Utilización del stack de las Javas del Troubleshooting CPU elevada

Contenido

Introducción

Este documento describe el stack de las Javas (Jstack) y cómo utilizarlo para determinar la causa raíz CPU elevada de la utilización en la habitación de la directiva de Cisco (CP).

Troubleshooting con Jstack

¿Cuál es Jstack?

Jstack toma un vaciado de memoria de un proceso corriente de las Javas (en los CP, QNS es un proceso de las Javas). Jstack tiene todos los detalles de eso proceso de las Javas, tal como hilos/aplicaciones y las funciones de cada hilo.

¿Por qué usted necesita Jstack?

Jstack proporciona la traza de Jstack de modo que los ingenieros y los promotores puedan familiarizarse con el estado de cada hilo.

El comando linux usado para obtener la traza de Jstack del proceso de las Javas es:

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

La ubicación del proceso de Jstack en cada CP (conocidos previamente como habitación de la directiva de Quantum (QPS)) la versión es '/usr/java/jdk1.7.0_10/bin/ donde está la versión de Java 'jdk1.7.0_10 y la versión de Java puede diferenciar en cada sistema.

Usted puede también ingresar un comando linux para encontrar el trayecto exacto del proceso de Jstack:

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

Jstack se explica aquí para conseguirle familiar con los pasos resolver problemas CPU elevada los problemas de la utilización debido al proceso de las Javas. En CPU elevada la utilización le encajona aprenden generalmente que un proceso de las Javas utiliza CPU elevada del sistema.

Procedimiento

Paso 1: Ingrese el comando linux **superior** para determinar que el proceso consume CPU elevada 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
```

De esta salida, saque los procesos que consumen más %CPU. Aquí, la Java toma el 5.9% pero puede consumir más CPU por ejemplo más el de 40%, el 100%, el 200%, el 300%, el 400%, y así sucesivamente.

Paso 2: Si un proceso de las Javas consume CPU elevada, ingrese uno de estos comandos para descubrir que el hilo consuma cuánto:

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

O

```
# ps -C <process ID> -L -o pcpu,cpu,nice,state,cputime,pid,tid | sort
```

Como un ejemplo, esta visualización muestra que el proceso de las Javas consume CPU elevada (el +40%) así como los hilos del proceso de las Javas responsable de la utilización intensa.

```
<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
```

¿Cuál es un hilo?

Suponga que usted tiene una aplicación (es decir, un solo proceso en ejecución) dentro del sistema. Sin embargo, para realizar muchas tareas usted requiere muchos procesos ser creado y cada proceso crea muchos hilos. Algunos de los hilos podrían ser lector, escritor, y diversos propósitos tales como creación del registro de detalles de la llamada (CDR) y así sucesivamente.

En el ejemplo anterior, el identificador de proceso de las Javas (por ejemplo, 28026) tiene varios subprocesos que incluyan 30915, 30916, 30927 y mucho más.

> Nota: El hilo ID (TID) está en el formato decimal.

Paso 3: Marque las funciones de los hilos de las Javas que consumen CPU elevada.

Ingrese estos comandos linux para obtener la traza completa de Jstack. El identificador de proceso es la Java PID, por ejemplo 28026 tal y como se muestra en de la salida anterior.

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

# jstack <process ID>
```

La salida del comando anterior parece:

```
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=0x000000000fb42000 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 [0x000000004c9ac000]
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)
```

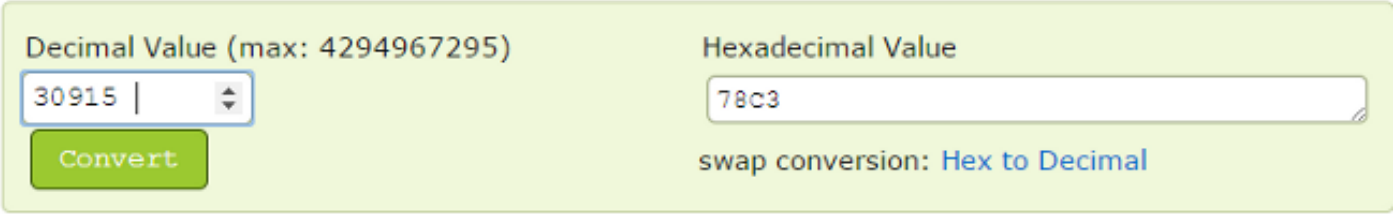
```
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1068)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:722)


<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 usted necesita determinar que el hilo del proceso de las Javas es responsable CPU elevada de la utilización.

Como un ejemplo, mirada en el TID 30915 como se menciona en el paso 2. Usted necesita convertir el TID en el decimal al formato hexadecimal porque en la traza de Jstack, usted puede encontrar solamente la forma hexadecimal. Utilice este convertidor para convertir el formato decimal en el formato hexadecimal.



Como usted puede ver en el paso 3, la segunda mitad de la traza de Jstack es el hilo que es uno de los hilos responsables detrás CPU elevada de la utilización. Cuando usted encuentra el 78C3 (formato hexadecimal) en la traza de Jstack, después usted encontrará solamente este hilo como 'nid=0x78c3. Por lo tanto, usted puede encontrar todos los hilos de eso proceso de las Javas que son responsables CPU elevada del consumo.

> Nota: Usted no necesita centrarse en el estado del hilo por ahora. Como punta del interés, algunos estados de los hilos como Runnable, bloqueados, Timed_Waiting, y esperar se han considerado.

Todas las ayudas CP de la información previa y otros desarrolladores de la tecnología le ayudan para conseguir a la causa raíz CPU elevada del problema de la utilización en el system/VM. Capture la información previamente mencionada cuando aparece el problema. Una vez que la utilización de la CPU está de nuevo al normal entonces los hilos que causaron CPU elevada el problema no pueden ser determinados.

Los registros CP necesitan ser capturados también. Aquí está la lista de registros CP del 'PCRFclient01 VM bajo la trayectoria “/var/log/broadhop”:

- consolidar-motor
- consolidado-qns

También, obtenga la salida de estos scripts y comandos del PCRFclient01 VM:

- # diagnostics.sh (este script no pudo ejecutarse en las versiones anteriores de los CP, tales como QNS 5.1 y QNS 5.2.)
- # df - KH
- # top