

## 02 | 基础篇：到底应该怎么理解“平均负载”？

2018-11-23 倪朋飞

Linux性能优化实战

[进入课程 >](#)



讲述：冯永吉

时长 13:44 大小 6.30M



你好，我是倪朋飞。


每次发现系统变慢时，我们通常做的第一件事，就是执行 `top` 或者 `uptime` 命令，来了解系统的负载情况。比如像下面这样，我在命令行里输入了 `uptime` 命令，系统也随即给出了结果。

复制代码

```
1 $ uptime
2 02:34:03 up 2 days, 20:14, 1 user, load average: 0.63, 0.83, 0.88
```

但我想问的是，你真的知道这里每列输出的含义吗？

我相信你对前面的几列比较熟悉，它们分别是当前时间、系统运行时间以及正在登录用户数。

 复制代码

```
1 02:34:03          // 当前时间
2 up 2 days, 20:14   // 系统运行时间
3 1 user             // 正在登录用户数
```

而最后三个数字呢，依次则是过去 1 分钟、5 分钟、15 分钟的平均负载（Load Average）。

**平均负载**？这个词对很多人来说，可能既熟悉又陌生，我们每天的工作中，也都会提到这个词，但你真正理解它背后的含义吗？如果你们团队来了一个实习生，他揪住你不放，你能给他讲清楚什么是平均负载吗？

其实，6 年前，我就遇到过这样的场景。公司一个实习生一直追问我，什么是平均负载，我支支吾吾半天，最后也没能解释明白。明明总看到也总会用到，怎么就说不明白呢？后来我静下来想想，其实还是自己的功底不够。

于是，这几年，我遇到问题，特别是基础问题，都会多问自己几个“为什么”，以求能够彻底理解现象背后的本质原理，用起来更灵活，也更有底气。

今天，我就带你来学习下，如何观测和理解这个最常见、也是最重要的系统指标。

我猜一定有人会说，平均负载不就是单位时间内的 CPU 使用率吗？上面的 0.63，就代表 CPU 使用率是 63%。其实并不是这样，如果你方便的话，可以通过执行 `man uptime` 命令，来了解平均负载的详细解释。

简单来说，平均负载是指单位时间内，系统处于**可运行状态**和**不可中断状态**的平均进程数，也就是**平均活跃进程数**，它和 CPU 使用率并没有直接关系。这里我先解释下，可运行状态和不可中断状态这两词儿。

所谓可运行状态的进程，是指正在使用 CPU 或者正在等待 CPU 的进程，也就是我们常用 `ps` 命令看到的，处于 R 状态（Running 或 Runnable）的进程。

不可中断状态的进程则是正处于内核态关键流程中的进程，并且这些流程是不可打断的，比如最常见的是等待硬件设备的 I/O 响应，也就是我们在 ps 命令中看到的 D 状态（Uninterruptible Sleep，也称为 Disk Sleep）的进程。

比如，当一个进程向磁盘读写数据时，为了保证数据的一致性，在得到磁盘回复前，它是不能被其他进程或者中断打断的，这个时候的进程就处于不可中断状态。如果此时的进程被打断了，就容易出现磁盘数据与进程数据不一致的问题。

所以，不可中断状态实际上是系统对进程和硬件设备的一种保护机制。

因此，你可以简单理解为，平均负载其实就是平均活跃进程数。平均活跃进程数，直观上的理解就是单位时间内的活跃进程数，但它实际上是活跃进程数的指数衰减平均值。这个“指数衰减平均”的详细含义你不用计较，这只是系统的一种更快速的计算方式，你把它直接当成活跃进程数的平均值也没问题。

既然平均的是活跃进程数，那么最理想的，就是每个 CPU 上都刚好运行着一个进程，这样每个 CPU 都得到了充分利用。比如当平均负载为 2 时，意味着什么呢？

在只有 2 个 CPU 的系统上，意味着所有的 CPU 都刚好被完全占用。


在 4 个 CPU 的系统上，意味着 CPU 有 50% 的空闲。

而在只有 1 个 CPU 的系统中，则意味着有一半的进程竞争不到 CPU。

## 平均负载为多少时合理

讲完了什么是平均负载，现在我们再回到最开始的例子，不知道你能否判断出，在 uptime 命令的结果里，那三个时间段的平均负载数，多大的时候能说明系统负载高？或是多小的时候就能说明系统负载很低呢？

我们知道，平均负载最理想的情况是等于 CPU 个数。所以在评判平均负载时，**首先你要知道系统有几个 CPU**，这可以通过 top 命令或者从文件 /proc/cpuinfo 中读取，比如：

 复制代码

```
1 # 关于 grep 和 wc 的用法请查询它们的手册或者网络搜索
2 $ grep 'model name' /proc/cpuinfo | wc -l
3 2
```

有了 CPU 个数，我们就可以判断出，当平均负载比 CPU 个数还大的时候，系统已经出现了过载。

不过，且慢，新的问题又来了。我们在例子中可以看到，平均负载有三个数值，到底该参考哪一个呢？

实际上，都要看。三个不同时间间隔的平均值，其实给我们提供了，分析**系统负载趋势**的数据来源，让我们能更全面、更立体地理解目前的负载状况。

打个比方，就像初秋时北京的天气，如果只看中午的温度，你可能以为还在 7 月份的大夏天呢。但如果你结合了早上、中午、晚上三个时间点的温度来看，基本就可以全方位了解这一天的天气情况了。

同样的，前面说到的 CPU 的三个负载时间段也是这个道理。

如果 1 分钟、5 分钟、15 分钟三个值基本相同，或者相差不大，那就说明系统负载很平稳。

但如果 1 分钟的值远小于 15 分钟的值，就说明系统最近 1 分钟的负载在减少，而过去 15 分钟内却有很大的负载。

反过来，如果 1 分钟的值远大于 15 分钟的值，就说明最近 1 分钟的负载在增加，这种增加有可能只是临时性的，也有可能还会持续增加下去，所以需要持续观察。一旦 1 分钟的平均负载接近或超过了 CPU 的个数，就意味着系统正在发生过载的问题，这时就得分析调查是哪里导致的问题，并要想办法优化了。

这里我再举个例子，假设我们在一个单 CPU 系统上看到平均负载为 1.73，0.60，7.98，那么说明在过去 1 分钟内，系统有 73% 的超载，而在 15 分钟内，有 698% 的超载，从整体趋势来看，系统的负载在降低。

那么，在实际生产环境中，平均负载多高时，需要我们重点关注呢？

在我看来，**当平均负载高于 CPU 数量 70% 的时候**，你就应该分析排查负载高的问题了。一旦负载过高，就可能导致进程响应变慢，进而影响服务的正常功能。

但 70% 这个数字并不是绝对的，最推荐的方法，还是把系统的平均负载监控起来，然后根据更多的历史数据，判断负载的变化趋势。当发现负载有明显升高趋势时，比如说负载翻倍了，你再去做分析和调查。

## 平均负载与 CPU 使用率

现实工作中，我们经常容易把平均负载和 CPU 使用率混淆，所以在这里，我也做一个区分。

可能你会疑惑，既然平均负载代表的是活跃进程数，那平均负载高了，不就意味着 CPU 使用率高吗？

我们还是要回到平均负载的含义上来，平均负载是指单位时间内，处于可运行状态和不可中断状态的进程数。所以，它不仅包括了**正在使用 CPU** 的进程，还包括**等待 CPU** 和**等待 I/O** 的进程。

而 CPU 使用率，是单位时间内 CPU 繁忙情况的统计，跟平均负载并不一定完全对应。比如：

CPU 密集型进程，使用大量 CPU 会导致平均负载升高，此时这两者是一致的；

I/O 密集型进程，等待 I/O 也会导致平均负载升高，但 CPU 使用率不一定很高；

大量等待 CPU 的进程调度也会导致平均负载升高，此时的 CPU 使用率也会比较高。

## 平均负载案例分析

下面，我们以三个示例分别来看这三种情况，并用 iostat、mpstat、pidstat 等工具，找出平均负载升高的根源。

因为案例分析都是基于机器上的操作，所以不要只是听听、看看就够了，最好还是跟着我实际操作一下。

## 你的准备

下面的案例都是基于 Ubuntu 18.04，当然，同样适用于其他 Linux 系统。我使用的案例环境如下所示。

机器配置：2 CPU，8GB 内存。

预先安装 stress 和 sysstat 包，如 apt install stress sysstat。

在这里，我先简单介绍一下 stress 和 sysstat。

stress 是一个 Linux 系统压力测试工具，这里我们用作异常进程模拟平均负载升高的场景。

而 sysstat 包含了常用的 Linux 性能工具，用来监控和分析系统的性能。我们的案例会用到这个包的两个命令 mpstat 和 pidstat。

mpstat 是一个常用的多核 CPU 性能分析工具，用来实时查看每个 CPU 的性能指标，以及所有 CPU 的平均指标。


pidstat 是一个常用的进程性能分析工具，用来实时查看进程的 CPU、内存、I/O 以及上下文切换等性能指标。

此外，每个场景都需要你开三个终端，登录到同一台 Linux 机器中。

实验之前，你先做好上面的准备。如果包的安装有问题，可以先在 Google 一下自行解决，如果还是解决不了，再来留言区找我，这事儿应该不难。

另外要注意，下面的所有命令，我们都是默认以 root 用户运行。所以，如果你是用普通用户登陆的系统，一定要先运行 sudo su root 命令切换到 root 用户。

如果上面的要求都已经完成了，你可以先用 uptime 命令，看一下测试前的平均负载情况：

 复制代码

```
1 $ uptime
2 ..., load average: 0.11, 0.15, 0.09
```

## 场景一：CPU 密集型进程

首先，我们在第一个终端运行 stress 命令，模拟一个 CPU 使用率 100% 的场景：

```
1 $ stress --cpu 1 --timeout 600
```

接着，在第二个终端运行 `uptime` 查看平均负载的变化情况：

```
1 # -d 参数表示高亮显示变化的区域
2 $ watch -d uptime
3 ..., load average: 1.00, 0.75, 0.39
```

最后，在第三个终端运行 `mpstat` 查看 CPU 使用率的变化情况：

```
1 # -P ALL 表示监控所有 CPU，后面数字 5 表示间隔 5 秒后输出一组数据
2 $ mpstat -P ALL 5
3 Linux 4.15.0 (ubuntu) 09/22/18 _x86_64_ (2 CPU)
4 13:30:06      CPU      %usr    %nice    %sys %iowait    %irq    %soft    %steal    %guest    %gnice
5 13:30:11    all    50.05     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
6 13:30:11       0     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
7 13:30:11       1   100.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
```

从终端二中可以看到，1 分钟的平均负载会慢慢增加到 1.00，而从终端三中还可以看到，正好有一个 CPU 的使用率为 100%，但它的 `iowait` 只有 0。这说明，平均负载的升高正是由于 CPU 使用率为 100%。


那么，到底是哪个进程导致了 CPU 使用率为 100% 呢？你可以使用 `pidstat` 来查询：

```
1 # 间隔 5 秒后输出一组数据
2 $ pidstat -u 5 1
3 13:37:07      UID        PID      %usr %system  %guest    %wait    %CPU   CPU  Command
4 13:37:12        0       2962   100.00    0.00    0.00    0.00  100.00    1  stress
```

从这里可以明显看到，stress 进程的 CPU 使用率为 100%。


## 场景二：I/O 密集型进程

首先还是运行 stress 命令，但这次模拟 I/O 压力，即不停地执行 sync：

 复制代码


```
1 $ stress -i 1 --timeout 600
```

还是在第二个终端运行 uptime 查看平均负载的变化情况：

 复制代码

```
1 $ watch -d uptime
2 ..., load average: 1.06, 0.58, 0.37
```


然后，第三个终端运行 mpstat 查看 CPU 使用率的变化情况：

 复制代码

```
1 # 显示所有 CPU 的指标，并在间隔 5 秒输出一组数据
2 $ mpstat -P ALL 5 1
3 Linux 4.15.0 (ubuntu)      09/22/18      _x86_64_      (2 CPU)
4 13:41:28      CPU      %usr      %nice      %sys %iowait      %irq      %soft      %steal      %guest      %gnice
5 13:41:33      all       0.21       0.00      12.07   32.67       0.00       0.21       0.00       0.00       0.00
6 13:41:33        0       0.43       0.00     23.87   67.53       0.00       0.43       0.00       0.00       0.00
7 13:41:33        1       0.00       0.00       0.81    0.20       0.00       0.00       0.00       0.00       0.00
```

从这里可以看到，1 分钟的平均负载会慢慢增加到 1.06，其中一个 CPU 的系统 CPU 使用率升高到了 23.87，而 iowait 高达 67.53%。这说明，平均负载的升高是由于 iowait 的升高。

那么到底是哪个进程，导致 iowait 这么高呢？我们还是用 pidstat 来查询：

 复制代码

```
1 # 间隔 5 秒后输出一组数据，-u 表示 CPU 指标
```



```

2 $ pidstat -u 5 1
3 Linux 4.15.0 (ubuntu)      09/22/18      _x86_64_      (2 CPU)
4 13:42:08      UID      PID      %usr %system %guest  %wait   %CPU   CPU   Command
5 13:42:13      0       104      0.00  3.39   0.00   0.00   3.39    1  kworker/1:1H
6 13:42:13      0       109      0.00  0.40   0.00   0.00   0.40    0  kworker/0:1H
7 13:42:13      0      2997      2.00 35.53   0.00   3.99  37.52    1  stress
8 13:42:13      0      3057      0.00  0.40   0.00   0.00   0.40    0  pidstat


```

可以发现，还是 stress 进程导致的。

### 场景三：大量进程的场景

当系统中运行进程超出 CPU 运行能力时，就会出现等待 CPU 的进程。

比如，我们还是使用 stress，但这次模拟的是 8 个进程：

 复制代码

```
1 $ stress -c 8 --timeout 600
```

由于系统只有 2 个 CPU，明显比 8 个进程要少得多，因而，系统的 CPU 处于严重过载状态，平均负载高达 7.97：

 复制代码

```

1 $ uptime
2 ...,  load average: 7.97, 5.93, 3.02

```

接着再运行 pidstat 来看一下进程的情况：

 复制代码

```

1 # 间隔 5 秒后输出一组数据
2 $ pidstat -u 5 1
3 14:23:25      UID      PID      %usr %system %guest  %wait   %CPU   CPU   Command
4 14:23:30      0      3190      25.00  0.00   0.00  74.80  25.00    0  stress
5 14:23:30      0      3191      25.00  0.00   0.00  75.20  25.00    0  stress
6 14:23:30      0      3192      25.00  0.00   0.00  74.80  25.00    1  stress
7 14:23:30      0      3193      25.00  0.00   0.00  75.00  25.00    1  stress

```

8	14:23:30	0	3194	24.80	0.00	0.00	74.60	24.80	0	stress
9	14:23:30	0	3195	24.80	0.00	0.00	75.00	24.80	0	stress
10	14:23:30	0	3196	24.80	0.00	0.00	74.60	24.80	1	stress
11	14:23:30	0	3197	24.80	0.00	0.00	74.80	24.80	1	stress
12	14:23:30	0	3200	0.00	0.20	0.00	0.20	0.20	0	pidstat

可以看出，8 个进程在争抢 2 个 CPU，每个进程等待 CPU 的时间（也就是代码块中的 %wait 列）高达 75%。这些超出 CPU 计算能力的进程，最终导致 CPU 过载。

## 小结

分析完这三个案例，我再来归纳一下[平均负载的理解](#)。

平均负载提供了一个快速查看系统整体性能的手段，反映了整体的负载情况。但只看平均负载本身，我们并不能直接发现，到底是哪里出现了瓶颈。所以，在理解平均负载时，也要注意：

平均负载高有可能是 CPU 密集型进程导致的；

平均负载高并不一定代表 CPU 使用率高，还有可能是 I/O 更繁忙了；

当发现负载高的时候，你可以使用 mpstat、pidstat 等工具，辅助分析负载的来源。

## 思考

最后，我想邀请你一起来聊聊你所理解的平均负载，当你发现平均负载升高后，又是怎么分析排查的呢？你可以结合我前面的讲解，来总结自己的思考。欢迎在留言区和我讨论。

# Linux 性能优化实战

10 分钟帮你找到系统瓶颈

倪朋飞

微软资深工程师  
Kubernetes 项目维护者



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 01 | 如何学习Linux性能优化？

下一篇 03 | 基础篇：经常说的 CPU 上下文切换是什么意思？（上）

## 精选留言 (363)

 写留言



倪朋飞 置顶

2018-11-25

 145

没想到大家的热情这么高，太激动了。统一回复一下案例中的几个问题：

1. iowait无法升高的问题，是因为案例中stress使用的是 sync() 系统调用，它的作用是刷新缓冲区内内存到磁盘中。对于新安装的虚拟机，缓冲区可能比较小，无法产生大的IO压力，这样大部分就都是系统调用的消耗了。所以，你会看到只有系统CPU使用率升高。...

展开 ▾



南北少卿 置顶

2018-11-23

 7

老师,在跟着操作场景三的时候,使用命令pidstat -u 5 1,并没有出%wait的值,我用的是阿里

云centos(CentOS Linux release 7.5.1804 (Core)  
) ,Linux 3.10.0-693.2.2.el7.x86\_64 (izbp13056tlb7huifh6gm3z) 11/23/2018  
\_x86\_64\_ (1 CPU)  
Average: UID PID %usr %system %guest %CPU CPU Command...  
展开 ∨

作者回复: 版本的问题, centos自带的sysstat版本稍微老一点, 11.5.5之后才增加的这个选项



longhaiqwe 置顶

2018-11-23

👍 5

倪老师提到的软件, 最好都用源码安装吧, 版本比较新, 尤其是centos的同学们。

作者回复: 📁 源码或者RPM升级都可以



冯宇

2018-11-23

👍 64

我一直用htop看负载, 因为它更直接(在F2配置中勾选所有开关项, 打开颜色区分功能), 不同的负载会用不同的颜色标识。比如cpu密集型的应用, 它的负载颜色是绿色偏高, iowait的操作, 它的负载颜色是红色偏高等等, 根据这些指标再用htop的sort就很容易定位到有问题的进程。还有个更好用的atop命令, 好像是基于sar的统计生成的报告, 直接就把有问题的进程标红了, 更直观

展开 ∨

作者回复: 📁 这几个工具也很好用



dancer

2018-12-04

👍 60

学习笔记:

一、什么是平均负载

正确定义: 单位时间内, 系统中处于可运行状态和不可中断状态的平均进程数。

错误定义: 单位时间内的cpu使用率。

可运行状态的进程: 正在使用cpu或者正在等待cpu的进程, 即ps aux命令下STAT处于R...

展开 ∨



slam

2018-11-23

👍 53

io高的例子，为何还是通过pidstat 看cpu？不应该是看哪个进程io高吗？只看sys占比就可以确认了？这里不是很理解

展开 ∨

作者回复: 👤 眼光很毒，的确更好的方法是进程的io情况，比如可以试试pidstat -d



shellmode

2018-11-23

👍 42

在 sched/loadavg.c 中计算平均值的算法为EMA，这种算法的目的主要是“距离目标预测窗口越近，则数据的价值越高，对未来影响越大”

如果说“更快的计算”应该只有里面的 fixed\_power\_int 函数用  $O(\log n)$  的时间来算  $x^n$  ...

展开 ∨

作者回复: 👤 源码级分析



雙

2018-11-26

👍 41

还是建议用top和ps或者lsof来分析，因为一般线上的机器不会额外安装这之外的工具，而且很多公司用堡垒机登录上去之后其他的基本上都用不了，用其自带的最保险



DJH

2018-11-23

👍 31

老师你好，请教一个问题，现在大多数CPU有超线程能力，在计算和评估平均负载的时候，CPU的核数是指物理核数，还是超线程功能的逻辑核数？

展开 ∨

作者回复: 逻辑核数



孤岛

2018-11-23

👍 24

我有一点自己的理解，请老师指正。CPU比喻成一辆地铁，正在使用CPU的进程就是在地铁上的人；等待CPU的进程就是在下一站等地铁来的人；等待I/O的进程就是在下一站要上车和下车的人，虽然现在对CPU没影响，可未来会影响，所以也要考虑到平均负载上。

展开 ∨

作者回复: 很好的比喻，补充一下这个地铁的乘客容量就是CPU个数



每天晒白牙

2018-11-24

👍 16

Centos7系统

安装stress ( Linux系统压力测试工具 ) 和sysstat ( Linux性能工具 )

yum install stress 一直找不到镜像处理方式 所以用了rpm方式安装...

展开 ∨

作者回复: 👍



白华

2018-11-23

👍 16

进行实验二 stress -i 1 --timeout 600模拟sync，平均负载确实上升了，但是在mpstst - P ALL 5 1查看是sys那一列接近100% 而不是iowait

展开 ∨



谁都别拦着...

2018-11-23

👍 14

有个疑问，就像置顶评论说需要最新的版本才能看到某些系统运行指标，但是常常出问题的线上机器我们作为开发工程师并没有root权限去安装，找运维同事给装他们也不一定答

应开这个口子，有可能用系统自带的或者说各类linux发行版都比较通用的系统命令（例如 uptime）来完成系统状态的查看吗？

展开 ▾



Leon 📷

2018-11-23

👍 14

老师你好，我在centos下模拟IO等待比较高场景，发现mpstat -P ALL 5 1没有出现iowait很高的情况

watch -d uptime指令是这样

10:47:15 up 20 min, 5 users, load average: 1.34, 0.85, 0.52

mpstat -P ALL 5 1指令结果是这样...

展开 ▾



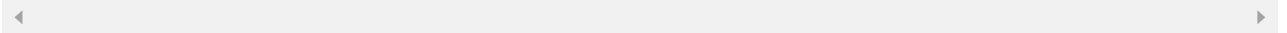
萬萬沒想到

2018-11-23

👍 11

解开了我多年来对平均负载的疑问，就凭这点，花的钱也值了！！

作者回复: 😊



C家族铁粉

2018-11-23

👍 11

『D3打卡』

本来想偷个懒不动手了，结果听着听着音频，就激动地爬起来去操作了。我就是那个把 load average和CPU使用率搞混的人，虽然以前每次都会用uptime查一下，但是只能隐约感觉去判断。老师说的确实很对，最简单的概念都不能清楚理解，复杂的系统关系更难...

展开 ▾

作者回复: 总结的很好👍 补充一下，工具的使用最好先查一查手册，网络上的搜索结果不一定完全准确。



低头走路，...

2018-11-23

👍 10

有些建议，cpu和cpu核心数不是一个概念，也不能划等号吧。

---



小美

2018-11-26

👍 9

不可中断状态的进程则是正处于内核态关键流程中的进程，并且这些流程是不可打断的，比如最常见的是等待硬件设备的 I/O 响应。----linux是有I/O中断的，为什么等待I/O响应却是不可中断的呢？那I/O中断用来干什么呢？

展开 ▾

---



一步

2018-11-25

👍 9

老师我有个问题哈：就是

总核数 = 物理CPU个数 X 每个物理CPU的核数

总逻辑CPU数 = 物理CPU个数 X 每个物理CPU的核数 X 超线程数

这里的平均负载应该是 总核数比较，还是核总逻辑CPU数 比较呢？

展开 ▾

---



威

2018-11-24

👍 9

请问老师，处于不可中断状态的进程，还会占用CPU时钟周期吗