Recopilar y representar gráficamente estadísticas de CPU mediante "PERF" Herramienta en NSO

Contenido

Introducción

Este documento describe cómo utilizar la herramienta de rendimiento en los hosts de NSO para investigar problemas de rendimiento.

Prerequisites

Requirements

Cisco recomienda que tenga conocimiento sobre estos temas:

- Uso básico de la línea de comandos de Linux/Unix
- Arquitectura y funcionamiento del sistema NSO (Network Services Orchestrator)
- Conceptos de análisis y definición de perfiles de CPU
- Familiaridad con flujos de trabajo de resolución de problemas de rendimiento

Componentes Utilizados

La información que contiene este documento se basa en las siguientes versiones de software y hardware.

- Sistema NSO o instalación local en un host Unix/Linux compatible
- Distribuciones Linux como Ubuntu, Debian, Fedora o derivados de RedHat

- perf tool (herramienta de análisis de rendimiento de Linux)

La información de este documento se originó a partir de dispositivos dentro de un ambiente de laboratorio específico. Todos los dispositivos que se utilizan en este documento se pusieron en funcionamiento con una configuración verificada (predeterminada). Si tiene una red en vivo, asegúrese de entender el posible impacto de cualquier comando.

Antecedentes

Perf es una potente herramienta de análisis de rendimiento en Linux, utilizada principalmente para la creación de perfiles de CPU. Proporciona información sobre en qué está trabajando actualmente la CPU mediante la captura y el análisis de la carga de funciones de nivel inferior. Esto ayuda a identificar qué funciones o procesos ocupan la CPU y es esencial para identificar cuellos de botella de rendimiento.

El rendimiento también puede generar gráficos de llamas, que son gráficos especiales que representan visualmente qué partes de un programa utilizan más tiempo de CPU. Los gráficos de llamas facilitan la detección de áreas en el código que necesitan optimización.

Y lo que es más importante, el rendimiento también se incluye en la lista de comprobación de recopilación de datos principal para los casos sin memoria (OOM), tal y como recomienda la unidad de negocio (BU) de NSO. Para obtener orientación más detallada sobre la resolución de problemas de OOM, póngase en contacto con Cisco TAC.

Solución de problemas de rendimiento de uso para problemas de rendimiento de NSO

Esta sección proporciona un flujo de trabajo completo para la instalación, el uso y el análisis de datos de la herramienta de rendimiento en hosts de NSO para solucionar problemas de rendimiento.

Rendimiento de instalación

Paso 1: Instale perf en su distribución de Linux. Utilice el comando correspondiente a su sistema operativo:

Para Ubuntu:

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

Para Debian:

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

Para derivados de Fedora/RedHat:

```
dnf install -y perf
```

Para obtener más información sobre advertencias conocidas durante la instalación de perf, póngase en contacto con el equipo del TAC de Cisco.

Muestreo de los datos

Paso 1: Identificar el proceso principal de NSO.

Utilice el siguiente comando para localizar el proceso de NSO (ncs.smp):

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

Ejemplo de salida:

```
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
```

Paso 2: Como alternativa, debe utilizar el PID del proceso Java principal vinculado a NSO, especialmente si se centra en las operaciones de Java. Ejecute:

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

Ejemplo de salida:

```
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
```

Paso 3: Ejecute el caso práctico o de prueba problemático para validar el escenario de rendimiento.

Paso 4: En una ventana de terminal diferente, ejecute perf con los ID de proceso (PID) relevantes.

Utilice el siguiente formato de comando dado, reemplazando XX,YY,ZZ con los PIDs obtenidos anteriormente:

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

Por ejemplo, para generar perfiles de todo el sistema y recopilar gráficos de llamadas a 99 Hz para PID específicos:

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

Ejemplo de salida:

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

Descripciones de opciones:

- -a: Todas las CPU; recopilación de todo el sistema de todas las CPU (valor predeterminado si no se especifica ningún destino).
- -g: Capture gráficos de llamadas (seguimientos de pila). Identifica dónde se llama a las funciones.
- -f: Frecuencia de muestreo en Hz. Las frecuencias más altas aumentan la precisión pero añaden sobrecarga.
- -p: Especifica los Id. de proceso.

Paso 5: Cuando haya terminado de recopilar muestras, detenga el rendimiento con Ctrl+C:

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

Ahora verá un archivo perf.data en el directorio actual.

Paso 6: Genere un informe de resumen con este comando:

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

Descripciones de opciones:

- -n: Mostrar símbolos sin agrupar (vista plana).
- —estudio: Forzar salida a salida estándar (el terminal).

En este momento, debe guardar ambos archivos (perf.data y perf_report.txt) y compartirlos con el contacto de soporte técnico antes de continuar con el análisis.

Si la captura se realizó correctamente, perf_report.txt muestra una estructura en forma de árbol que representa un gráfico jerárquico de llamadas. Los porcentajes le ayudan a identificar zonas activas en las que se emplea la mayor parte del tiempo de la CPU.

Extracto de ejemplo:

```
# 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
```

Interpretación:

- Proceso/Subproceso: Se está analizando el subproceso CompilerThre de C2.
- Uso total de la CPU: Este subproceso es responsable del 30,61% del tiempo de la CPU.
- Flujo de funciones: El subproceso comienza con `start_thread` y los delegados trabajan en varias capas. La mayor parte del tiempo de la CPU (30,47%) se emplea en `C2Compiler::compile_method`, lo que indica una zona de conexión potencial.

Generación de un gráfico de llamas

Paso 1: Generar un ejemplo de rendimiento de todas las CPU y procesos durante un intervalo definido (por ejemplo, 60 segundos):

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

Ejemplo de salida:

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

Paso 2: Copie o transfiera este archivo perf.data a un host desde el que pueda descargar el repositorio de plantillas de flamegraph.

Paso 3: Convierta el archivo perf.data a un formato de texto:

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

Paso 4: Clonar el repositorio de FlameGraph GitHub y colocar data.perf en este directorio:

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

Paso 5: Contraer los seguimientos de pila para el procesamiento de flamegraph:

<#root>

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

Paso 6: Genere el fichero SVG del gráfico de llamas:

<#root>

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

Nota: Si encuentra el error "no se puede localizar open.pm en @INC" en CentOS o RHEL, instale el módulo Perl requerido:

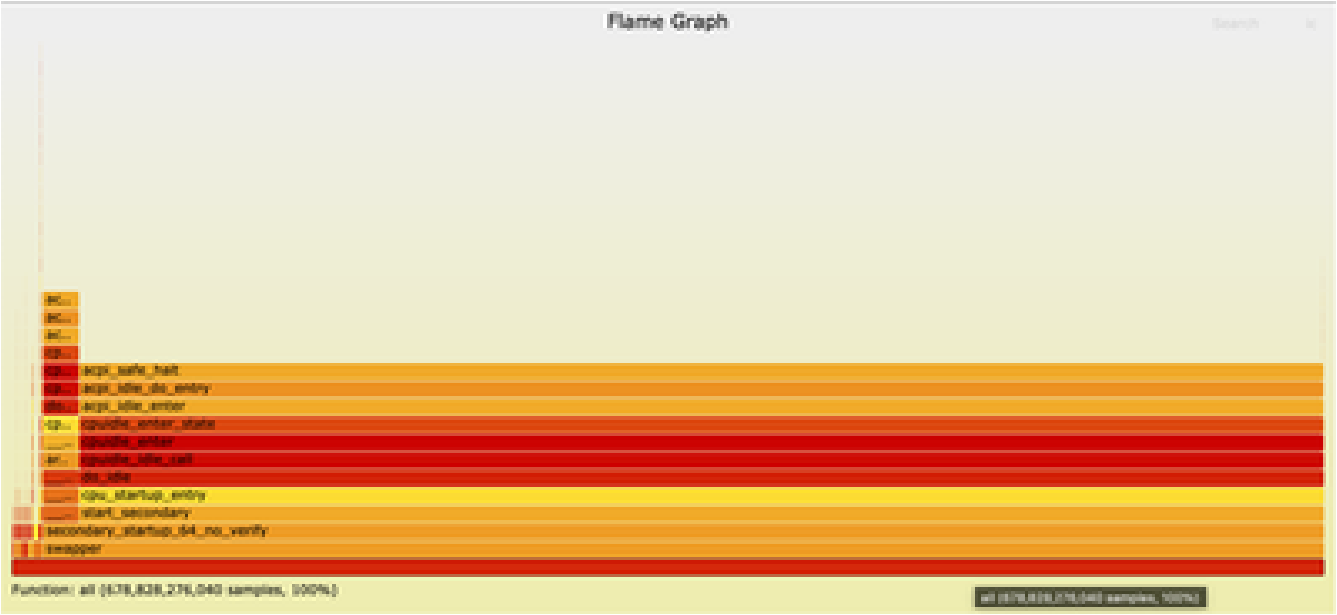
```
yum install perl-open.noarch
```

Paso 7: Abra el archivo data.svg en su navegador web preferido para visualizar el gráfico de

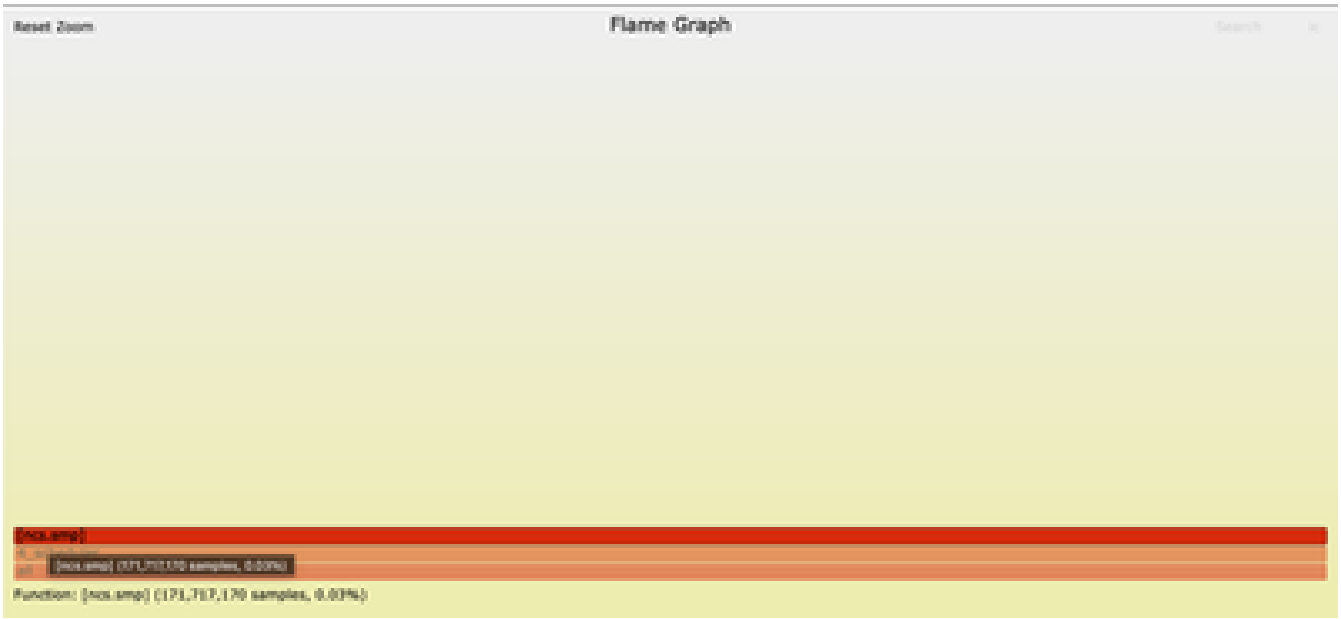
llamas.

Examinar el gráfico de llamas

Una vez abierto el archivo de gráfico de llamas en el explorador, puede interactuar con él haciendo clic en cualquier cuadro para acercar esa función y su pila de llamadas. La longitud de cada cuadro representa la cantidad de tiempo de CPU empleado en esa función y su pila de llamadas. Esta visualización facilita la identificación de puntos de conexión y áreas de optimización.



Ampliado en ncs.smp:



Información Relacionada

- Advertencias conocidas de Linux Perf

- Soporte técnico y descargas de Cisco

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).