Collecter et représenter graphiquement les statistiques du processeur à l'aide de " ; PERF" ; Outil dans NSO

Table des matières

Introduction

Ce document décrit comment utiliser l'outil de performances sur les hôtes NSO pour étudier les problèmes de performances.

Conditions préalables

Exigences

Cisco vous recommande de prendre connaissance des rubriques suivantes :

- Utilisation de base de la ligne de commande Linux/Unix
- Architecture et fonctionnement du système NSO (Network Services Orchestrator)
- Concepts de profilage et d'analyse du CPU
- Familiarité avec les workflows de dépannage des performances

Composants utilisés

Les informations contenues dans ce document sont basées sur les versions de matériel et de logiciel suivantes :

- Système NSO ou installation locale sur un hôte Unix/Linux pris en charge
- Les distributions Linux telles que Ubuntu, Debian, Fedora ou les dérivés RedHat

- outil perf (outil d'analyse des performances Linux)

Les informations contenues dans ce document ont été créées à partir des périphériques dans un environnement de laboratoire spécifique. All of the devices used in this document started with a cleared (default) configuration. Si votre réseau est en ligne, assurez-vous de bien comprendre l'incidence possible des commandes.

Informations générales

Perf est un puissant outil d'analyse des performances sous Linux, utilisé principalement pour le profilage du CPU. Il fournit des informations sur les tâches actuelles du processeur en capturant et en analysant la charge des fonctions de niveau inférieur. Cela permet d'identifier les fonctions ou les processus qui occupent le processeur et est essentiel pour identifier les goulots d'étranglement au niveau des performances.

Perf peut également générer des graphiques de flamme, qui sont des graphiques spéciaux qui représentent visuellement quelles parties d'un programme utilisent le plus de temps CPU. Les graphes à flamme facilitent le repérage des zones du code qui doivent être optimisées.

Il est important de noter que les performances sont également incluses dans la liste de contrôle de collecte de données principale pour les cas de mémoire insuffisante (OOM), comme recommandé par la division NSO. Pour obtenir des conseils plus détaillés sur le dépannage OOM, contactez le TAC Cisco.

Dépannage de l'utilisation des performances pour les problèmes NSO

Cette section fournit un workflow complet pour l'installation, l'utilisation et l'analyse des données de l'outil de performances sur les hôtes NSO afin de résoudre les problèmes de performances.

Installer les performances

Étape 1 : Installez les performances sur votre distribution Linux. Utilisez la commande appropriée pour votre système d'exploitation :

Pour Ubuntu :

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

Pour Debian :

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

Pour les dérivés Fedora/RedHat :

```
dnf install -y perf
```

Pour plus d'informations sur les mises en garde connues lors de l'installation des performances, contactez l'équipe Cisco TAC.

Échantillonnage des données

Étape 1 : Identifier le principal processus des ONS.

Utilisez la commande ci-dessous pour localiser le processus NSO (ncs.smp) :

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

Exemple de rapport :

```
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
```

Étape 2 : Vous pouvez également utiliser le PID du processus Java principal lié à NSO, en particulier si vous vous concentrez sur les opérations Java. Exécutez la commande :

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

Exemple de rapport :

```
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
```

Étape 3 : Exécutez le cas de test ou le cas d'utilisation problématique pour valider le scénario de performances.

Étape 4 : Dans une autre fenêtre de terminal, exécutez perf sur les ID de processus (PID) appropriés. Utilisez le format de commande ci-dessous, en remplaçant XX, YY, ZZ par les PID

obtenus ci-dessus :

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

Par exemple, pour profiler l'ensemble du système et rassembler des graphiques d'appels à 99 Hz pour des PID spécifiques :

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

Exemple de rapport :

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

Description des options :

- -a : Tous les processeurs ; Collecte à l'échelle du système à partir de tous les processeurs (valeur par défaut si aucune cible n'est spécifiée).
- -g : Capturer des graphiques d'appels (suivis de pile). Identifie l'endroit où les fonctions sont appelées.
- -F : Fréquence d'échantillonnage en Hz. Les fréquences plus élevées augmentent la précision, mais ajoutent une surcharge.
- -p : Spécifie le ou les ID de processus.

Étape 5 : Lorsque vous avez terminé de collecter des échantillons, arrêtez les performances avec Ctrl+C :

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

Vous voyez maintenant un fichier perf.data dans le répertoire courant.

Étape 6 : Générez un rapport récapitulatif à l'aide de la commande suivante :

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

Description des options :

- -n : Afficher les symboles sans regroupement (vue plate).
- —stdio : Forcer la sortie vers la sortie standard (le terminal).

À ce stade, vous devez enregistrer les deux fichiers (perf.data et perf_report.txt) et les partager avec votre contact de support avant de passer à une analyse plus approfondie.

Si la capture a réussi, perf_report.txt affiche une structure arborescente représentant un graphique d'appels hiérarchique. Les pourcentages vous aident à identifier les points chauds où le temps processeur est le plus utilisé.

Exemple d'extrait :

```
# 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
```

Interprétation :

- Processus/Thread : Le thread C2 CompilerThree est en cours d'analyse.
- Utilisation totale du processeur : Ce thread est responsable de 30,61 % du temps processeur.
- Flux de fonctions : Le thread commence par `start_thread` et les délégués travaillent sur plusieurs couches. La majeure partie du temps CPU (30,47%) est passé dans `C2Compiler::compile_method`, indiquant un hotspot potentiel.

Génération d'un graphique flamme

Étape 1 : Générez un échantillon de performances à partir de tous les processeurs et processus pendant un intervalle défini (par exemple, 60 secondes) :

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

Exemple de rapport :

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

Étape 2 : Copiez ou transférez ce fichier perf.data vers un hôte à partir duquel vous pouvez télécharger le référentiel de modèles flamegraph.

Étape 3 : Convertissez le fichier perf.data au format texte :

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

Étape 4 : Clonez le référentiel FlameGraph GitHub et placez data.perf dans ce répertoire :

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

Étape 5 : Réduire les traces de la pile pour le traitement du flamegraph :

<#root>

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

Étape 6 : Générez le fichier SVG du graphique de flamme :

<#root>

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

Remarque : Si vous rencontrez l'erreur « can not locate open.pm in @INC » sur CentOS ou RHEL, installez le module Perl requis :

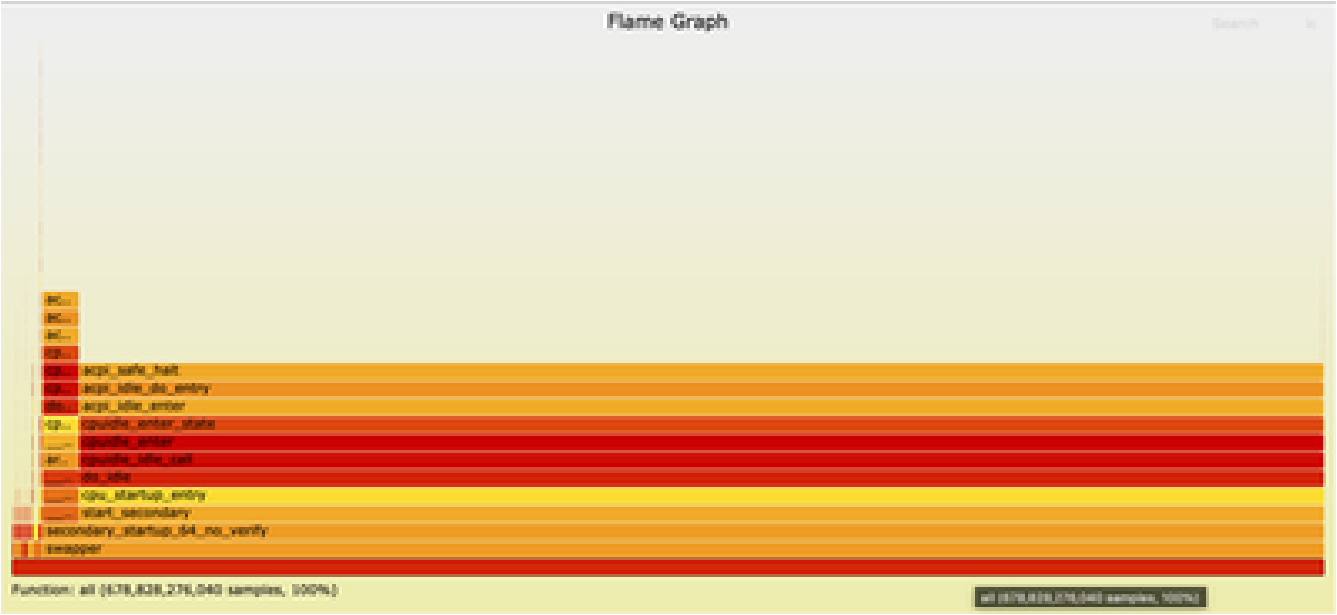
```
yum install perl-open.noarch
```

Étape 7 : Ouvrez le fichier data.svg dans votre navigateur Web préféré pour visualiser le

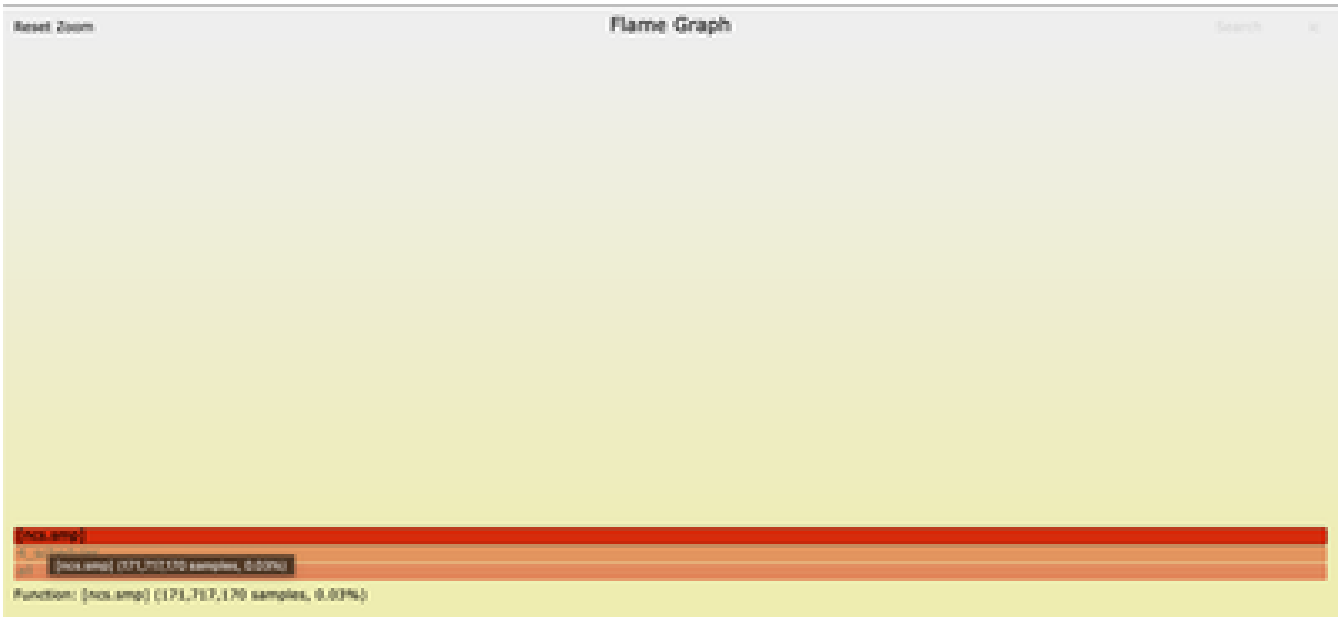
graphique de flamme.

Parcourir le graphique de flamme

Une fois que le fichier graphique de flamme est ouvert dans votre navigateur, vous pouvez interagir avec lui en cliquant sur n'importe quelle zone pour zoomer sur cette fonction et sa pile d'appels. La longueur de chaque case représente le temps processeur passé dans cette fonction et sa pile d'appels. Cette visualisation facilite l'identification des zones sensibles et des zones à optimiser.



Zoomé dans ncs.smp :



Informations connexes

- [Avertissements connus relatifs aux performances Linux]

- Assistance technique de Cisco et téléchargements

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.