Monday, June 14. 2010

### Practical DTracing

There is a simple reason why DTrace is such a great tool. I did the following steps while a production system ran at full speed at approx. 11:45 ... at the highest load of the day (the storm right before before lunch).The situationLet's assume you come to a customer and find such a situation.

```
 CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  1    2   0  471   563    6 1549  562  277   77    0  8377   94   6   0   0
  2    3   0  397   570    6 1417  591  294   67    0  9141   94   6   0   0
  3    3   0  315   496    6 1681  554  313   60    0  8794   94   6   0   0
  4    2   0  451   704    6 1915  706  314   76    0 10715   93   7   0   0
  5    3   0  464   563    5 1760  678  312  195    0  7451   94   6   0   0
  6    5   0  352   494    5 1585  533  290   45    0  8357   94   6   0   0
  7    7   0  392   570    5 1633  623  312   40    0 10005   94   6   0   0
```

Your first thought is ... "Damned, 0 percent idle and why is there such a high load of system calls".

The situation: In this system the customer started 7 JVMs. The application running in each JVM triggers a lot of threads to handle the user load. The application is called StrangeApp.

I have to apologize here, because my notes aren't complete or not usable for public usage .... so i've made up some numbers and changed names. But it should be sufficient to describe the path i used to find this problem.

Using DTraceDTrace is a tool to answer questions, not a magic bullet that finds questions and answers them. So i will write this article by asking questions.

First questionOkay ... first question "Dear DTrace, how many system calls  of which kind are made by processes called StrangeApp?"

And DTrance answered:

```
root@server:~ # dtrace -n \'syscall:::entry/execname == \"StrangeApp\"/{@[execname,probefunc] = count()}\'
dtrace: description \'syscall:::entry\' matched 234 probes
^C

[...]
  StrangeApp      yield    121980
```

Hmmm, yield() calls galore.

Second questionOkay, annother question "Dear DTrace, what calls yield() that often in the Application?". The answer of DTrace wasn't read directly ... i knew it would be longer thus i wrote it directly into a file:

```
root@server:~ dtrace -n 'syscall::yield:entry {@[ustack()] = count()}' > dtracefile
dtrace: description 'syscall::yield:entry ' matched 1 probes
root@server:~ ^C
```

The output of that file is a little bit garbled. I just piped it through c++filt:

```
root@server:~ cat dtracefile | c++filt
[...]
              libc.so.1`lwp_yield+0x4
              libjvm.so`bool ParallelTaskTerminator::offer_termination(TerminatorT erminator*)+0x90
              libjvm.so`void ParEvacuateFollowersClosure::do_void()+0x9c8
              libjvm.so`void ParNewGenTask::work(int)+0x148
              libjvm.so`void GangWorker::loop()+0x84
              libjvm.so`java_start+0x22c
              libc.so.1`_lwp_start
            24859

              libc.so.1`lwp_yield+0x4
              libjvm.so`bool ParallelTaskTerminator::offer_termination(TerminatorT erminator*)+0x90
```

```
              libjvm.so`void ParEvacuateFollowersClosure::do_void()+0x9c8
              libjvm.so`void ParNewRefProcTaskProxy::work(int)+0xf0
              libjvm.so`void GangWorker::loop()+0x84
              libjvm.so`java_start+0x22c
              libc.so.1`_lwp_start
            30074
```

Okay ... Java, it's in the libjvm.so. At this moment i knew what the problem was.Or at least the alarm was ringing as loud as the signal horn at a fire engine...jmoekamp@hivemind:/datapool/bibliothek/sources/jdk6/hotspot# grep -R "ParallelTaskTerminator::offer_termination" *
src/share/vm/utilities/taskqueue.cpp:ParallelTaskTerminator::offer_termination(TerminatorTerminator* terminator) {
src/share/vm/utilities/taskqueue.cpp:      gclog_or_tty->print_cr("ParallelTaskTerminator::offer_termination() "Let's have a look into the source:

```
114       if (hard_spin_count > WorkStealingSpinToYieldRatio) {
115         yield();
116         hard_spin_count = 0;
117         hard_spin_limit = hard_spin_start;
118 #ifdef TRACESPINNING
119         _total_yields++;
120 #endif
121       } else {
122         // Hard spin this time
123         // Increase the hard spinning period but only up to a limit.
124         hard_spin_limit = MIN2(2*hard_spin_limit,
125                                (uint) WorkStealingHardSpins);
126         for (uint j = 0; j < hard_spin_limit; j++) {
127           SpinPause();
128         }
```

Okay ... no wonder that there are so many yield calls .... let's go up the stack and let's find out what is calling this function in the great way of things. I've opted to find out about about "ParNewGenTask" albeit i did knew at that moment that i was a part of the Garbage Collector (Parallel New Generation is really obvious when you know a thing or two about Garbage Collection).

A grep for ParNewGenTask confirmed that ...
jmoekamp@hivemind:/datapool/bibliothek/sources/jdk6/hotspot# grep -R "ParNewGenTask::work" *
src/share/vm/gc_implementation/parNew/parNewGeneration.cpp:void ParNewGenTask::work(int i) {
/gc_implementation ... the place where the wild things are ...

Third questionI had another question for DTrace ... "Dear DTrace, what threads are calling yield() that often". I wanted to know if this yield calls are just made by certain threads or if they were made by all threads doing just their work.

```
root@server:~ dtrace -n 'syscall::yield:entry {@[pid, tid] = count()}'
dtrace: description 'syscall::yield:entry ' matched 1 probe
[...]
14344  4  25000
14344  3          26000
14344  7  27000
14344  5          28000
14344  6  29000
14344  2          30000
14344  8  31000
[...]
^C
```

Ask GoogleJust seven threads and this pattern repeated for each JVM, otherwise there were almost no other yield calls. The yield() was really just triggered by certain threads and the number of threads was equal to the number of CPUs. That is enough information to feed Google with it, and a short search yielded the solution. The first hit when searching for "garbage collection threads cpu" yielded :
"By default on a host with N CPUs, the throughput collector uses N garbage collector threads in the minor collection."Okay ... 7 JVMs are starting 7 garbage collection threads each ... 49 garbage collection threads. And that led me to write "About Java, parallel garbage collection and processor sets"

Posted by Joerg Moellenkamp in English, Oracle, Solaris at 07:46

Interesting read.

Keep up the good work.

Glen

Anonymous on Jun 15 2010, 17:03