Java スタック CPU 高使用率のトラブルシューティング

目次

概要

この資料は Java スタック（Jstack）をおよび Cisco Policy Suite （CPS）の CPU使用率が高い状態の根本的な原因を判別するためにそれを使用する方法を記述したものです。

Jstack で解決して下さい

Jstack とは何か。

Jstack は動作 Java プロセスの記憶 ダンプを奪取 します（CPS で、QNS は Java プロセスです）。 Jstack にそれのすべての詳細が各スレッドのスレッド/アプリケーションおよび機能性のような Java プロセス、あります。

Jstack を必要とする理由

Jstack はエンジニアおよび開発者が各スレッドの状態を知ることを得ることができるように Jstack トレースを提供します。

Java プロセスの Jstack トレースを得るのに使用される Linux コマンドは次のとおりです:

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

各 CPS の Jstack プロセスの位置（以前にとして Quantum Policy Suite （QPS 知られている））バージョンは 'jdk1.7.0_10 が Javaのバージョンの Javaのバージョンが各システムで異なることができる '/usr/java/jdk1.7.0_10/bin/ であり。

また Jstack プロセスの正確 な パスを見つけ出すために Linux コマンドを入力できます:

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

Jstack はステップを詳し Java プロセスが理由で CPU使用率が高い状態問題を解決するために得るためにここに説明されます。 CPU使用率が高い状態で Java プロセスがシステムからの高CPU を利用することを一般に学びます包装します。

手順

ステップ 1： どのプロセスが Virtual Machine （ VM ）からの高CPU を消費するか判別するために上 Linux コマンドを入力して下さい。

```
[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
```

この出力から、より多くの %CPU を消費するプロセスを奪取 して下さい。 ここでは、Java は 5.9% を奪取 します 40% 以上、100%、200%、300%、400%、等より多くの CPU をのような消費する場合があります。

ステップ 2： Java プロセスが高CPU を消費する場合、調べるためにどの位かスレッドが消費するこれらのコマンドの 1 つを入力して下さい:

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

または

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

一例として、このディスプレイは Java プロセスが高CPU （ +40% ）を消費することを示します、また Java のスレッドは高い稼働率に責任がある処理します。

```
<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
```

スレッドとは何か。

システムの中のアプリケーション（すなわち、単一実行中のプロセス）があることを仮定して下さい。 ただし、多くのタスクを行うために多くのプロセスが作成されるように要求し、各プロセスは多くのスレッドを作成します。 いくつかのスレッドはコール詳細レコード（CDR）作成のような読者、ライターおよび等異なる目的である可能性があります。

前例では、Java プロセス ID （たとえば、28026） 30915、30916、30927 および多くを含む複数のスレッドを持っています。

> **注:** スレッド ID （TID）は 10進法式にあります。

ステップ 3： 高CPU を消費する Java スレッドの機能性をチェックして下さい。

Jstack 完全なトレースを得るためにこれらの Linux コマンドを入力して下さい。 プロセス ID は前の出力に示すように Java PID、たとえば 28026 です。

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

# jstack <process ID>
```

前のコマンド外観の出力のような:

```
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)
```

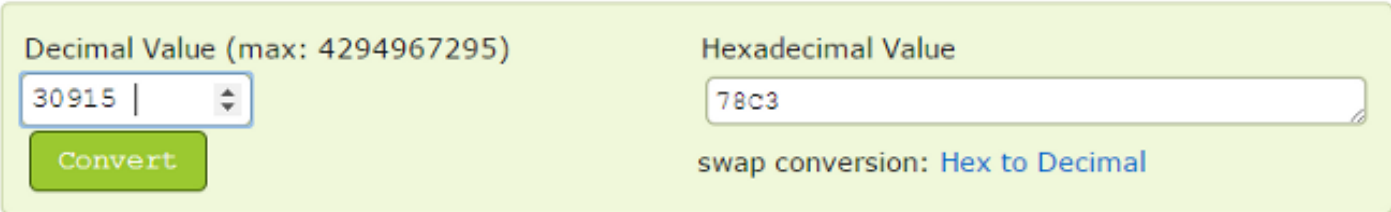
```
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1068)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:722)


<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)
```

この場合 Java プロセスのどのスレッドが CPU使用率が高い状態に責任があるか判別する必要があります。

一例として、ステップ 2.に言及されているように TID 30915 の外観。 Jstack トレースしか、16進法形式を見つけることができないので 16進フォーマットに小数点の TID を変換する必要があります。 16進フォーマットに 10進法式を変換するためにこのコンバータを使用して下さい。



ステップ 3 を見てわかるように Jstack トレースの後半は CPU使用率が高い状態の後ろの責任があるスレッドの 1 つのスレッドです。 Jstack トレースの 78C3 （16進フォーマット）を見つける場合、'nid=0x78c3 としてだけこのスレッドを見つけます。 それ故に、Java プロセス 高CPU 消費に責任があるそれのすべてのスレッドを見つけることができます。

> **注:** スレッドの状態に今のところ焦点を合わせる必要はありません。 対象のポイントとして、Timed_Waiting ブロックされる、Runnable および待っていることのようなスレッドのいくつかの状態は参照されました。

すべての前の情報ヘルプ CPS および他のテクノロジー開発者は system/VM の CPU使用率が高い状態問題の根本的な原因に到達するために助けます。 問題が現われる時以前に述べられた情報を

キャプチャ して下さい。 CPU稼働率が標準に戻ってそしてあれば高CPU 問題を引き起こしたスレッドは判別することができません。

CPS ログは同様にキャプチャ される必要があります。 'PCRFclient01 VM からの CPS ログのリストはパス「/var/log/broadhop の下に」ここにあります:

- 強化エンジン
- 強化されるqns

また、PCRFclient01 VM からのこれらのスクリプトおよびコマンドの出力を得て下さい:

- # diagnostics.sh （ このスクリプトは QNS 5.1 および QNS 5.2 のような CPS のより古いバージョンで、動作しないかもしれません。 ）
- # df - kh
- #上