别再纠结线程池池大小、线程数量了，哪有什么固定公式 | 京东云技术团队

京东云开发者 2023-09-20 16,351 阅读10分钟 专栏：案例分享

可能很多人都看到过一个线程数设置的理论：

- CPU 密集型的程序 - 核心数 + 1
- I/O 密集型的程序 - 核心数 * 2

不会吧，不会吧，真的有人按照这个理论规划线程数?

线程数和CPU利用率的小测试

抛开一些操作系统，计算机原理不谈，说一个基本的理论（不用纠结是否严谨，只为好理解）：
一个CPU核心，单位时间内只能执行一个线程的指令

那么理论上，我一个线程只需要不停的执行指令，就可以跑满一个核心的利用率。

来写个死循环空跑的例子验证一下：

测试环境：AMD Ryzen 5 3600, 6 - Core, 12 - Threads

typescript 体验AI代码助手 代码解读 复制代码

```typescript
public class CPUUtilizationTest {
    public static void main(String[] args) {
            //死循环，什么都不做
            while (true){
            }
    }
}
```

运行这个例子后，来看看现在CPU的利用率：

```
top - 15:06:23 up  1:13,  0 users,  load average: 2.68, 5.03, 3.05
Tasks:  10 total,   1 running,   9 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.9 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13050720 total, 12861996 free,   102176 used,    86548 buff/cache
KiB Swap:  4194304 total,  4194304 free,        0 used. 12753652 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  725 jiangxin  20   0 6175400  30312  17472 S 100.0  0.2   1:34.58 java
    1 root      20   0     904    580    520 S   0.0  0.0   0:00.02 init
   10 root      20   0     896     88     20 S   0.0  0.0   0:00.00 init
   11 root      20   0     896     88     20 S   0.0  0.0   0:00.08 init
```

从图上可以看到，我的3号核心利用率已经被跑满了

那基于上面的理论，我多开几个线程试试呢?

typescript 体验AI代码助手 代码解读 复制代码

```typescript
public class CPUUtilizationTest {
    public static void main(String[] args) {

            for (int j = 0; j < 6; j++) {
                    new Thread(new Runnable() {
                            @Override
                            public void run() {
                                    while (true){
                                    }
                            }
                    }).start();
            }
    }
}
```

目录 收起

京东云开发者 LV.8
技术运营 @京东科技信息技术...
作者榜No.1 优秀作者
1.9k 文章 3.3m 阅读 21k 粉丝

此时再看CPU利用率，1/2/5/7/9/11 几个核心的利用率已经被跑满：

```
top - 15:56:39 up 36 min,  0 users,  load average: 4.37, 1.47, 0.53
Tasks:  10 total,   1 running,   9 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni, 95.2 id,  0.0 wa,  0.0 hi,  4.8 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13050720 total, 12847776 free,   108180 used,    94764 buff/cache
KiB Swap:  4194304 total,  4194304 free,        0 used. 12743540 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  212 jiangxin  20   0 6574784  33824  17448 S 600.3  0.3   3:19.39 java
```


那如果开12个线程呢，是不是会把所有核心的利用率都跑满？答案一定是会的：

```
top - 16:05:06 up 45 min,  0 users,  load average: 11.27, 7.19, 3.64
Tasks:  10 total,   1 running,   9 sleeping,   0 stopped,   0 zombie
%Cpu0  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13050720 total, 12850344 free,   105560 used,    94816 buff/cache
KiB Swap:  4194304 total,  4194304 free,        0 used. 12746136 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  300 jiangxin  20   0 6974168  30560  16924 S  1200  0.2  32:07.01 java
```


如果此时我把上面例子的线程数继续增加到24个线程，会出现什么结果呢？

```
top - 16:09:09 up 49 min,  0 users,  load average: 22.79, 14.28, 7.22
Tasks:  13 total,   1 running,  12 sleeping,   0 stopped,   0 zombie
%Cpu0  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13050720 total, 12810792 free,   115984 used,   123944 buff/cache
KiB Swap:  4194304 total,  4194304 free,        0 used. 12721120 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  358 jiangxin  20   0 7772936  34144  17416 S  1196  0.3  28:38.47 java
```


从上图可以看到，CPU利用率和上一步一样，还是所有核心100%，不过此时负载已经从11.x增加到了22.x（load average解释参考scoutapm.com/blog/unders...），说明此时CPU更繁忙，线程的任务无法及时执行。

现代CPU基本都是多核心的，比如我这里测试用的AMD 3600，6核心12线程（超线程），我们可以简单的认为它就是12核心CPU。那么我这个CPU就可以同时做12件事，互不打扰。

如果要执行的线程大于核心数，那么就需要通过操作系统的调度了。操作系统给每个线程分配CPU时间片资源，然后不停的切换，从而实现“并行”执行的效果。

但是这样真的更快吗？从上面的例子可以看出，**一个线程**就可以把**一个核心**的利用率跑满。如果每个线程都很“霸道”，不停的执行指令，不给CPU空闲的时间，并且同时执行的线程数大于CPU的核心数，就会导致操作系统**更频繁的执行切换线程执行**，以确保每个线程都可以得到执行。

不过切换是有代价的，**每次切换会伴随着寄存器数据更新，内存页表更新等操作**。虽然一次切换的代价和I/O操作比起来微不足道，但如果线程过多，线程切换的过于频繁，甚至在单位时间内切换的耗时已经大于程序执行的时间，就会导致CPU资源过多的浪费在上下文切换上，而不是在执行程序，得不偿失。

上面死循环空跑的例子，有点过于极端了，正常情况下不太可能有这种程序。

大多程序在运行时都会有一些 I/O操作，可能是读写文件，网络收发报文等，这些 I/O 操作在进行时时需要等待反馈的。比如网络读写时，需要等待报文发送或者接收到，在这个等待过程中，线程是等待状态，CPU没有工作。此时操作系统就会调度CPU去执行其他线程的指令，这样就完美利用了CPU这段空闲期，提高了CPU的利用率。

上面的例子中，程序不停的循环什么都不做，CPU要不停的执行指令，几乎没有啥空闲的时间。如果插入一段I/O操作呢，I/O 操作期间 CPU是空闲状态，CPU的利用率会怎么样呢？先看看单线程下的结果：

java 体验AI代码助手 代码解读 复制代码

```java
public class CPUUtilizationTest {
    public static void main(String[] args) throws InterruptedException {

            for (int n = 0; n < 1; n++) {
                    new Thread(new Runnable() {
                            @Override
                            public void run() {
                                    while (true){
                        //每次空循环 1亿 次后，sleep 50ms，模拟 I/O等待、切换
                                            for (int i = 0; i < 100_000_000l; i++) {
                                            }
                                            try {
                                                    Thread.sleep(50);
                                            }
                                            catch (InterruptedException e) {
                                                    e.printStackTrace();
                                            }
                                    }
                            }
                    }).start();
            }
    }
}
```

```
top - 19:51:16 up  4:31,  0 users,  load average: 0.41, 3.11, 3.41
Tasks:  14 total,   1 running,  13 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  : 50.0 us,  0.0 sy,  0.0 ni, 50.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13050720 total, 12820984 free,   105432 used,   124304 buff/cache
KiB Swap:  4194304 total,  4194304 free,        0 used. 12731520 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2455 jiangxin  20   0 6241964  31552  17408 S  50.0  0.2   3:04.66 java
```


哇，唯一有利用率的9号核心，利用率也才50%，和前面没有sleep的100%相比，已经低了一半了。现在把线程数调整到12个看看：

```
top - 19:59:41 up  4:39,  0 users,  load average: 7.75, 6.52, 4.96
Tasks:  14 total,   1 running,  13 sleeping,   0 stopped,   0 zombie
%Cpu0  : 61.2 us,  0.0 sy,  0.0 ni, 35.6 id,  0.0 wa,  0.0 hi,  3.2 si,  0.0 st
%Cpu1  : 60.1 us,  0.0 sy,  0.0 ni, 39.6 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu2  : 60.5 us,  0.0 sy,  0.0 ni, 39.2 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu3  : 62.7 us,  0.0 sy,  0.0 ni, 37.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 61.3 us,  0.0 sy,  0.0 ni, 38.1 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu5  : 62.2 us,  0.0 sy,  0.0 ni, 37.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 61.1 us,  0.0 sy,  0.0 ni, 38.5 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu7  : 59.7 us,  0.0 sy,  0.0 ni, 40.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  : 61.7 us,  0.0 sy,  0.0 ni, 38.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  : 57.0 us,  0.0 sy,  0.0 ni, 43.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 : 64.0 us,  0.0 sy,  0.0 ni, 36.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 : 60.1 us,  0.0 sy,  0.0 ni, 39.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13050720 total, 12821816 free,   104492 used,   124412 buff/cache
KiB Swap:  4194304 total,  4194304 free,        0 used. 12732400 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2560 jiangxin  20   0 6974168  30288  17088 S 734.9  0.2  56:11.73 java
```


单个核心的利用率60左右，和刚才的单线程结果差距不大，还没有把CPU利用率跑满，现在将线程数增加到18：

```
top - 20:02:20 up  4:42,  0 users,  load average: 11.21, 7.98, 5.72
Tasks:  14 total,   1 running,  13 sleeping,   0 stopped,   0 zombie
%Cpu0  : 95.0 us,  0.0 sy,  0.0 ni,  5.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 93.3 us,  0.0 sy,  0.0 ni,  6.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 97.0 us,  0.0 sy,  0.0 ni,  3.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 91.3 us,  0.0 sy,  0.0 ni,  8.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 93.7 us,  0.0 sy,  0.0 ni,  5.9 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu5  : 92.7 us,  0.0 sy,  0.0 ni,  7.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 91.1 us,  0.0 sy,  0.0 ni,  8.3 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu7  : 97.3 us,  0.0 sy,  0.0 ni,  2.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  : 92.0 us,  0.0 sy,  0.0 ni,  8.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  : 95.7 us,  0.0 sy,  0.0 ni,  4.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 : 95.0 us,  0.0 sy,  0.0 ni,  5.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 : 95.3 us,  0.0 sy,  0.0 ni,  4.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13050720 total, 12820236 free,   106024 used,   124460 buff/cache
KiB Swap:  4194304 total,  4194304 free,        0 used. 12730844 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2618 jiangxin  20   0 7373552  31936  17332 S  1130  0.2  16:44.35 java
```


此时单核心利用率，已经接近100%了。由此可见，当线程中有 I/O 等操作不占用CPU资源时，操作系统可以调度CPU可以同时执行更多的线程。

现在将I/O事件的频率调高看看呢，把循环次数减到一半，50_000_000，同样是18个线程：

```
top - 20:06:23 up  4:46,  0 users,  load average: 9.37, 9.74, 7.04
Tasks:  14 total,   1 running,  13 sleeping,   0 stopped,   0 zombie
%Cpu0  : 71.3 us,  0.0 sy,  0.0 ni, 28.3 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  : 65.1 us,  0.0 sy,  0.0 ni, 33.6 id,  0.0 wa,  0.0 hi,  1.3 si,  0.0 st
%Cpu2  : 75.6 us,  0.0 sy,  0.0 ni, 24.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 63.7 us,  0.0 sy,  0.0 ni, 35.6 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu4  : 72.8 us,  0.0 sy,  0.0 ni, 27.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 65.2 us,  0.0 sy,  0.0 ni, 34.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 73.7 us,  0.0 sy,  0.0 ni, 26.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 72.3 us,  0.3 sy,  0.0 ni, 27.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  : 73.5 us,  0.0 sy,  0.0 ni, 26.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  : 69.2 us,  0.0 sy,  0.0 ni, 30.5 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu10 : 70.3 us,  0.0 sy,  0.0 ni, 29.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 : 69.7 us,  0.3 sy,  0.0 ni, 30.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13050720 total, 12818532 free,   107624 used,   124564 buff/cache
KiB Swap:  4194304 total,  4194304 free,        0 used. 12729192 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2682 jiangxin  20   0 7373552  31844  17268 S 841.9  0.2   4:51.18 java
```


此时每个核心的利用率，大概只有70%左右了。

线程数和CPU利用率的小总结

上面的例子，只是辅助，为了更好的理解线程数/程序行为/CPU状态的关系，来简单总结一下：

1. 一个极端的线程（不停执行“计算”型操作时），就可以把单个核心的利用率跑满，多核心CPU最多只能同时执行等于核心数的“极端”线程数
2. 如果每个线程都这么“极端”，且同时执行的线程数超过核心数，会导致不必要的切换，造成负载过高，只会让执行更慢
3. I/O 等暂停类操作时，CPU处于空闲状态，操作系统调度CPU执行其他线程，可以提高CPU利用率，同时执行更多的线程
4. I/O 事件的频率频率越高，或者等待/暂停时间越长，CPU的空闲时间也就更长，利用率越低，操作系统可以调度CPU执行更多的线程

线程数规划的公式

前面的铺垫，都是为了帮助理解，现在来看看书本上的定义。《Java 并发编程实战》介绍了一个线程数计算的公式：

如果希望程序跑到CPU的目标利用率，需要的线程数公式为：

$$Nthreads = Ncpu * Ucpu * (1 + \frac{W}{C})$$

公式很清晰，现在来带入上面的例子试试看：

如果我期望目标利用率为90%（多核90），那么需要的线程数为：

核心数12 * 利用率0.9 * (1 + 50(sleep时间)/50(循环50_000_000耗时)) ≈ 22

现在把线程数调到22，看看结果：

现在CPU利用率大概80+，和预期比较接近了，由于线程数过多，还有些上下文切换的开销，再加上测试用例不够严谨，所以实际利用率低一些也正常。

把公式变个形，还可以通过线程数来计算CPU利用率：

线程数22 / (核心数12 * (1 + 50(sleep时间)/50(循环50_000_000耗时))) ≈ 0.9

虽然公式很好，但在真实的程序中，**一般很难获得准确的等待时间和计算时间，因为程序很复杂，不只是“计算”**。一段代码中会有很多的内存读写，计算，I/O 等复合操作，精确的获取这两个指标很难，所以光靠公式计算线程数过于理想化。

真实程序中的线程数

那么在实际的程序中，或者说一些Java的业务系统中，线程数（线程池大小）规划多少合适呢?

先说结论：没有固定答案，先设定预期，比如我期望的CPU利用率在多少，负载在多少，GC频率多少之类的指标后，再通过测试不断的调整到一个合理的线程数

比如一个普通的，SpringBoot 为基础的业务系统，默认Tomcat容器+HikariCP连接池+G1回收器，如果此时项目中也需要一个业务场景的多线程（或者线程池）来异步/并行执行业务流程。

此时我按照上面的公式来规划线程数的话，误差一定会很大。因为此时这台主机上，已经有很多运行中的线程了，Tomcat有自己的线程池，HikariCP也有自己的后台线程，JVM也有一些编译的线程，连G1都有自己的后台线程。这些线程也是运行在当前进程、当前主机上的，也会占用CPU的资源。

所以受环境干扰下，单靠公式很难准确的规划线程数，一定要通过测试来验证。

流程一般是这样：

1. 分析当前主机上，有没有其他进程干扰
2. 分析当前JVM进程上，有没有其他运行中或可能运行的线程
3. 设定目标
    1. 目标CPU利用率 - 我最高能容忍我的CPU飙到多少?
    2. 目标GC频率/暂停时间 - 多线程执行后，GC频率会增高，最大能容忍到什么频率，每次暂停时间多少?
    3. 执行效率 - 比如批处理时，我单位时间内要开多少线程才能及时处理完毕
    4. ……
4. 梳理链路关键点，是否有卡脖子的点，因为如果线程数过多，链路上某些节点资源有限可能会导致大量的线程在等待资源（比如三方接口限流，连接池数量有限，中间件压力过大无法支撑等）
5. 不断的增加/减少线程数来测试，按最高的要求去测试，最终获得一个“满足要求”的线程数**

而且而且而且！不同场景下的线程数理念也有所不同：

1. Tomcat中的maxThreads，在Blocking I/O和No-Blocking I/O下就不一样
2. Dubbo 默认还是单连接呢，也有I/O线程（池）和业务线程（池）的区分，I/O线程一般不是瓶颈，所以不必太多，但业务线程很容易称为瓶颈
3. Redis 6.0以后也是多线程了，不过它只是I/O 多线程，“业务”处理还是单线程

所以，不要纠结设置多少线程了。没有标准答案，一定要结合场景，带着目标，通过测试去找到一个最合适的线程数。

可能还有同学可能会有疑问：“我们系统也没啥压力，不需要那么合适的线程数，只是一个简单的异步场景，不影响系统其他功能就可以”

很正常，很多的内部业务系统，并不需要啥性能，稳定好用符合需求就可以了。那么我的推荐的线程数是：**CPU核心数**

附录

Java 获取CPU核心数

```scss
Runtime.getRuntime().availableProcessors()//获取逻辑核心数，如6核心12线程，那么返回的是12
```

Linux 获取CPU核心数

```bash
# 总核数 = 物理CPU个数 X 每颗物理CPU的核数
# 总逻辑CPU数 = 物理CPU个数 X 每颗物理CPU的核数 X 超线程数

# 查看物理CPU个数
cat /proc/cpuinfo| grep "physical id"| sort| uniq| wc -l

# 查看每个物理CPU中core的个数(即核数)
cat /proc/cpuinfo| grep "cpu cores"| uniq

# 查看逻辑CPU的个数
cat /proc/cpuinfo| grep "processor"| wc -l
```

> 如果我的文章对您有帮助，请点赞/收藏/关注鼓励支持一下吧❤❤❤❤❤❤

> 作者：京东保险 蒋信
>
> 来源：京东云开发者社区 转载请注明来源

标签： cpu 后端 Java 话题： 金石计划征文活动

本文收录于以下专栏

案例分享 专栏目录
京东云技术案例分享
233 订阅 · 142 篇文章

订阅

上一篇 主动写入流对@ResponseBody注解...
下一篇 慢SQL原因分析之索引失效 | 京东物...

评论 42

登录 / 注册 即可发布评论!

最热 | 最新

STAYFOCUS @百度
11月前 点赞 评论

开心码代码

最热 | 最新

STAYFOCUS @百度
11月前 点赞 评论

开心码代码
有知道dubbo为什么调用量一大 服务CPU就飙升吗 看都是Dubbo线程占用了大量的CPU
1年前 点赞 1

Yanlun0323 : 接口响应不过来，大量连接堵塞占用资源?
1年前 1 回复

skyishero
线程redis线程池最大连接数设置的是4w，要不根本就顶不住
1年前 点赞 评论

查看全部 42 条评论

为你推荐

别再纠结线程池大小/线程数量了，没有固定公式的
空无 | 4年前 | 10k 208 24 Java

线程池不再乱配线程数了
ThinkSmart | 1年前 | 4.2k 62 13 后端 面试 Java

如何合理地估算线程池大小
石臻臻的杂货铺 | 2年前 | 1.1k 12 评论 掘金·金石计划

线程优化需要了解的一些点
CodeOver | 4年前 | 659 点赞 评论 Android

CPU 和 线程
日月星辰Ace | 2月前 | 77 1 评论 cpu

灵魂发问！线程池到底创建多少线程比较合理?
公众号_yi博说 | 4年前 | 1.4k 6 2 Java

线程池的数量和线程池中线程数量如何设置-理论篇
蚂蚁背大象 | 3年前 | 4.1k 30 3 后端 Java

合理配置线程池的线程数量

CoderJie | 3年前 | 7.0k | 25 | 评论 | 后端 | Java

面试官：如何写出让 CPU 跑得更快的代码?

小林coding | 4年前 | 1.0k | 4 | 评论 | 操作系统

如何定位线上Java应用CPU飙高

长安不见 | 1年前 | 359 | 1 | 评论 | 后端

线程池参数该怎么配置才能充分压榨CPU?

blaze | 1年前 | 417 | 2 | 评论 | 后端

并发问题的三大根源

biubiuQ | 1年前 | 300 | 1 | 评论 | Java

线程相关的知识归纳整理

积跬步以致千里_ylc | 5年前 | 221 | 2 | 评论 | Java

java线程-Java内存模型

Shawn_Shawn | 2年前 | 1.6k | 11 | 1 | Java

18张图让你搞懂高并发中的线程与线程池，看完还不会你来打我!

谷鸡泰 | 4年前 | 262 | 4 | 评论 | Java