

Solución de problemas de uso elevado de la CPU de la pila Java

Contenido

[Introducción](#)

[Solución de problemas con Jstack](#)

[¿Qué es Jstack?](#)

[¿Por qué necesita Jstack?](#)

[Procedimiento](#)

[¿Qué es un hilo?](#)

Introducción

Este documento describe Java Stack (Jstack) y cómo utilizarlo para determinar la causa raíz de la alta utilización de la CPU en Cisco Policy Suite (CPS).

Solución de problemas con Jstack

¿Qué es Jstack?

Jstack toma un vaciado de memoria de un proceso Java en ejecución (en CPS, QNS es un proceso Java). Jstack tiene todos los detalles de ese proceso Java, como subprocesos/aplicaciones y la funcionalidad de cada subproceso.

¿Por qué necesita Jstack?

Jstack proporciona el seguimiento de Jstack para que los ingenieros y desarrolladores puedan conocer el estado de cada subproceso.

El comando Linux utilizado para obtener el seguimiento Jstack del proceso Java es:

```
# jstack <process id of Java process>
```

La ubicación del proceso Jstack en cada versión de CPS (anteriormente conocida como Quantum Policy Suite (QPS)) es '/usr/java/jdk1.7.0_10/bin/' donde 'jdk1.7.0_10' es la versión de Java y la versión de Java puede variar en cada sistema.

También puede ingresar un comando Linux para encontrar el trayecto exacto del proceso Jstack:

```
# find / -iname jstack
```

Jstack se explica aquí para familiarizarle con los pasos para resolver problemas de uso elevado de la CPU debido al proceso Java. En los casos de uso elevado de la CPU, por lo general, aprende que un proceso Java utiliza la CPU alta del sistema.

Procedimiento

Paso 1: Ingrese el comando `top` Linux para determinar qué proceso consume una CPU alta de la máquina virtual (VM).

```
[root@pcrfclient01 ~]# top
top - 08:36:01 up 221 days, 20:52,  4 users,  load average: 5.86, 3.32, 2.60
Tasks: 1048 total,  1 running, 1037 sleeping,  0 stopped,  10 zombie
Cpu(s): 13.8%us,  4.2%sy,  0.0%ni, 80.0%id,  0.7%wa,  0.2%hi,  1.2%si,  0.0%st
Mem:   5975016k total,  5612888k used,  362128k free,   59776k buffers
Swap:  2097144k total,  1434016k used,  663128k free,   913832k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
14763	root	25	0	10.4g	1.3g	9.8m	S	5.9	23.3	5728:23	java
21534	qns	18	0	121m	71m	1460	S	1.7	1.2	6250:45	cisco
6667	apache	16	0	312m	20m	3984	S	1.3	0.3	0:15.51	httpd
929	mongod	15	0	572m	97m	71m	S	1.0	1.7	1744:19	mongod
14973	root	15	0	13428	2060	940	R	1.0	0.0	0:00.09	top
4950	apache	16	0	312m	19m	3984	S	0.3	0.3	0:09.06	httpd
11839	apache	16	0	312m	20m	3984	S	0.3	0.3	0:27.41	httpd
12819	apache	16	0	312m	20m	3984	S	0.3	0.3	0:16.89	httpd
1	root	15	0	10368	628	596	S	0.0	0.0	7:00.45	init
2	root	RT	-5	0	0	0	S	0.0	0.0	9:12.97	migration/0

A partir de este resultado, elimine los procesos que consumen más %CPU. Aquí, Java consume el 5,9%, pero puede consumir más CPU, como más del 40%, 100%, 200%, 300%, 400%, etc.

Paso 2: Si un proceso Java consume una CPU alta, ingrese uno de estos comandos para averiguar qué subproceso consume cuánta:

```
# ps -C java -L -o pcpu,cpu,nice,state,cputime,pid,tid | sort
```

```
O
```

```
# ps -C
```

A modo de ejemplo, esta visualización muestra que el proceso Java consume una CPU elevada (+40%), así como los subprocesos del proceso Java responsables de la utilización alta.

```
<snip>
```

```
0.2 - 0 S 00:17:56 28066 28692
```

```
0.2 - 0 S 00:18:12 28111 28622
```

```

0.4 - 0 S 00:25:02 28174 28641
0.4 - 0 S 00:25:23 28111 28621
0.4 - 0 S 00:25:55 28066 28691
43.9 - 0 R 1-20:24:41 28026 30930
44.2 - 0 R 1-20:41:12 28026 30927
44.4 - 0 R 1-20:57:44 28026 30916
44.7 - 0 R 1-21:14:08 28026 30915
%CPU CPU NI S TIME      PID  TID

```

¿Qué es un hilo?

Suponga que tiene una aplicación (es decir, un único proceso en ejecución) dentro del sistema. Sin embargo, para realizar muchas tareas es necesario crear muchos procesos y cada proceso crea muchos subprocesos. Algunos de los subprocesos pueden ser lector, escritor y diferentes propósitos, como la creación del registro de detalles de llamadas (CDR), etc.

En el ejemplo anterior, el ID de proceso Java (por ejemplo, 28026) tiene varios subprocesos que incluyen 30915, 30916, 30927 y muchos más.

Nota: El ID de subproceso (TID) tiene formato decimal.

Paso 3: Verifique la funcionalidad de los subprocesos Java que consumen la CPU alta.

Ingrese estos comandos Linux para obtener el seguimiento Jstack completo. El ID de proceso es el PID de Java, por ejemplo 28026 como se muestra en la salida anterior.

```
# cd /usr/java/jdk1.7.0_10/bin/
```

```
# jstack <process ID>
```

La salida del comando anterior es similar a:

```

2015-02-04 21:12:21
Full thread dump Java HotSpot(TM) 64-Bit Server VM (23.7-b01 mixed mode):

"Attach Listener" daemon prio=10 tid=0x00000000fb42000 nid=0xc8f waiting on
condition [0x0000000000000000]
java.lang.Thread.State: RUNNABLE

"ActiveMQ BrokerService[localhost] Task-4669" daemon prio=10 tid=0x00002aaab41fb800
nid=0xb24 waiting on condition [0x0000000004c9ac000]
java.lang.Thread.State: TIMED_WAITING (parking)
at sun.misc.Unsafe.park(Native Method)
- parking to wait for <0x00000000c2c07298>
(a java.util.concurrent.SynchronousQueue$TransferStack)
at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:226)
at java.util.concurrent.SynchronousQueue$TransferStack.awaitFulfill
(SynchronousQueue.java:460)
at java.util.concurrent.SynchronousQueue$TransferStack.transfer
(SynchronousQueue.java:359)
at java.util.concurrent.SynchronousQueue.poll(SynchronousQueue.java:942)
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1068)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:722)

```

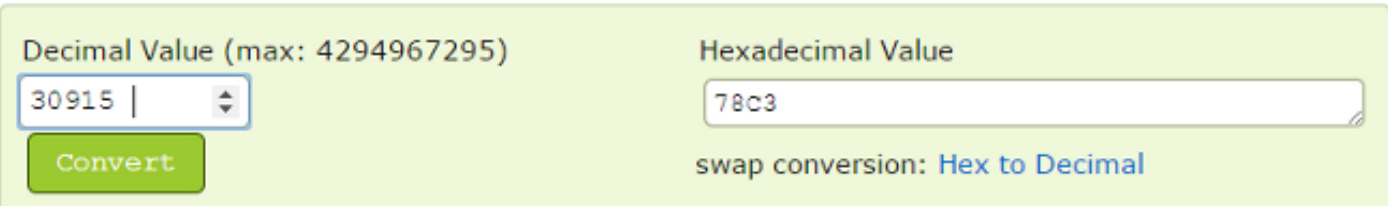
```
"ActiveMQ BrokerService[localhost] Task-4668" daemon prio=10 tid=0x00002aaab4b55800
nid=0xa0f waiting on condition [0x0000000043a1d000]
java.lang.Thread.State: TIMED_WAITING (parking)
at sun.misc.Unsafe.park(Native Method)
- parking to wait for <0x00000000c2c07298>
(a java.util.concurrent.SynchronousQueue$TransferStack)
at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:226)
at java.util.concurrent.SynchronousQueue$TransferStack.awaitFulfill
(SynchronousQueue.java:460)
at java.util.concurrent.SynchronousQueue$TransferStack.transfer
(SynchronousQueue.java:359)
at java.util.concurrent.SynchronousQueue.poll(SynchronousQueue.java:942)
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1068)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:722)
```

<snip>

```
"pool-84-thread-1" prio=10 tid=0x00002aaac45d8000 nid=0x78c3 runnable
[0x000000004c1a4000]
java.lang.Thread.State: RUNNABLE
at sun.nio.ch.IOUtil.drain(Native Method)
at sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:92)
- locked <0x00000000c53717d0> (a java.lang.Object)
at sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:87)
- locked <0x00000000c53717c0> (a sun.nio.ch.Util$2)
- locked <0x00000000c53717b0> (a java.util.Collections$UnmodifiableSet)
- locked <0x00000000c5371590> (a sun.nio.ch.EPollSelectorImpl)
at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:98)
at zmq.Signaler.wait_event(Signaler.java:135)
at zmq.Mailbox.recv(Mailbox.java:104)
at zmq.SocketBase.process_commands(SocketBase.java:793)
at zmq.SocketBase.send(SocketBase.java:635)
at org.zeromq.ZMQ$Socket.send(ZMQ.java:1205)
at org.zeromq.ZMQ$Socket.send(ZMQ.java:1196)
at com.broadhop.utilities.zmq.concurrent.MessageSender.run(MessageSender.java:146)
at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:471)
at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)
at java.util.concurrent.FutureTask.run(FutureTask.java:166)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:722)
```

Ahora debe determinar qué subproceso del proceso Java es responsable de la alta utilización de la CPU.

A modo de ejemplo, observe el TID 30915 como se menciona en el Paso 2. Debe convertir el TID en formato decimal a hexadecimal porque en el seguimiento de Jstack sólo puede encontrar el formato hexadecimal. Utilice este [convertidor](#) para convertir el formato decimal en el formato hexadecimal.



Decimal Value (max: 4294967295)	Hexadecimal Value
<input type="text" value="30915"/>	<input type="text" value="78c3"/>
<input type="button" value="Convert"/>	swap conversion: Hex to Decimal

Como puede ver en el Paso 3, la segunda mitad del seguimiento de Jstack es el subproceso que

es uno de los subprocesos responsables detrás del uso elevado de la CPU. Cuando encuentre el 78C3 (formato hexadecimal) en el seguimiento de Jstack, sólo encontrará este subproceso como 'nid=0x78c3'. Por lo tanto, puede encontrar todos los subprocesos de ese proceso Java que son responsables del alto consumo de CPU.

Nota: No es necesario centrarse en el estado del subproceso por ahora. Como punto de interés, se han visto algunos estados de los subprocesos como Ejecutar, Bloqueado, Tiempo_en_espera y Espera.

Toda la información anterior ayuda a CPS y a otros desarrolladores de tecnología a llegar a la causa raíz del problema de uso elevado de la CPU en el sistema/VM. Capture la información mencionada anteriormente en el momento en que aparezca el problema. Una vez que la utilización de la CPU vuelve a la normalidad, no se pueden determinar los subprocesos que causaron el problema de la CPU elevada.

Los registros CPS también deben capturarse. Esta es la lista de registros CPS de la VM 'PCRfclient01' bajo la trayectoria '/var/log/broadcast':

- **motor consolidado**
- **consolidadas-qns**

Además, obtenga el resultado de estos scripts y comandos de la VM PCRfclient01:

- **# diagnostics.sh** (Es posible que este script no se ejecute en versiones anteriores de CPS, como QNS 5.1 y QNS 5.2)
- **# df -kh**
- **N.º top**