Raccogli ed elabora grafici statistiche CPU utilizzando "PERF" Strumento in NSO

Sommario

Introduzione

In questo documento viene descritto come utilizzare lo strumento di prestazioni sugli host NSO per analizzare i problemi di prestazioni.

Prerequisiti

Requisiti

Cisco raccomanda la conoscenza dei seguenti argomenti:

- Utilizzo della riga di comando Linux/Unix di base
- Architettura e funzionamento del sistema NSO (Network Services Orchestrator)
- Concetti di analisi e profilatura della CPU
- Familiarità con i flussi di lavoro per la risoluzione dei problemi relativi alle prestazioni

Componenti usati

Le informazioni fornite in questo documento si basano sulle seguenti versioni software e hardware:

- Sistema NSO o installazione locale su un host Unix/Linux supportato
- Distribuzioni Linux come Ubuntu, Debian, Fedora o derivati RedHat
- strumento perf (strumento di analisi delle prestazioni Linux)

Le informazioni discusse in questo documento fanno riferimento a dispositivi usati in uno specifico

ambiente di emulazione. Su tutti i dispositivi menzionati nel documento la configurazione è stata ripristinata ai valori predefiniti. Se la rete è operativa, valutare attentamente eventuali conseguenze derivanti dall'uso dei comandi.

Premesse

Perf è un potente strumento di analisi delle prestazioni in Linux, utilizzato principalmente per la profilatura della CPU. Fornisce informazioni dettagliate su ciò su cui la CPU sta attualmente lavorando acquisendo e analizzando il carico delle funzioni di livello inferiore. Ciò consente di identificare le funzioni o i processi che occupano la CPU ed è essenziale per individuare i colli di bottiglia delle prestazioni.

Perf può anche generare grafici a fiamma, che sono grafici speciali che rappresentano visivamente le parti di un programma che utilizzano più tempo CPU. I grafici a fiamma facilitano l'individuazione delle aree del codice che devono essere ottimizzate.

È importante sottolineare che le prestazioni sono incluse anche nell'elenco di controllo per la raccolta dei dati per i casi OOM (Out of Memory), come consigliato dalla Business Unit (BU) dell'NSO. Per ulteriori informazioni sulla risoluzione dei problemi OOM, contattare Cisco TAC.

Risoluzione dei problemi di prestazioni relativi all'utilizzo delle prestazioni per NSO

In questa sezione viene fornito un flusso di lavoro completo per l'installazione, l'utilizzo e l'analisi dei dati dallo strumento per le prestazioni sugli host NSO per la risoluzione dei problemi relativi alle prestazioni.

Prestazioni installazione

Passaggio 1: Installare le prestazioni sulla distribuzione Linux. Utilizzare il comando appropriato per il sistema operativo:

Per Ubuntu:

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

Per Debian:

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

Per i derivati di Fedora/RedHat:

```
dnf install -y perf
```

Per ulteriori informazioni sugli avvertimenti noti durante l'installazione delle prestazioni, contattare il team Cisco TAC.

Campionamento dei dati

Passaggio 1: Identificare il processo NSO principale.

Utilizzare il comando seguente per individuare il processo NSO (ncs.smp):

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

Output di esempio:

```
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
```

Passaggio 2: In alternativa, è necessario utilizzare il PID del processo Java principale legato a NSO, soprattutto se si focalizza sulle operazioni Java. Lanciare:

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

Output di esempio:

```
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
```

Passaggio 3: Eseguire il test case o lo use case problematico per convalidare lo scenario delle prestazioni.

Passaggio 4: Su una finestra di terminale diversa, eseguire perf sui relativi ID di processo (PID). Utilizzare il formato di comando indicato di seguito, sostituendo XX,YY,ZZ con i PID ottenuti in precedenza:

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

Ad esempio, per creare un profilo a livello di sistema e raccogliere grafici delle chiamate a 99 Hz per PID specifici:

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

Output di esempio:

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

Descrizioni delle opzioni:

- -a: tutte le CPU; raccolta a livello di sistema da tutte le CPU (impostazione predefinita se non viene specificata alcuna destinazione).
- -g: Acquisisci grafici chiamate (analisi stack). Identifica dove vengono chiamate le funzioni.
- -F: Frequenza di campionamento in Hz. Frequenze più alte aumentano la precisione ma aggiungono il sovraccarico.
- -p: Specifica gli ID del processo.

Passaggio 5: Al termine della raccolta dei campioni, arrestare le prestazioni 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) ]
```

A questo punto è possibile visualizzare un file perf.data nella directory corrente.

Passaggio 6: Generare un report di riepilogo con questo comando:

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

Descrizioni delle opzioni:

- -n: Mostra i simboli senza raggruppamento (vista piatta).
- —studio: Forza l'output all'output standard (il terminale).

A questo punto, è necessario salvare entrambi i file (perf.data e perf_report.txt) e condividerli con il contatto di supporto prima di procedere con l'analisi.

Se l'acquisizione è riuscita, perf_report.txt mostra una struttura ad albero che rappresenta un grafico gerarchico delle chiamate. Le percentuali consentono di identificare gli hotspot in cui viene utilizzata la maggior parte del tempo CPU.

Esempio estratto:

```
# 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
```

Interpretazione

- Processo/Thread: È in corso l'analisi del thread C2 CompilerThree.
- Utilizzo totale CPU: Questo thread è responsabile del 30,61% del tempo CPU.
- Flusso funzione: Il thread inizia con `start_thread` e i delegati lavorano su più livelli. La maggior parte del tempo della CPU (30,47%) viene impiegato in `C2Compiler::compile_method`, indicando un potenziale hotspot.

generazione di un grafico a fiamma

Passaggio 1: Genera un campione di prestazioni da tutte le CPU e i processi in un intervallo definito (ad esempio, 60 secondi):

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

Output di esempio:

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

Passaggio 2: Copiare o trasferire il file perf.data in un host da cui è possibile scaricare il repository dei modelli di flamegraph.

Passaggio 3: Convertire il file perf.data in un formato testo:

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

Passaggio 4: Clonare il repository GitHub di FlameGraph e inserire data.perf nella directory seguente:

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

Passaggio 5: Comprimere le tracce dello stack per l'elaborazione del flamegrafo:

<#root>

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

Passaggio 6: Generate il file SVG del grafico a fiamma:

<#root>

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

Nota: Se si verifica l'errore "can not locate open.pm in @INC" (impossibile individuare open.pm in @INC) in CentOS o RHEL, installare il modulo Perl richiesto:

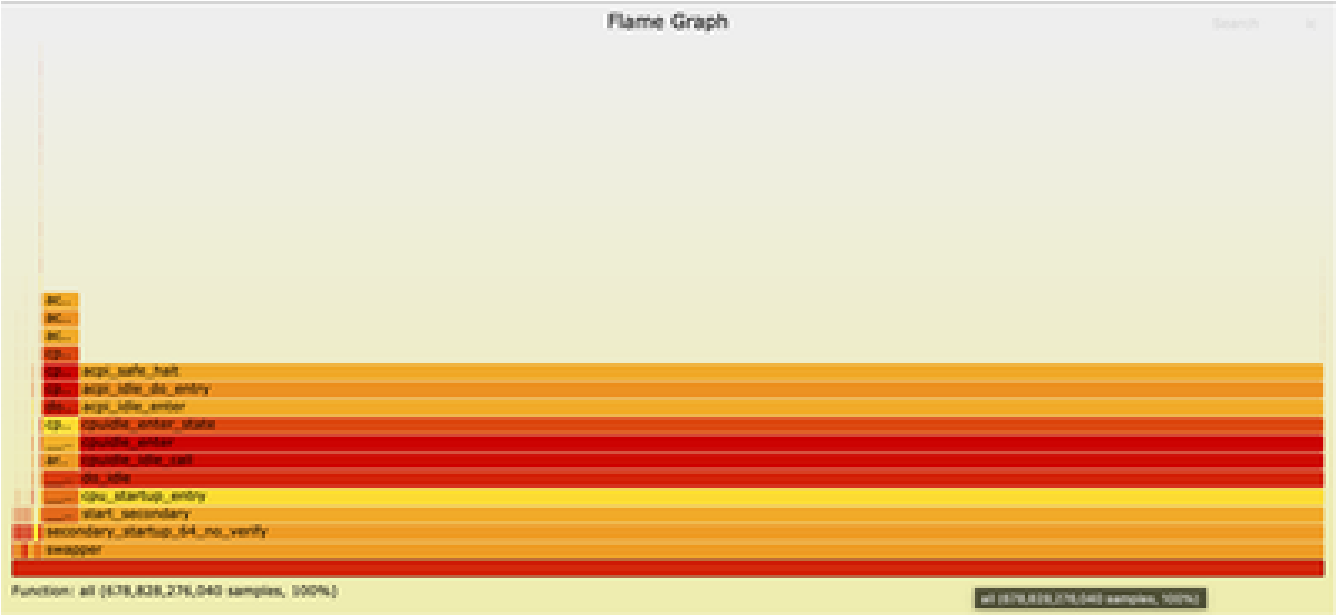
```
yum install perl-open.noarch
```

Passaggio 7: Aprite il file data.svg nel browser Web preferito per visualizzare il grafico a fiamma.

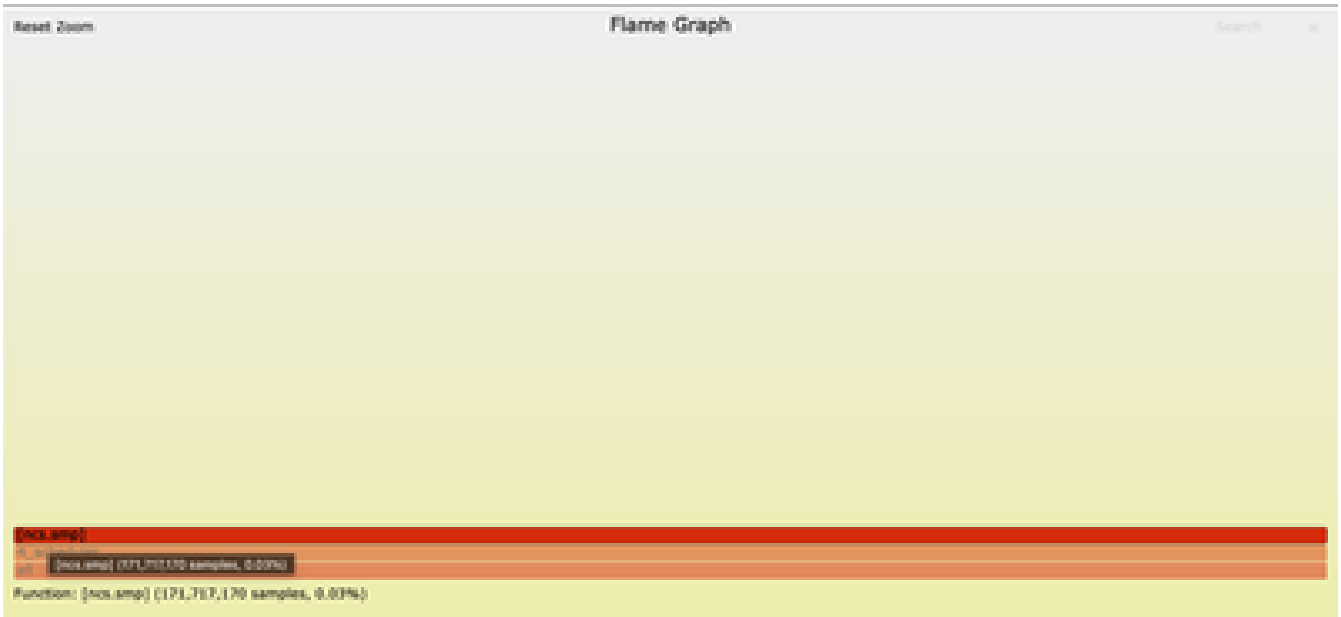
Esplorazione del grafico Fiamma

Una volta che il file del grafico a fiamma è aperto nel browser, potete interagire con esso facendo clic su qualsiasi casella per ingrandire la funzione e il suo stack di chiamate. La lunghezza di ogni

casella rappresenta la quantità di tempo CPU trascorso in quella funzione e nel relativo stack di chiamate. Questa visualizzazione facilita l'identificazione degli hotspot e delle aree da ottimizzare.



Ingrandito in ncs.smp:



Informazioni correlate

- Avvertenze note sulle prestazioni Linux
- Supporto tecnico Cisco e download

Informazioni su questa traduzione

Cisco ha tradotto questo documento utilizzando una combinazione di tecnologie automatiche e umane per offrire ai nostri utenti in tutto il mondo contenuti di supporto nella propria lingua. Si noti che anche la migliore traduzione automatica non sarà mai accurata come quella fornita da un traduttore professionista. Cisco Systems, Inc. non si assume alcuna responsabilità per l’accuratezza di queste traduzioni e consiglia di consultare sempre il documento originale in inglese (disponibile al link fornito).