存里，则进行地址翻译；若在硬盘里，则发生页缺失。操作系统立即阻塞该进程，

将硬盘里对应的页换入内存，然后使该进程就绪，可以继续运行。而这种交换你也可以理解为一个错误，但通常意义上来讲，这是很通常发生的事情，

而经常线程就会因为这种方式而导致线程执行中断，这个时候如果没有另外一个线程利用这个cpu的核，那么就产生了浪费，这也就是为什么N+1的原因。

当然，除了这个页故障，还有其它原因，因此，只要做到和核数靠近的数字进行设置，基本就没有什么问题。

2.IO密集型

IO密集型的问题主要是，IO瓶颈特别大，例如数据库，需要和低速设备打交道，这种方式导致IO阻塞导致线程等待时间非常的长，因此再按照计算密集型的方式

设置线程池的大小就得不偿失了，这时候，你必须估算一下任务的等待时间和真正的计算时间的比值，

给定下列定义：

$$N_{cpu} = number\ of\ CPUs$$

$$U_{cpu} = target\ CPU\ utilization,\ 0 \leqslant U_{cpu} \leqslant 1$$

$$\frac{W}{C} = ratio\ of\ wait\ time\ to\ compute\ time$$

要使处理器达到期望的使用率，线程池的最优大小等于：

$$N_{threads} = N_{cpu} * U_{cpu} * \left(1 + \frac{W}{C}\right)$$

公式如上，N是cpu的核数，U是cpu的利用率，即可以通过top等工具来监视处cpu的实际利用率，对于等待时间和计算时间的比值就是W/C，

整体的公式是三者的乘积，整体思路也就是将等待时间的浪费转化为线程数的增加。

当然上述的比值，不一定很精确，还是需要通过top等工具来监视，计算出一个合理的范畴，找到拐点，线程池设置就合理了。

总结：并行时代的线程池设置根据应用类型和cpu核数来进行理论预估，辅助以测试工具压测，到达性能拐点即可设置正确。

东方通加码大数据业务 拟募资8亿收购微智信业 (/html/308/201504/206211355/1.html)

玩转Netty – 从Netty3升级到Netty4 (/html/308/201504/206233287/1.html)

金蝶中间件2015招聘来吧！Come on！ (/html/308/201505/206307460/1.html)

GlassFish 4.1 发布，J2EE 应用服务器 (/html/308/201505/206323120/1.html)

Tomcat对keep-alive的实现逻辑 (/html/308/201505/206357679/1.html)

猜您喜欢

谈谈Python协程 (/html/161/201607/2247483729/1.html)

阿里建“数据安全成熟度模型” 推动国际数据安全标准 (/html/176/201605/2650219072/1.html)

Java 重写方法与初始化的隐患 (/html/432/201510/400385059/1.html)

我把第一次献给了HBase (/html/46/201503/205149200/1.html)

快乐圣诞喜迎新年，广州传智播客微博活动礼品大放送 (/html/444/201312/10018032/1.html)