استكشاف أخطاء استخدام Java High CPU وإصلاحها

المحتويات

المقدمة

يصف هذا المستند Jstack (Java Stack) وكيفية استخدامه لتحديد السبب الجذري لاستخدام CPU العالي في Cisco Policy Suite (CPS).

أستكشاف الأخطاء وإصلاحها باستخدام JSTACK

ما هو JSTACK؟

يأخذ JSTACK تفريغ ذاكرة لعملية Java قيد التشغيل (في CPS، QNS هي عملية Java). يحتوي JSTACK على كافة تفاصيل عملية Java، مثل مؤشرات الترابط/التطبيقات ووظائف كل مؤشر ترابط.

لماذا تحتاج إلى Jstack؟

يوفر JSTACK تتبع JSTACK بحيث يمكن للمهندسين والمطورين معرفة حالة كل مؤشر ترابط.

أمر Linux المستخدم للحصول على تتبع Jstack لعملية Java هو:

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

موقع عملية Jstack في كل CPS (المعروفة سابقا ب QPS) إصدار '/usr/java/jdk1.7.0_10/bin/' حيث 'jdk1.7.0_10' هو إصدار Java ويمكن أن يختلف إصدار Java في كل نظام.

يمكنك أيضا إدخال أمر Linux لإيجاد المسار الدقيق لعملية Jstack:

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

يتم شرح JSTACK هنا لإطلاعك على الخطوات الخاصة باستكشاف أخطاء إستخدام وحدة المعالجة المركزية

(CPU) العالية وإصلاحها بسبب عملية Java. في حالات إستخدام وحدة المعالجة المركزية (CPU) بشكل عام، تعلم أن عملية Java تستخدم وحدة المعالجة المركزية (CPU) العالية من النظام.

الإجراء

الخطوة 1: أدخل أمر Linux **Top** لتحديد العملية التي تستهلك وحدة المعالجة المركزية (CPU) العالية من الجهاز الظاهري (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
```

من هذا الإخراج، قم بإخراج العمليات التي تستهلك أكثر من CPU٪. هنا، Java تأخذ 5.9٪ ولكن يمكنها أن تستهلك أكثر من وحدة المعالجة المركزية مثل أكثر من 40٪، 100٪، 200٪، 300٪، 400٪، وهكذا.

الخطوة 2: إذا استهلكت عملية Java وحدة معالجة مركزية (CPU) عالية، فأدخل أحد هذه الأوامر لمعرفة أي خيط يستهلك مقدار:

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

أو

```
# ps -C
```

على سبيل المثال، يوضح هذا العرض أن عملية Java تستهلك وحدة معالجة مركزية (CPU) عالية (+40٪) بالإضافة إلى الخيوط الخاصة بعملية Java المسؤولة عن الاستخدام العالي.

```
<snip>

   S 00:17:56 28066 28692 0 - 0.2
   S 00:18:12 28111 28622 0 - 0.2
   S 00:25:02 28174 28641 0 - 0.4
   S 00:25:23 28111 28621 0 - 0.4
   S 00:25:55 28066 28691 0 - 0.4
R 1-20:24:41 28026 30930 0 - 43.9
```

```
R 1-20:41:12 28026 30927 0 - 44.2
R 1-20:57:44 28026 30916 0 - 44.4
R 1-21:14:08 28026 30915 0 - 44.7
  CPU CPU NI S TIME     PID  TID%
```

ما هو الخيط؟

افترض أن لديك تطبيقا (أي عملية واحدة قيد التشغيل) داخل النظام. ومع ذلك، يتطلب تنفيذ العديد من المهام إنشاء العديد من العمليات، كما تنشئ كل عملية العديد من مؤشرات الترابط. يمكن أن تكون بعض مؤشرات الترابط قارئ وكاتب وأغراض مختلفة مثل إنشاء سجل تفاصيل المكالمة (CDR) وما إلى ذلك.

في المثال السابق، يحتوي معرف عملية Java (على سبيل المثال، 28026) على مؤشرات ترابط متعددة تتضمن 30915 و 30916 و 30927 وغير ذلك الكثير.

ملاحظة: معرف مؤشر الترابط (TID) بتنسيق عشري.

الخطوة 3: تحقق من وظائف مؤشرات ترابط Java التي تستهلك وحدة المعالجة المركزية (CPU) العالية.

أدخل أوامر Linux هذه للحصول على تتبع Jstack الكامل. معرف العملية هو معرف Java PID، على سبيل المثال 28026 كما هو موضح في الإخراج السابق.

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

# jstack <process ID>
```

يبدو إخراج الأمر السابق كما يلي:

```
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)
```

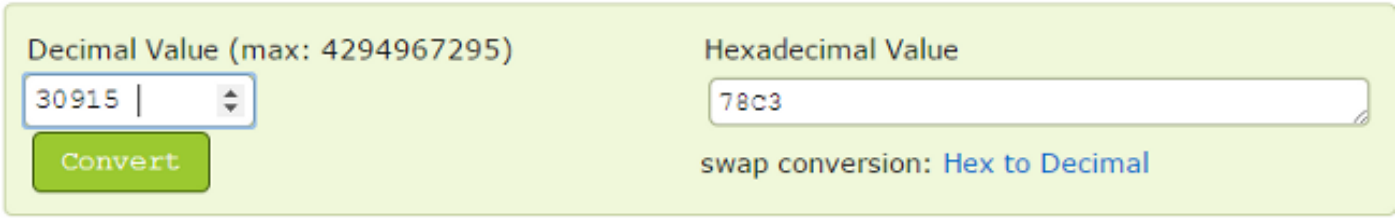
```
(at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:226
      at java.util.concurrent.SynchronousQueue$TransferStack.awaitFulfill
                                                  (SynchronousQueue.java:460)
          at java.util.concurrent.SynchronousQueue$TransferStack.transfer
                                                  (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
```

والآن، يتعين عليك تحديد مؤشر ترابط عملية Java المسؤول عن إستخدام وحدة المعالجة المركزية (CPU) العالي.

كمثال، أنظروا إلى TID 30915 كما هو مذكور في الخطوة 2. تحتاج لتحويل TID بتنسيق عشري إلى تنسيق سداسي عشر لأنه في تتبع Jstack، يمكنك العثور على النموذج السداسي عشر فقط. أستخدم هذا المحول لتحويل التنسيق العشري إلى تنسيق سداسي عشر.



كما ترى في الخطوة 3، يمثل النصف الثاني من تتبع Jstack مسار التنفيذ الذي يعد أحد مؤشرات الترابط المسؤولة وراء الاستخدام العالي لوحدة المعالجة المركزية (CPU). عندما تجد 78C3 (التنسيق السداسي العشري) في تتبع Jstack، فلن تجد إلا مؤشر الترابط هذا ك 'nid=0x78c3'. وبالتالي، يمكنك العثور على كافة مؤشرات الترابط الخاصة بعملية Java هذه المسؤولة عن إستهلاك وحدة المعالجة المركزية (CPU) المرتفع.

ملاحظة: لا تحتاج إلى التركيز على حالة مسار التنفيذ في الوقت الحالي. كنقطة اهتمام، تم مشاهدة بعض حالات مؤشرات الترابط مثل Runnable و Blocked و Timing_Waiting و Waiting.

وتساعدك كافة المعلومات السابقة على الوصول إلى السبب الرئيسي لمشكلة إستخدام وحدة المعالجة المركزية (CPU) في النظام/الجهاز الافتراضي. التقط المعلومات المذكورة سابقا وقت ظهور المشكلة. بمجرد عودة إستخدام وحدة المعالجة المركزية (CPU) إلى الوضع الطبيعي، لا يمكن تحديد مؤشرات الترابط التي تسببت في مشكلة وحدة المعالجة المركزية (CPU) العالية.

يجب التقاط سجلات CPS كذلك. فيما يلي قائمة سجلات CPS من "PCRFclient01" VM أسفل المسار '/var/log/broadhop':

- **محرك مدمج**
- **جودة الخدمة (QNS) المدمجة**

احصل أيضا على إخراج هذه البرامج النصية والأوامر من PCRFclient01 VM:

- **# diagnostics.sh** (قد لا يعمل هذا البرنامج النصي على الإصدارات الأقدم من CPS، مثل QNS 5.1 و QNS 5.2.)
- **# df -كيلو**
- **# أعلى**

حول هذه الترجمة
ترجمت Cisco هذا المستند باستخدام مجموعة من التقنيات الآلية والبشرية لتقديم محتوى دعم للمستخدمين في جميع أنحاء العالم بلغتهم الخاصة. يُرجى ملاحظة أن أفضل ترجمة آلية لن تكون دقيقة كما هو الحال مع الترجمة الاحترافية التي يقدمها مترجم محترف. تخلي Cisco Systems مسئوليتها عن دقة هذه الترجمات وتُوصي بالرجوع دائمًا إلى المستند الإنجليزي الأصلي (الرابط متوفر).