了解vEdge 5000/1000/100B和vEdge Cloud平台的vManage报告的高CPU利用率

目录

简介

本文描述您为什么在**顶层**也许为vEdge 5000/1000/100B发现在vManage和vEdge尽管是的平台的性能的Cloud平台报告的高CPU使用方法正常的没有如查看报告的高CPU。

了解被报告关于vEdge 5000/1000/100B和vEdge Cloud平台的高CPU利用率

使用17.2.x及以后版本，高CPU和内存消耗量vEdge和vEdge Cloud平台的可以被观察。这在一个特定设备的vManage显示板被注意。有时，这也导致增加号码戒备和警告在vManage。

说明

报告的高CPU使用方法，当设备通常执行与正常时，低，或者没有负荷的原因归结于在用于的公式上的一个变化为了计算使用方法。 使用17.2版本， CPU利用率根据**负载平均**被计算从在vEdge的**show system status**。

vManage显示设备的实时CPU利用率。它拉**1分钟平均的[min1_avg]，**并且**5分钟平均为**根据历史数据的**[min5_avg]。负载平均**，根据定义，包括造成利用率计算的多种事和不仅CPU周期。 例如，IO等待时间，处理等待时间，并且其他值考虑，当您提交平台时的此值。在这种情况下，您忽略为CPU状态和CPU值显示的值在**top命令**从vShell。

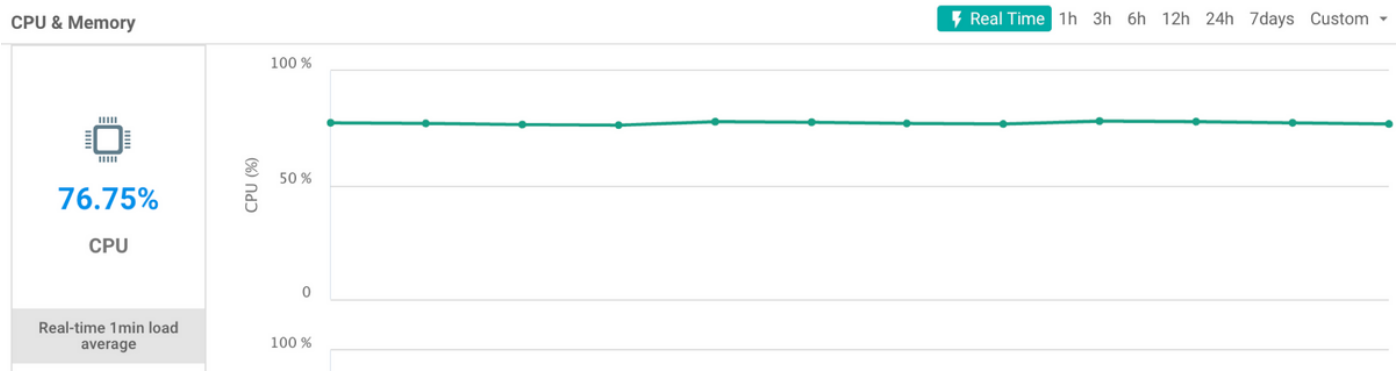
这是关于怎样的一个示例CPU利用率，实际上是**1个分钟负载平均**，在vManage显示板得到计算和显示：

当您检查从vEdge CLI时的负荷，这能被看到：

```
vEdge# show system status | include Load
Load average:            1 minute: 3.10, 5 minutes: 3.06, 15 minutes: 3.05
Load average:            1 minute: 3.12, 5 minutes: 3.07, 15 minutes: 3.06
Load average:            1 minute: 3.13, 5 minutes: 3.08, 15 minutes: 3.07
Load average: 1 minute: 3.10, 5 minutes: 3.07, 15 minutes: 3.05
```

在这种情况下， CPU利用率根据负荷**平均值/#**被计算**核心(vCPUs)**。对于此示例，节点有4个核心。在您由核心前的数量分开负荷平均值被要素100然后转换。 当您平均为从所有核心的负载平均并且乘以100时，您在值到达为~310。占用此值如镜像所显示，并且由4个产量的分界， CPU读77.5%

CPU，与在时间附近被捕获的vManage的实时图形看到的值对齐CLI输出收集了和。



为了看到负载平均和CPU核心的数量在系统的，**顶层的**输出可以从在设备的vShell参见。

在下面的示例中的， vEdge包含4 vCPUs。 第一个核心(**Cpu0**)使用**控制**(进行下去更低的用户利用率)，当依然是的3个核心使用**数据时**：

```
top - 01:14:57 up 1 day,  3:15,  1 user,  load average: 3.06, 3.06, 3.08
Tasks: 219 total,   5 running, 214 sleeping,   0 stopped,   0 zombie
Cpu0  :  1.7%us,  4.0%sy,  0.0%ni, 94.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  : 56.0%us, 44.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  : 54.2%us, 45.8%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  : 59.3%us, 40.7%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7382664k total,  2835232k used,  4547432k free,   130520k buffers
Swap:        0k total,        0k used,        0k free,   587880k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  978 root      20   0 3392m 664m 127m R  100  9.2   1635:21 fp-um-2
  692 root      20   0 3392m 664m 127m R  100  9.2   1635:18 fp-um-1
  979 root      20   0 3392m 664m 127m R  100  9.2   1634:51 fp-um-3
  694 root      20   0 1908m 204m 131m S    1  2.8  15:29.95 ftmd
  496 root      20   0  759m  72m 3764 S    0  1.0   1:31.50 confd
```

为了从vEdge CLI获得CPU的编号，可以使用此命令：

```
vEdge# show system status | display xml | include total_cpu
    <total_cpu_count>4</total_cpu_count>
```

提供得计算的另一个示例在vEdge 1000的vManage显示的值的这里。在您发出从vShell后的**顶层**，I感兴趣为了显示所有核心的负荷：

```
top - 18:19:49 up 19 days,  1:37,  1 user,  load average: 0.55, 0.71, 0.73
```

因为vEdge 1000仅有可用一个CPU的核心，报告的负荷这里是55% (0.55*100)。

与FP um进程的高CPU使用方法

您能从**顶层**有时也注意FP um进程高度运行并且显示100% CPU。这在使用数据层面处理的CPU核心预计。

从参考资料的**top命令**前， 3个核心运行在100% CPU，并且1个核心显示正常利用率：

```
top - 01:14:57 up 1 day,  3:15,  1 user,  load average: 3.06, 3.06, 3.08
```

```
Tasks: 219 total,   5 running, 214 sleeping,   0 stopped,   0 zombie
Cpu0   :  1.7%us,  4.0%sy,  0.0%ni, 94.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1   : 56.0%us, 44.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2   : 54.2%us, 45.8%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3   : 59.3%us, 40.7%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7382664k total,  2835232k used,  4547432k free,   130520k buffers
Swap:        0k total,        0k used,        0k free,   587880k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  978 root      20   0 3392m 664m 127m R  100  9.2   1635:21 fp-um-2
  692 root      20   0 3392m 664m 127m R  100  9.2   1635:18 fp-um-1
  979 root      20   0 3392m 664m 127m R  100  9.2   1634:51 fp-um-3
...
```

此第一个核心(Cpu0)使用**控制**和用于**数据**的三个剩余的核心。正如你在进程列表看到， FP **um**进程使用那些资源。

FP um是使用一轮询模式驱动程序的进程，因此意味着经常坐并且轮询信息包的基础端口，以便能处理所有帧，当被接受。 此进程处理转发并且与在vEdge 1000， vEdge 2000年和vEdge 100的快速路径转发是等同的。Intel使用此轮询模式体系结构根据DPDK框架的高效的信息包处理。由于信息包转发在一个严密的循环实现， CPU保持在或近100%一直。虽然这执行，潜伏期没有通过这些CPU被引入，因为这是期望的工作情况。

关于DPDK轮询的背景信息能他查找得这里。

vEdge Cloud和vEdge 5000平台使用同一转发体系结构并且鉴于此显示同一个工作情况。这是从从**顶部**输出5000的一个示例拉的vEdge。它有28个核心， 2 (Cpu0和Cpu1)使用**控制**(类似vEdge 2000)和26使用**数据**。

```
top - 02:18:30 up 1 day,  7:33,  1 user,  load average: 26.24, 26.28, 26.31
Tasks: 382 total,  27 running, 355 sleeping,   0 stopped,   0 zombie
Cpu0   :  0.7%us,  1.3%sy,  0.0%ni, 98.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1   :  0.7%us,  1.3%sy,  0.0%ni, 98.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2   : 79.4%us, 20.6%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3   : 73.4%us, 26.6%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu4   : 73.4%us, 26.6%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
Cpu12  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu13  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu14  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu15  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu16  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu17  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu18  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu19  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu20  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu21  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu22  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu23  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu24  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu25  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu26  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu27  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  32659508k total, 10877980k used, 21781528k free,   214788k buffers
```

```
Swap:        0k total,        0k used,        0k free,  1039104k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2028 root      20   0 12.1g 668m 124m R  100  2.1   1897:21 fp-um-3
 2029 root      20   0 12.1g 668m 124m R  100  2.1   1897:22 fp-um-4
 2030 root      20   0 12.1g 668m 124m R  100  2.1   1897:12 fp-um-5
 2031 root      20   0 12.1g 668m 124m R  100  2.1   1897:22 fp-um-6
 2032 root      20   0 12.1g 668m 124m R  100  2.1   1897:22 fp-um-7
 2034 root      20   0 12.1g 668m 124m R  100  2.1   1897:22 fp-um-9
 2035 root      20   0 12.1g 668m 124m R  100  2.1   1897:21 fp-um-10
 2038 root      20   0 12.1g 668m 124m R  100  2.1   1897:21 fp-um-13
 2040 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-15
 2041 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-16
 2043 root      20   0 12.1g 668m 124m R  100  2.1   1897:22 fp-um-18
 2045 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-20
 2052 root      20   0 12.1g 668m 124m R  100  2.1   1897:18 fp-um-27
 2033 root      20   0 12.1g 668m 124m R  100  2.1   1897:21 fp-um-8
 2036 root      20   0 12.1g 668m 124m R  100  2.1   1897:21 fp-um-11
 2037 root      20   0 12.1g 668m 124m R  100  2.1   1897:21 fp-um-12
 2039 root      20   0 12.1g 668m 124m R  100  2.1   1897:09 fp-um-14
 2042 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-17
 2044 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-19
 2046 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-21
 2047 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-22
 2048 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-23
 2049 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-24
 2050 root      20   0 12.1g 668m 124m R  100  2.1   1897:22 fp-um-25
 2051 root      20   0 12.1g 668m 124m R  100  2.1   1897:23 fp-um-26
 1419 root      20   0  116m 5732 2280 S    0  0.0   0:02.00 chmgrd
 1323 root      20   0  753m  70m 3764 S    0  0.2   1:51.20 confd
 1432 root      20   0 1683m 172m 134m S    0  0.5   0:58.91 fpmd
```

这里，因为26出于28个处理器运行在100%由于FP um进程，负载平均总是高。

结论

在vManage的报告的CPU使用情况17.2.x在17.2.7之前的版本的不是实际CPU使用情况，然而根据负载平均被计算。这可能导致在了解报告的值的混乱和导致假告警与高CPU，当平台正常运行与正常时，低，或者没有实际数据流/network负荷有关。

此工作情况因17.2.7和18.2改变/修改发布这样根据从**顶层的**cpu_user读可以当前是准确的CPU读。

问题在17.2版本注释也被提及。