

二

03 通过你的CPU主频，我们来谈谈“性能”究竟是什么？

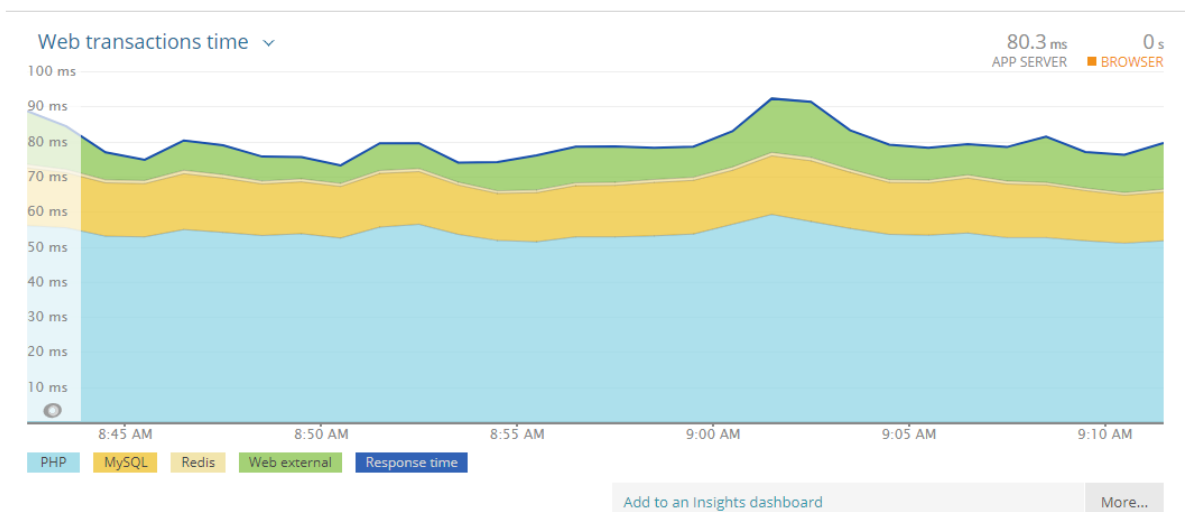
“性能”这个词，不管是在日常生活还是写程序的时候，都经常被提到。比方说，买新电脑的时候，我们会说“原来的电脑性能跟不上了”；写程序的时候，我们会说，“这个程序性能需要优化一下”。那么，你有没有想过，我们常常挂在嘴边的“性能”到底指的是什么呢？我们能不能给性能下一个明确的定义，然后来进行准确的比较呢？

在计算机组成原理乃至体系结构中，“性能”都是最重要的一个主题。我在前面说过，学习和研究计算机组成原理，就是在理解计算机是怎么运作的，以及为什么要这么运作。“为什么”所要解决的事情，很多时候就是提升“性能”。

什么是性能？时间的倒数

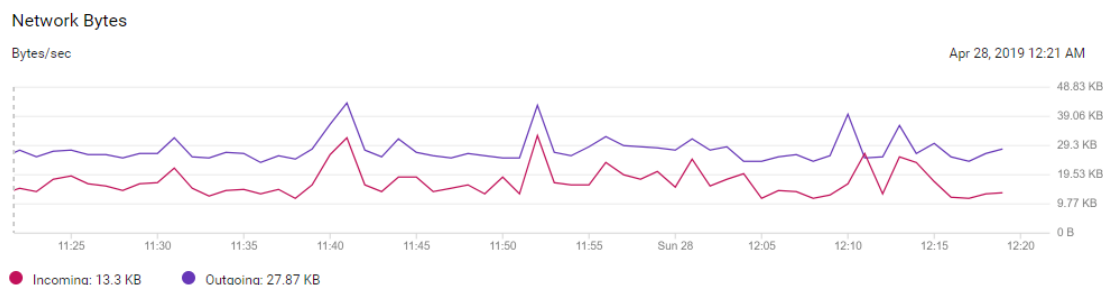
计算机的性能，其实和我们干体力劳动很像，好比是我们要搬东西。对于计算机的性能，我们需要有个标准来衡量。这个标准中主要有两个指标。

第一个是**响应时间**（Response time）或者叫执行时间（Execution time）。想要提升响应时间这个性能指标，你可以理解为让计算机“跑得更快”。



图中是我们实际系统里性能监测工具 NewRelic 中的响应时间，代表了每个外部的 Web 请求的执行时间

第二个是**吞吐量**（Throughput）或者带宽（Bandwidth），想要提升这个指标，你可以理解为让计算机“搬得更多”。



服务器使用的网络带宽，通常就是一个吞吐量性能指标

所以说，响应时间指的就是，我们执行一个程序，到底需要花多少时间。花的时间越少，自然性能就越好。

而吞吐量是指我们在一定的时间范围内，到底能处理多少事情。这里的“事情”，在计算机里就是处理的数据或者执行的程序指令。

和搬东西来做对比，如果我们的响应时间短，跑得快，我们可以来回多跑几趟多搬几趟。所以说，缩短程序的响应时间，一般来说都会提升吞吐量。

除了缩短响应时间，我们还有别的方法吗？当然有，比如说，我们还可以多找几个人一起来搬，这就类似现代的服务器都是 8 核、16 核的。人多力量大，同时处理数据，在单位时间内就可以处理更多数据，吞吐量自然也就上去了。

提升吞吐率的办法有很多。大部分时候，我们只要多加一些机器，多堆一些硬件就好了。但是响应时间的提升却没有那么容易，因为 CPU 的性能提升其实在 10 年前就处于“挤牙膏”的状态了，所以我们得慎重地来分析对待。下面我们具体来看。

我们一般把性能，定义成响应时间的倒数，也就是：

性能 = $1 / \text{响应时间}$

这样一来，响应时间越短，性能的数值就越大。同样一个程序，在 Intel 最新的 CPU Coffee Lake 上，只需要 30s 就能运行完成，而在 5 年前 CPU Sandy Bridge 上，需要 1min 才能完成。那么我们自然可以算出来，Coffee Lake 的性能是 $1/30$ ，Sandy Bridge 的性能是 $1/60$ ，两个的性能比为 2。于是，我们就可以说，Coffee Lake 的性能是 Sandy

Bridge 的 2 倍。

过去几年流行的手机跑分软件，就是把多个预设好的程序在手机上运行，然后根据运行需要的时间，算出一个分数来给出手机的性能评估。而在业界，各大 CPU 和服务厂商组织了一个叫作**SPEC**（Standard Performance Evaluation Corporation）的第三方机构，专门用来指定各种“跑分”的规则。

SPEC® CPU2017 Floating Point Rate Result		
Copyright 2017-2018 Standard Performance Evaluation Corporation		
ASUSTeK Computer Inc. ASUS RS700-E9(Z11PP-D24) Server System (2.70 GHz, Intel Xeon Gold 6150)		SPECrate2017_fp_base = 199 SPECrate2017_fp_peak = 201
CPU2017 License:	9016	Test Date: Dec-2017
Test Sponsor:	ASUSTeK Computer Inc.	Hardware Availability: Jul-2017
Tested by:	ASUSTeK Computer Inc.	Software Availability: Sep-2017

Benchmark result graphs are available in the [PDF report](#).

Hardware				Software										
CPU Name:	Intel Xeon Gold 6150			OS:	SUSE Linux Enterprise Server 12 (x86_64) SP2 Kernel 4.4.21-69-default									
Max MHz:	3700			Compiler:	C/C++: Version 18.0.0.128 of Intel C/C++ Compiler for Linux; Fortran: Version 18.0.0.128 of Intel Fortran Compiler for Linux									
Nominal:	2700			Parallel:	No									
Enabled:	36 cores, 2 chips, 2 threads/core			Firmware:	Version 0601 released Oct-2017									
Orderable:	1, 2 chip(s)			File System:	btrfs									
Cache L1:	32 KB I + 32 KB D on chip per core			System State:	Run level 3 (multi-user)									
L2:	1 MB I+D on chip per core			Base Pointers:	64-bit									
L3:	24.75 MB I+D on chip per chip			Peak Pointers:	64-bit									
Other:	None			Other:	None									
Memory:	768 GB (24 x 32 GB 2Rx4 PC4-2666V-R)													
Storage:	1 x 960 GB SATA SSD													
Other:	None													
Results Table														
Benchmark	Base						Peak							
	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
503.bwaves_r	72	1529	472	1529	472	1530	472	72	1526	473	1526	473	1526	473
507.cactuBSSN_r	72	504	181	505	181	505	180	72	505	180	505	180	505	180
508.namd_r	72	383	179	381	180	383	179	72	382	179	382	179	383	179
510.parest_r	72	1669	113	1684	112	1700	111	72	1687	112	1699	111	1695	111
511.povray_r	72	603	279	618	272	596	282	72	521	323	524	321	525	320
519.lbm_r	72	678	112	677	112	677	112	72	677	112	678	112	677	112
521.wrf_r	72	772	209	784	206	787	205	72	788	205	785	205	785	205
526.blender_r	72	495	222	495	222	495	222	72	487	225	487	225	488	225
527.cam4_r	72	547	230	550	229	550	229	72	543	232	542	232	542	232
538.imagick_r	72	509	352	509	352	510	351	72	511	351	511	351	511	351
544.nab_r	72	399	303	398	304	400	303	72	396	306	396	306	395	307
549.fotonik3d_r	72	1985	141	1986	141	1986	141	72	1985	141	1981	142	1985	141
554.roms_r	72	1263	90.6	1268	90.2	1269	90.2	72	1271	90.0	1270	90.1	1271	90.1

一份 SPEC 报告通常包含了大量不同测试的评分

SPEC 提供的 CPU 基准测试程序，就好像 CPU 届的“高考”，通过数十个不同的计算程序，对于 CPU 的性能给出一个最终评分。这些程序丰富多彩，有编译器、解释器、视频压缩、人工智能国际象棋等等，涵盖了方方面面的应用场景。感兴趣的话，你可以点击[这个链接](#)看看。

计算机的计时单位：CPU 时钟

虽然时间是一个很自然的用来衡量性能的指标，但是用时间来衡量时，有两个问题。

第一个就是时间不“准”。如果用你自己随便写的一个程序，来统计程序运行的时间，每一次统计结果不会完全一样。有可能这一次花了 45ms，下一次变成了 53ms。

为什么会不准呢？这里面有好几个原因。首先，我们统计时间是用类似于“掐秒表”一样，记录程序运行结束的时间减去程序开始运行的时间。这个时间也叫 Wall Clock Time 或者 Elapsed Time，就是在运行程序期间，挂在墙上的钟走掉的时间。

但是，计算机可能同时运行着好多个程序，CPU 实际上不停地在各个程序之间进行切换。在这些走掉的时间里面，很可能 CPU 切换去运行别的程序了。而且，有些程序在运行的时候，可能要从网络、硬盘去读取数据，要等网络和硬盘把数据读出来，给到内存和 CPU。所以说，**要想准确统计某个程序运行时间，进而去比较两个程序的实际性能，我们得把这些时间给刨除掉。**

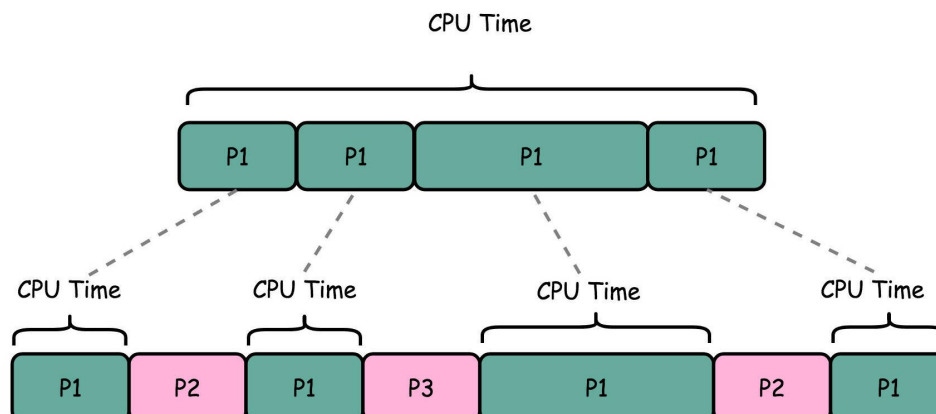
那这件事怎么实现呢？Linux 下有一个叫 time 的命令，可以帮我们统计出来，同样的 Wall Clock Time 下，程序实际在 CPU 上到底花了多少时间。


我们简单运行一下 time 命令。它会返回三个值，第一个是**real time**，也就是我们说的 Wall Clock Time，也就是运行程序整个过程中流逝掉的时间；第二个是**user time**，也就是 CPU 在运行你的程序，在用户态运行指令的时间；第三个是**sys time**，是 CPU 在运行你的程序，在操作系统内核里运行指令的时间。而**程序实际花费的 CPU 执行时间（CPU Time），就是 user time 加上 sys time。**

```
$ time seq 1000000 | wc -l
1000000
```

```
real  0m0.101s
user  0m0.031s
sys   0m0.016s
```

在我给的这个例子里，你可以看到，实际上程序用了 0.101s，但是 CPU time 只有 $0.031 + 0.016 = 0.047\text{s}$ 。运行程序的时间里，只有不到一半是实际花在这个程序上的。





Wall Clock Time / Elapsed Time

程序实际占用的 CPU 时间一般比 Elapsed Time 要少不少

其次，即使我们已经拿到了 CPU 时间，我们也不一定可以直接“比较”出两个程序的性能差异。即使在同一台计算机上，CPU 可能满载运行也可能降频运行，降频运行的时候自然花的时间会多一些。

除了 CPU 之外，时间这个性能指标还会受到主板、内存这些其他相关硬件的影响。所以，我们需要对“时间”这个我们可以感知的指标进行拆解，把程序的 CPU 执行时间变成 CPU 时钟周期数（CPU Cycles）和 时钟周期时间（Clock Cycle）的乘积。

程序的 CPU 执行时间 = CPU 时钟周期数 × 时钟周期时间

我们先来理解一下什么是时钟周期时间。你在买电脑的时候，一定关注过 CPU 的主频。比如我手头的这台电脑就是 Intel Core-i7-7700HQ 2.8GHz，这里的 2.8GHz 就是电脑的主频（Frequency/Clock Rate）。这个 2.8GHz，我们可以先粗浅地认为，CPU 在 1 秒时间内，可以执行的简单指令的数量是 2.8G 条。

如果想要更准确一点描述，这个 2.8GHz 就代表，我们 CPU 的一个“钟表”能够识别出来的最小的时间间隔。就像我们挂在墙上的挂钟，都是“滴答滴答”一秒一秒地走，所以通过墙上的挂钟能够识别出来的最小时间单位就是秒。

而在 CPU 内部，和我们平时戴的电子石英表类似，有一个叫晶体振荡器（Oscillator Crystal）的东西，简称为晶振。我们把晶振当成 CPU 内部的电子表来使用。晶振带来的每一次“滴答”，就是时钟周期时间。

在我这个 2.8GHz 的 CPU 上，这个时钟周期时间，就是 $1/2.8G$ 。我们的 CPU，是按照这个“时钟”提示的时间来进行自己的操作。主频越高，意味着这个表走得越快，我们的 CPU 也就“被逼”着走得越快。

如果你自己组装过台式机的话，可能听说过“超频”这个概念，这说的其实就相当于把买回来的 CPU 内部的钟给调快了，于是 CPU 的计算跟着这个时钟的节奏，也就自然变快了。当然这个快不是没有代价的，CPU 跑得越快，散热的压力也就越大。就和人一样，超过生理极限，CPU 就会崩溃了。

我们现在回到上面程序 CPU 执行时间的公式。

程序的 CPU 执行时间 = CPU 时钟周期数 × 时钟周期时间

最简单的提升性能方案，自然缩短时钟周期时间，也就是提升主频。换句话说，就是换一块好一点的 CPU。不过，这个是我们这些软件工程师控制不了的事情，所以我们就把目光挪到了乘法的另一个因子——CPU 时钟周期数上。如果能够减少程序需要的 CPU 时钟周期数量，一样能够提升程序性能。

对于 CPU 时钟周期数，我们可以再做一个分解，把它变成“指令数×**每条指令的平均时钟周期数**（Cycles Per Instruction，简称 CPI）”。不同的指令需要的 Cycles 是不同的，加法和乘法都对应着一条 CPU 指令，但是乘法需要的 Cycles 就比加法要多，自然也就慢。在这样拆分了之后，我们的程序的 CPU 执行时间就可以变成这样三个部分的乘积。

程序的 CPU 执行时间 = 指令数×CPI×Clock Cycle Time

因此，如果我们想要解决性能问题，其实就是要优化这三者。

1. 时钟周期时间，就是计算机主频，这个取决于计算机硬件。我们所熟知的**摩尔定律**就一直在不停地提高我们计算机的主频。比如说，我最早使用的 80386 主频只有 33MHz，现在手头的笔记本电脑就有 2.8GHz，在主频层面，就提升了将近 100 倍。
2. 每条指令的平均时钟周期数 CPI，就是一条指令到底需要多少 CPU Cycle。在后面讲解 CPU 结构的时候，我们会看到，现代的 CPU 通过流水线技术（Pipeline），让一条指令需要的 CPU Cycle 尽可能地少。因此，对于 CPI 的优化，也是计算机组成和体系结构中的重要一环。
3. 指令数，代表执行我们的程序到底需要多少条指令、用哪些指令。这个很多时候就把挑战交给了编译器。同样的代码，编译成计算机指令时候，就有各种不同的表示方式。

我们可以把自己想象成一个 CPU，坐在那里写程序。计算机主频就好像是你的打字速度，打字越快，你自然可以多写一点程序。CPI 相当于你在写程序的时候，熟悉各种快捷键，越是打同样的内容，需要敲击键盘的次数就越少。指令数相当于你的程序设计得够合理，同样的程序要写的代码行数就少。如果三者皆能实现，你自然可以很快地写出一个优秀的程序，你的“性能”从外面来看就是好的。

总结延伸

好了，学完这一讲，对“性能”这个名词，你应该有了更清晰的认识。我主要对于“响应时间”这个性能指标进行抽丝剥茧，拆解成了计算机时钟周期、CPI 以及指令数这三个独立的指标的乘积，并且为你指明了优化计算机性能的三条康庄大道。也就是，提升计算机主频，优化 CPU 设计使得在单个时钟周期内能够执行更多指令，以及通过编译器来减少需要的指令数。

在后面的几讲里面，我会为你讲解，具体怎么在电路硬件、CPU 设计，乃至指令设计层

面，提升计算机的性能。

[上一页](#)

[下一页](#)