NSOの"PERF"；ツールを使用したCPU統計の収集とグラフ化

内容

はじめに

このドキュメントでは、パフォーマンスの問題を調査するためにNSOホストでperfツールを使用する方法について説明します。

前提条件

要件

次の項目に関する知識があることが推奨されます。

- 基本的なLinux/UNIXコマンドラインの使用
- NSO(Network Services Orchestrator)システムのアーキテクチャと運用
- CPUプロファイリングと分析の概念
- パフォーマンスのトラブルシューティングワークフローに精通している

使用するコンポーネント

このドキュメントの情報は、次のソフトウェアとハードウェアのバージョンに基づいています。

- サポートされているUNIX/LinuxホストでのNSOシステムまたはローカルインストール
- Ubuntu、Debian、Fedora、RedHatなどのLinuxディストリビューション
- perfツール（Linuxパフォーマンス分析ツール）

このドキュメントの情報は、特定のラボ環境にあるデバイスに基づいて作成されたものです。こ

のドキュメントで使用するすべてのデバイスは、クリアな（デフォルト）設定で作業を開始しています。本稼働中のネットワークでは、各コマンドによって起こる可能性がある影響を十分確認してください。

背景説明

Perfは、主にCPUプロファイリングに使用される、Linuxの強力なパフォーマンス分析ツールです。下位レベルの機能の負荷をキャプチャして分析することで、CPUが現在取り組んでいる内容を把握できます。これは、どの機能またはプロセスがCPUを占有しているかを特定するのに役立ち、パフォーマンスのボトルネックを特定するために不可欠です。

Perfは、プログラムのどの部分が最もCPU時間を使用しているかを視覚的に表す特別なグラフである炎のグラフを生成することもできます。Flameグラフを使用すると、最適化が必要なコード内の領域を簡単に特定できます。

重要な点として、NSO Business Unit(BU)が推奨するメモリ不足(OOM)の場合のメインデータ収集チェックリストにもパフォーマンスが含まれていることが挙げられます。 OOMのトラブルシューティングの詳細については、Cisco TACにお問い合わせください。

NSOのパフォーマンス問題に対するパフォーマンス使用率のトラブルシューティング

このセクションでは、パフォーマンスの問題をトラブルシューティングするために、NSOホスト上でperfツールからデータをインストール、使用、および分析するための包括的なワークフローを提供します。

パフォーマンスのインストール

ステップ1:Linuxディストリビューションにperfをインストールします。使用しているOSに適したコマンドを使用します。

Ubuntuの場合：

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

Debian用：

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

Fedora/RedHat派生物の場合：

```
dnf install -y perf
```

パフォーマンスのインストール時に発生する既知の注意事項の詳細については、Cisco TACチームにお問い合わせください。

データのサンプリング

手順1：主要なNSOプロセスを特定します。

NSOプロセス(ncs.smp)を検索するには、以下の指定されたコマンドを使用します。

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

出力例：

```
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
```

ステップ2：または、特にJava操作に焦点を当てる場合は、NSOに関連付けられたメインJavaプロセスのPIDを使用する必要があります。次のコマンドを実行します。

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

出力例：

```
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
```

ステップ3：問題のあるテストケースまたはユースケースを実行して、パフォーマンスシナリオを検証します。

ステップ4：別のターミナルウィンドウで、関連するプロセスID(PID)に対してperfを実行します。 次のコマンド形式を使用して、XX、YY、ZZを上記で取得したPIDに置き換えます。

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

たとえば、特定のPIDについてシステム全体のプロファイルを作成し、99Hzでコールグラフを収集するには、次の手順を実行します。

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

出力例：

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

オプションの説明：

- -a:全CPU。システム全体ですべてのCPUから収集されます（ターゲットが指定されていない場合はデフォルト）。
- -g:コールグラフ（スタックトレース）をキャプチャします。 関数が呼び出される場所を示します。
- -F:サンプリング周波数(Hz)。周波数を高くすると精度は高くなりますが、オーバーヘッドが増加します。
- -p:プロセスIDを指定します。

ステップ5：サンプルの収集が完了したら、Ctrl+Cで実行を停止します。

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

現在のディレクトリにperf.dataファイルが表示されます。

ステップ6：次のコマンドを使用してサマリーレポートを生成します。

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

オプションの説明：

- -n:記号をグループ化せずに表示します（フラット表示）。
- —stdio:出力を標準出力（端末）に強制します。

この時点で、両方のファイル(perf.dataとperf_report.txt)を保存し、詳細な分析に進む前にサポート担当者と共有する必要があります。

キャプチャが正常に行われた場合、perf_report.txtには階層型コールグラフを表すツリー状の構造が表示されます。パーセンテージは、ほとんどのCPU時間が費やされているホットスポットを特定するのに役立ちます。

抜粋の例：

```
# 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
```

解釈：

- プロセス/スレッド： C2コンパイラスレッドを分析中です。
- 合計CPU使用率：このスレッドはCPU時間の30.61 %を占めています。
- 関数の流れ：スレッドは`start_thread`で始まり、デリゲートが複数のレイヤーにまたがって動作します。CPU時間の大部分(30.47 %)は`C2Compiler::compile_method`で消費され、潜在的なホットスポットを示します。

炎グラフの生成

ステップ1：定義された間隔（例：60秒）内のすべてのCPUとプロセスからパフォーマンスサンプルを生成します。

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

出力例：

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

ステップ2：このperf.dataファイルをコピーするか、ホストに転送します。ホストから flamegraphテンプレートリポジトリをダウンロードできます。

ステップ3: perf.dataファイルをテキスト形式に変換します。

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

ステップ4:FlameGraph GitHubリポジトリをクローニングし、次のディレクトリにdata.perfを配置します。

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

手順5：フラググラフ処理のスタックトレースを折りたたみます。

<#root>

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

ステップ6：炎グラフSVGファイルを生成します。

<#root>

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

注：CentOSまたはRHELで「can not locate open.pm in @INC」エラーが発生した場合は、必要なPerlモジュールをインストールしてください。

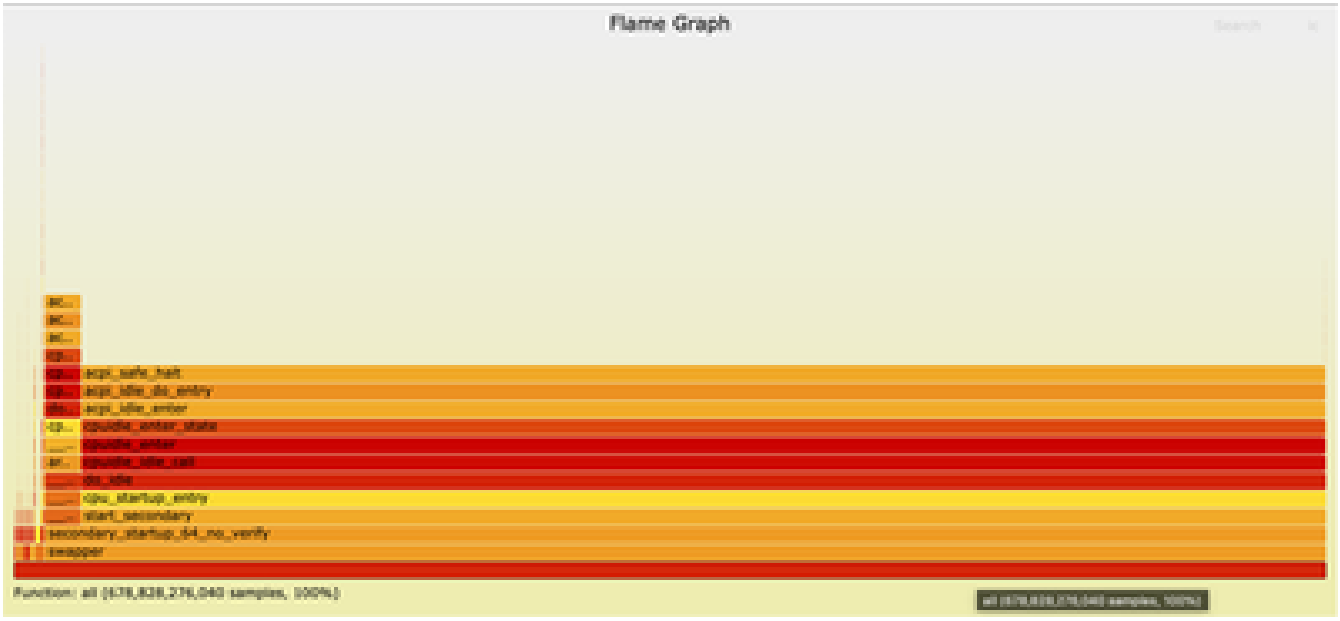
```
yum install perl-open.noarch
```

ステップ7：任意のWebブラウザでdata.svgファイルを開き、炎グラフを表示します。

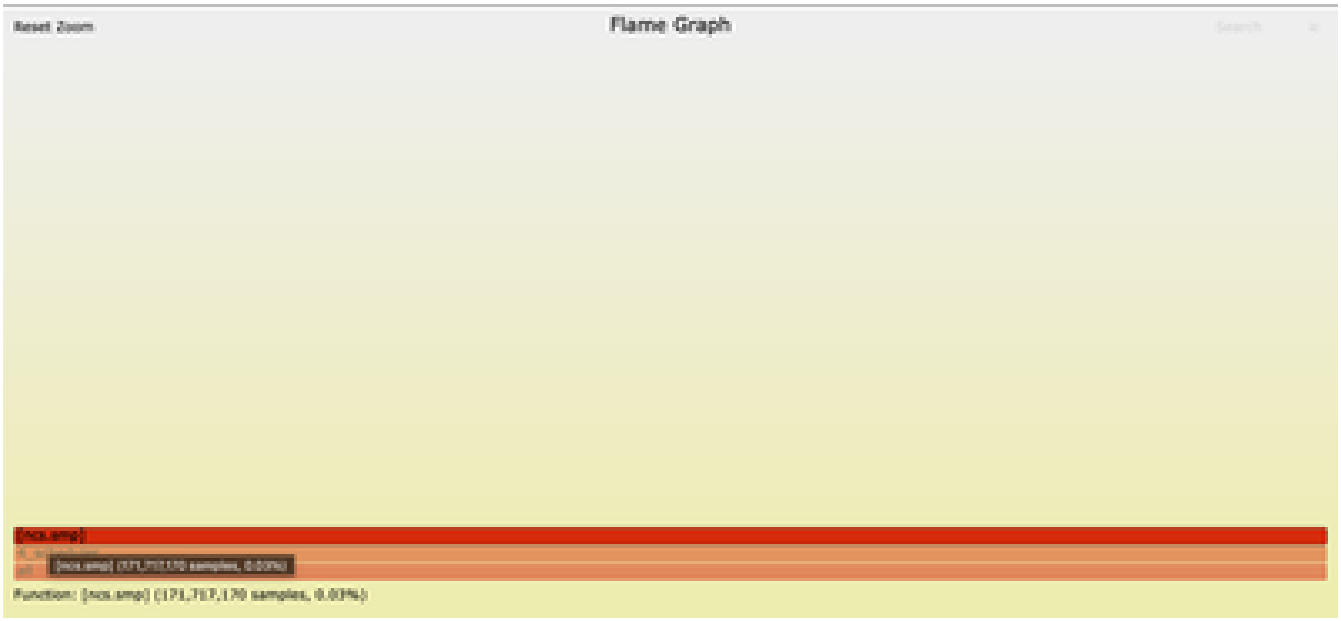
炎グラフを参照する

ブラウザでFlame Graphファイルを開くと、任意のボックスをクリックしてその関数とその呼び出しスタックを拡大表示することで、ファイルを操作できます。各ボックスの長さは、その関数とその呼び出しスタックで使用されたCPU時間の長さを表します。この可視化により、最適化す

べきホットスポットとエリアを簡単に特定できます。



ncs.smpでズーム：



関連情報

- Linuxのパフォーマンスに関する既知の注意事項
- シスコのテクニカルサポートとダウンロード

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。