使用"PERF"；收集和绘制CPU统计信息NSO中的工具

目录

简介

本文档介绍如何在NSO主机上使用perf工具调查性能问题。

先决条件

要求

Cisco 建议您了解以下主题：

- 基本Linux/Unix命令行使用
- NSO（网络服务协调器）系统体系结构和操作
- CPU分析和分析概念
- 熟悉性能故障排除工作流程

使用的组件

本文档中的信息基于以下软件和硬件版本：

- 在受支持的Unix/Linux主机上的NSO系统或本地安装
- Linux发行版，如Ubuntu、Debian、Fedora或RedHat衍生版
- perf工具（Linux性能分析工具）

本文档中的信息在特定实验室环境设备上创建。本文档中使用的所有设备最初均采用原始（默认）配置。如果您的网络处于活动状态，请确保您了解所有命令的潜在影响。

背景信息

Perf是Linux中功能强大的性能分析工具，主要用于分析CPU。它通过捕获和分析低级功能的负载来深入了解CPU当前的工作情况。这有助于确定哪些功能或进程占用了CPU，对于查明性能瓶颈至关重要。

Perf还可以生成火焰图，这是一种特殊图表，可以直观地表示程序的哪些部分占用了最多的CPU时间。火焰图可以更轻松地找出代码中需要优化的区域。

重要的是，性能还会按照NSO业务部门(BU)的建议，包含在内存不足(OOM)案例的主要数据收集核对表中。 有关OOM故障排除的更多详细指导，请联系Cisco TAC。

排除NSO性能问题的性能使用故障

本节提供在NSO主机上安装、使用和分析来自perf工具的数据以排除性能问题的综合工作流程。

安装性能

步骤 1：在Linux发行版上安装性能。使用适用于您的操作系统的命令：

对于Ubuntu:

```
apt-get update && apt-get -y install linux-tools-generic
```

对于Debian:

```
apt-get update && apt-get -y install linux-perf
```

对于Fedora/RedHat衍生工具：

```
dnf install -y perf
```

有关安装perf时的已知警告的详细信息，请联系Cisco TAC团队。

对数据进行采样

步骤 1：确定主要NSO流程。

使用以下给定命令查找NSO进程(ncs.smp):

```
ps -ef | grep ncs\.smp
```

示例输出：

```
root    120829       1  16 13:23 ? 00:11:08 /opt/ncs/current/lib/ncs/erts/bin/ncs.smp -K true -P 277140
root    121424   120604  0 14:30 pts/0 00:00:00 grep --color=auto ncs.smp
```

步骤 2：或者，您必须使用与NSO关联的主要Java进程的PID，特别是在关注Java操作时。运行：

```
ps -ef | grep NcsJVMLauncher
```

示例输出：

```
root    120903   120833  6 13:32 ? 00:03:40 java -classpath :/opt/ncs/current/java/jar/* -Dhost=127.0.0
root    121435   120604  0 14:33 pts/0 00:00:00 grep --color=auto NcsJVMLauncher
```

步骤 3：执行有问题的测试用例或使用例以验证性能方案。

步骤 4：在不同的终端窗口中，根据相关进程ID(PID)运行perf。 使用以下给定命令格式，用上面获得的PID替换XX、YY、ZZ:

```
perf record -F 100 -g -p XX,YY,ZZ
```

例如，要分析系统范围并收集特定PID的99Hz呼叫图：

```
perf record -a -g -F 99 -p 120829,120903
```

示例输出：

```
Warning:
PID/TID switch overriding SYSTEM
```

选项说明：

- -a:所有CPU;从所有CPU进行系统范围的收集（如果未指定目标，则为默认值）。
- -g:捕获调用图（堆栈跟踪）。 标识调用函数的位置。
- -F:采样频率(Hz)。频率越高，精度越高，但开销也越大。
- -p:指定进程ID。

步骤 5：收集完样本后，使用Ctrl+C停止执行操作：

```
^C
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.646 MB perf.data (4365 samples) ]
```

您现在可以在当前目录中看到perf.data文件。

步骤 6：使用以下命令生成摘要报告：

```
perf report -n --stdio > perf_report.txt
```

选项说明：

- -n:显示不带分组的符号（平面视图）。
- — 音频：强制输出到标准输出（终端）。

此时，您必须保存这两个文件(perf.data和perf_report.txt)并与您的支持联系人共享它们，然后再继续进行进一步分析。

如果捕获成功，perf_report.txt将显示表示分层调用图的树状结构。百分比有助于确定占用大部分CPU时间的热点。

示例摘录：

```
# Children      Self       Samples  Command          Shared Object         Symbol
# ........  ........  ............  ...............  ....................  ............................
#    30.61%     0.00%             0  C2 CompilerThre  libc.so.6             [.] start_thread
#            ---start_thread
#               thread_native_entry(Thread*)
#               Thread::call_run()
#               JavaThread::thread_main_inner()
#               CompileBroker::compiler_thread_loop()
#                           --30.58%--CompileBroker::invoke_compiler_on_method(CompileTask*)
#                                     --30.47%--C2Compiler::compile_method(ciEnv*, ciMethod*, int, bool
#                                     Compile::Compile(ciEnv*, ciMethod*, int, bool, bool, bool, bool, |
#                                               |--17.57%--Compile::Code_Gen()
#                                               |          |--12.46%--PhaseChaitin::Register_Allocate()
#                                               |          |          |--2.79%--PhaseChaitin::build_ifg_
```

```
#                                                    |          |          |          --1.05%--PhaseChaitin
#                                                    |          |--1.49%--PhaseChaitin::Split(unsigned int,
#                                                    |          |--1.26%--PhaseChaitin::post_allocate_copy_r
```

解释：

- 进程/线程：正在分析C2 CompilerThre线程。
- 总CPU使用率：此线程占用CPU时间的30.61%。
- 功能流：线程以`start_thread`开头，并跨多个层委托工作。大部分CPU时间(30.47%)花费在`C2Compiler::compile_method`中，表示可能存在热点。

生成火焰图

步骤 1：在定义的时间间隔（例如，60秒）内从所有CPU和进程生成性能示例：

```
perf record -a -g -F 99 sleep 60
```

示例输出：

```
[ perf record: Woken up 32 times to write data ]
[ perf record: Captured and wrote 10.417 MB perf.data (67204 samples) ]
```

步骤 2：将此perf.data文件复制或传输到可从中下载flamegraph模板存储库的主机。

步骤 3：将perf.data文件转换为文本格式：

```
perf script > data.perf
```

步骤 4：克隆FlameGraph GitHub存储库，并将data.perf放入此目录：

```
cp data.perf $PWD/FlameGraph/.
```

步骤 5：折叠用于flametgraph处理的堆栈跟踪：

<#root>

```
cat data.perf | ./stackcollapse-perf.pl > data.perf-folded
```

步骤 6：生成火焰图SVG文件：

<#root>

```
./flamegraph.pl data.perf-folded > data.svg
```

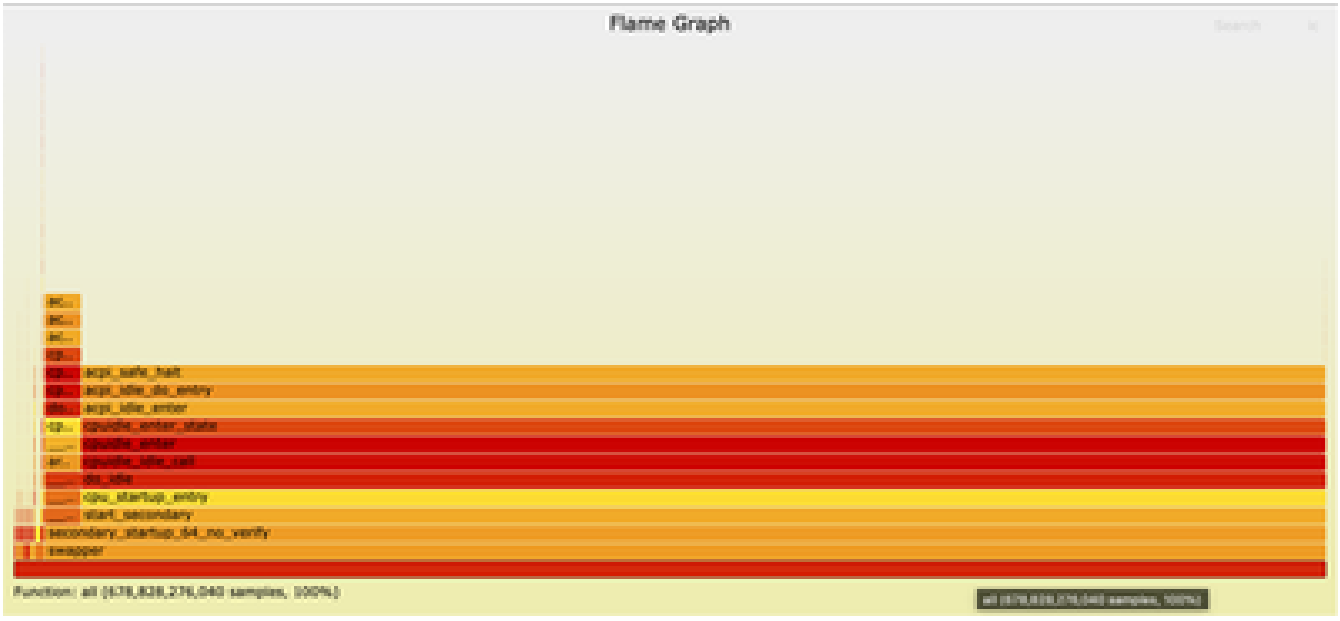
注意：如果在CentOS或RHEL上遇到“无法在@INC中找到open.pm”错误，请安装所需的Perl模块：

```
yum install perl-open.noarch
```

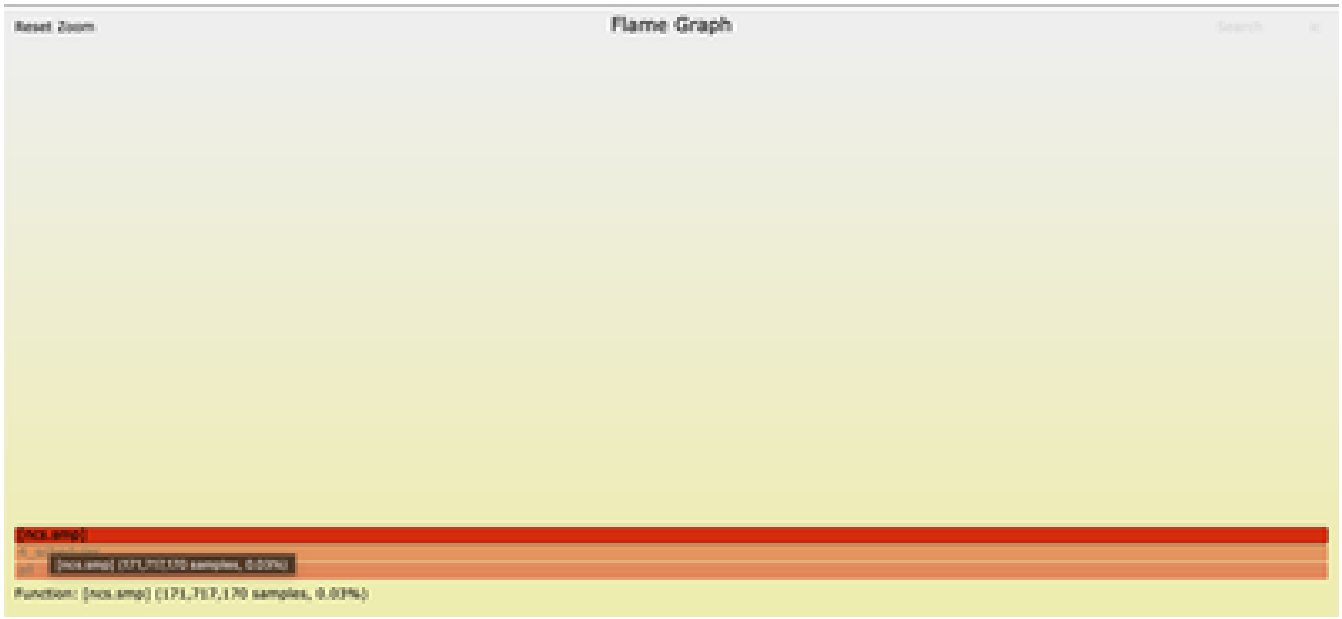
步骤 7：在您的首选Web浏览器中打开data.svg文件以可视化火焰图。

浏览火焰图

在浏览器中打开火焰图文件后，您可以通过单击任何框来与其交互，以放大该函数及其调用堆栈。每个框的长度表示该函数及其调用栈中花费的CPU时间量。此可视化功能可以轻松确定需要优化的热点和区域。



放大ncs.smp:



相关信息

- Linux性能已知警告
- 思科技术支持和下载

关于此翻译
思科采用人工翻译与机器翻译相结合的方式将此文档翻译成不同语言，希望全球的用户都能通过各自的语言得到支持性的内容。

请注意：即使是最好的机器翻译，其准确度也不及专业翻译人员的水平。

Cisco Systems, Inc. 对于翻译的准确性不承担任何责任，并建议您总是参考英文原始文档（已提供链接）。