Dépannez l'utilisation du CPU élevé de pile de Javas

Contenu

Introduction

Ce document décrit la pile de Javas (Jstack) et comment l'employer afin de déterminer la cause principale de l'utilisation du CPU élevé dans la suite de stratégie de Cisco (CPS).

Dépannez avec Jstack

Quel est Jstack ?

Jstack prend un vidage mémoire de mémoire d'un processus courant de Javas (dans le CPS, QNS est un processus de Javas). Jstack a tous les détails de cela processus de Javas, tel que des thread/applications et la fonctionnalité de chaque thread.

Pourquoi avez-vous besoin de Jstack ?

Jstack fournit le suivi de Jstack de sorte que les ingénieurs et les développeurs puissent finir par connaître l'état de chaque thread.

La commande de Linux utilisée pour obtenir le suivi de Jstack du processus de Javas est :

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

L'emplacement du processus de Jstack dans chaque CPS (précédemment connu sous le nom de suite de stratégie de Quantum (QPS)) la version est '/usr/java/jdk1.7.0_10/bin/ où 'jdk1.7.0_10 est la version de Javas et la version de Javas peut différer dans chaque système.

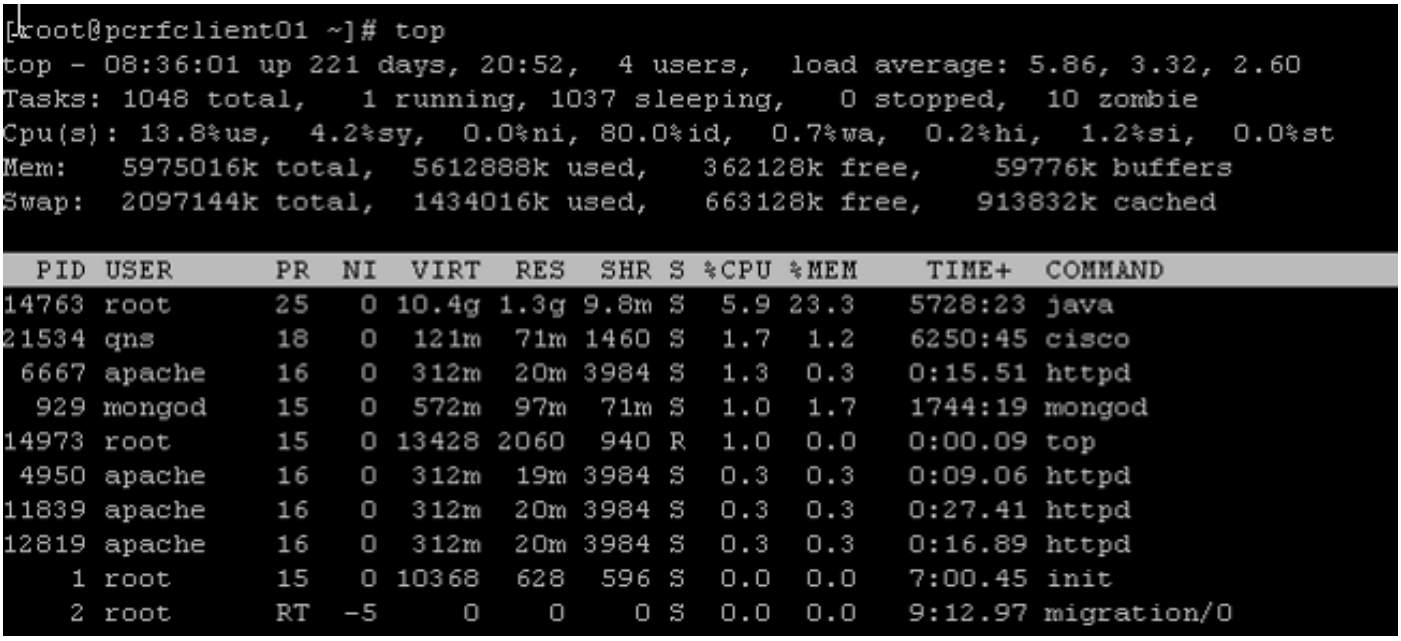
Vous pouvez également sélectionner une commande de Linux afin de trouver le chemin précis du processus de Jstack :

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

Jstack est expliqué ici afin de vous obliger au courant des étapes pour dépanner des questions d'utilisation du CPU élevé en raison du processus de Javas. Dans l'utilisation du CPU élevé vous enferme apprennent généralement qu'un processus de Javas utilise la CPU de haute du système.

Procédure

Étape 1 : Sélectionnez la commande **supérieure de** Linux afin de déterminer quel processus consomme la CPU de haute du virtual machine (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 cette sortie, sortez les processus qui consomment plus de %CPU. Ici, Java prend 5.9 % mais elle peut consommer plus de CPU comme plus de 40%, 100%, 200%, 300%, 400%, et ainsi de suite.

Étape 2 : Si un processus de Javas consomme la CPU de haute, sélectionnez une de ces commandes afin de découvrir que le thread consomme combien :

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

OU

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

Comme exemple, cet affichage prouve que le processus de Javas consomme CPU de haute (+40%) aussi bien que les thread du processus de Javas responsable de l'utilisation élevée.

```
<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
```

Quel est un thread ?

Supposez que vous avez une application (c'est-à-dire, un processus actif simple) à l'intérieur du système. Cependant, afin d'effectuer beaucoup de tâches vous exigez de beaucoup de processus d'être créés et chaque processus crée beaucoup de thread. Certains des thread pourraient être lecteur, auteur, et différents buts tels que la création de l'article mouvement d'appel (CDR) et ainsi de suite.

Dans l'exemple précédent, l'ID de processus de Javas (par exemple, 28026) a les fils multiples qui incluent 30915, 30916, 30927 et beaucoup plus.

> Remarque: L'ID de thread (TID) est dans le format décimal.

Étape 3 : Vérifiez la fonctionnalité des thread de Javas qui consomment la CPU de haute.

Sélectionnez ces commandes de Linux afin d'obtenir le suivi complet de Jstack. L'ID de processus est Java PID, par exemple 28026 suivant les indications de la sortie précédente.

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

# jstack <process ID>
```

La sortie de la commande précédente ressemble à :

```
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)
```

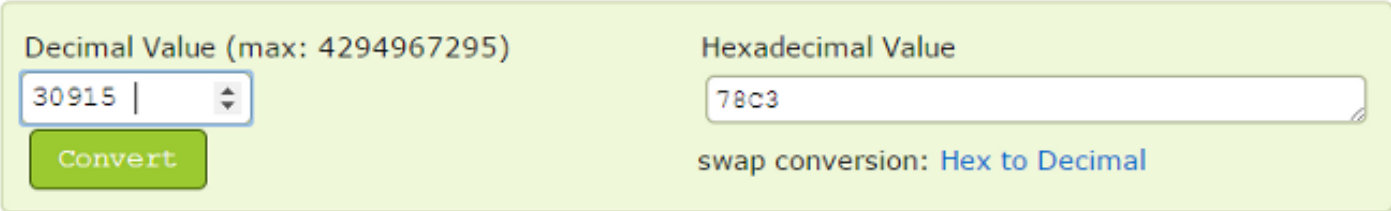
```
at java.util.concurrent.SynchronousQueue.poll(SynchronousQueue.java:942)
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1068)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
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)
```

Maintenant vous devez déterminer quel thread du processus de Javas est responsable de l'utilisation du CPU élevé.

Comme exemple, regardez TID 30915 comme mentionné dans l'étape 2. Vous devez convertir le TID dans la décimale en format hexadécimale parce que dans le suivi de Jstack, vous pouvez seulement trouver la forme hexadécimale. Utilisez ce convertisseur afin de convertir le format décimal en format hexadécimal.



Comme vous pouvez voir dans l'étape 3, la deuxième moitié du suivi de Jstack est le thread qui est l'un des thread responsables derrière l'utilisation du CPU élevé. Quand vous trouvez le 78C3 (format hexadécimal) dans le suivi de Jstack, alors vous trouverez seulement ce thread comme 'nid=0x78c3. Par conséquent, vous pouvez trouver tous les thread de cela processus de Javas qui sont responsables de la consommation élevée CPU.

> Remarque: Vous n'avez pas besoin de se concentrer sur l'état du thread pour l'instant. Comme point d'intérêt, quelques états des thread comme praticable, bloqués, Timed_Waiting, et attendre ont été vus.

Toutes les aides précédentes CPS de l'information et d'autres développeurs de technologie vous aident pour obtenir à la cause principale de la question d'utilisation du CPU élevé dans le system/VM. Saisissez les informations précédemment mentionnées lorsque le numéro apparaît. Une fois que l'utilisation du processeur est de nouveau à la normale puis les thread qui ont entraîné la question CPU de haute ne peuvent pas être déterminés.

Le besoin de logs CPS d'être aussi bien capturé. Voici la liste de logs CPS de la VM 'PCRFclient01 sous le chemin « /var/log/broadhop » :

- consolider-engine
- consolidé-qns

En outre, obtenez la sortie de ces scripts et commandes de la VM PCRFclient01 :

- # diagnostics.sh (ce script ne pourrait pas fonctionner sur des versions plus anciennes de CPS, telles que QNS 5.1 et QNS 5.2.)
- # DF - le KH
- # dessus