Profiling Java in production - Twitter - 20131112

Title Info
Profiling Java in production, Kaushik Srenevasan, Tech Lead – Diagnostics, Twitter
(2yrs - before at Microsoft working on Javascript engine in Internet Explorer)

Introduction
JVM instrumentation and profiling
Twitter tools and ‘vm’
Twitter University youtube channel in a couple of weeks
To speak – sfjava.org

Stack mostly java components
Services < finagle < netty < jvm, native code (eg. memcache …) and Ruby (getting smaller by the day) < linux
Jvm a big deal – netty, finagle and services run on it

Tool support
Excellent tools, but “in production”
Lightweight tools, already on the box needed – can’t install eclipse, …. On prod

What want
Tools have to be global, performant, and independent – want to debug runtime and lang on it – java, scala, …
Independent – don’t want it to be coupled to the runtime, or the os – if restart then problem may go away – need to attach to what is already running

Guiding principle
Don’t want to re-invent the wheel – extend current tools? Gdb, perf, … - perf was what built on
Twitter owns whole stack – so can do more – willing to break api if will get a big performance boost – but cautious about this, don’t want to break interfaces unless really have to

State of the art
Detrace? A kernel and user mode instrumentation – dtrace not on linux, so settled on perf – so built on to of perf

**Hotspot**

Jvmti/jvmp

dTrace (systemtap)

in memory metadata, data structure layout database

dtrace/systemtap – jvm recogs systemtap and hooks into

hotspot metadata very rich – the jdb debugger is built on top of this metadata database – a database used by hotspot as run

**Trying it out**

Built a version of jdb which was able to show native frames in hotspot and the java frames – the java calls – what compiling, what currently executing …

Oracle don’t advertise this a lot, but there is this extensive database of info ready to be used

**Perf**

Built with the kernel and ships in the kernel tree, even usermode components – an abstraction over the cpu’s program counters … - may have to install in its own package

Perf gives an abstraction over program counters and helps you make sense of these and tie into your own code – tell perf how often sample, and when reach a count say what was doing/needed

Critical to make sense of what has elapsed – know own code, want to know what happening in your code when the cpu went off

**Why perf**


Test – perf was pretty close to benchmark without attach. Yourkit in sampling mode was pretty good to.

Heapster and yourkit (tracing mode)

Heapster – a twitter tool – replaces call sites in bytecode to report allocations in http port and get pprof – pprof = twitter profiler tools – bytecode instrumentation v costly overhead

The overhead of perf and yourkit (in sampling mode) both use sampling – stop at a freq and reports what finds (eg. 1000Hz)

Heapster doing bytecode instrumentation – at every point making a call to a heapster function that reports object type allocating, and what the object was, size of it, …

Heapster needs to do jni calls to figure out what doing and crosses that boundary … bytecode instrumentation – but very rich instrumentation, v clear – prolog and epilog hooks to figure out what was going on

One of the lowest cost tools to profile – the cpu counters not useful in itself, so need to associate events in your program with what going on – that is what perf gives you –

**Hotspot and perf**

How to get talk to each other –

Stack walking

Symbol table

Perf essentially stops app from time to time and gathers the info you say want – call stacks, arguments passed to function on top of stack, … perf is that powerful!

Can see what was on top of stack most time …

Does stack walking from kernel mode, and goes where it needs to – assumes all frame pointer – frame pointer a linked list on the call stack – a few instructions on the top and bottom of every function on the stack being executed – breadcrumbs if you will – and these form a linked list – x86 has a register for the frame at the top of the stack – walk the stack with these frame pointers to tell you the call chain

Perf uses this info to show what executing on the thread it’s looking at
SO: hotspot needs to set those frames up so that perf can read them – C2, opto compiler doesn’t use frame pointers – uses the fp register as a regular
register – BUT throwing away the only info on the stack that will tie to make sense of what doing. As a compiler guy thinks that compiler has root in x86,
and x86 a register starved processor

Java always had its own datastructures for frame layout and traversal …

Chose to change hotspot to use frame pointers – changed hotspot and ran traffic – and find find that doesn’t matter if stop using the framepointer – at least
for twitter workloads

Engineering-wise the better decision was to make the change hotspot – set the optimizing compiler to preserve frame pointers – twitter are maintainers of
hotspot – changed prolog emission code and set so not use rvp register – made the default, didn’t add a flag, just use it that way

Ok, so how do you make java names appear instead of hex code

Turns out easy – some others were adding jit support to perf – let jit compiler writers print symbols – essentially wrote to a file in /tmp and have perf pick
that up automatically

Demo: perf in action

A private version – a fork of the open jdk – sync with upstream sources once a month – and deploy this on twitter servers – ship things need in the jdk so
efficient – eg. the perf agent java –agentlib:perfagent –Xcomp(compile everything)

Perf –G will let you expand – call_stub one of hotspot internals whose symbols are also exported

Sudo perf record -g -a java ….

Written to a perf record file what can ship off and look at elsewhere – if have symbols will go all the way to the disassemblies, … if have elf symbols perf
will go down to the line level, but haven’t built that in yet for java as working on other features

Will opensource in due course and publish on github – and do plan to talk to oracle and see if can push upstream – would be interesting to hear oracles
own data on performance running on amd64

Stack – scala, into finagle, into netty, …

Tracing

Richer, context specific data

Unified event bus – java and native and …

Lock contention and gc usually the bottleneck, cpu not usually the constraint although solved some issues with cpu profiling

The next thing to do is tracing – but perf can be tuned to pagefaults, … not just time (cpu)

Want to know what makes the vm behave in a certain way

What’s going on the in the gc, …

Want a global tracing and profiling infrastructure – want a unified event bus – and perf already exports one such thing.

Need to tell perf that when a funct executes = gc starting, when this, gc stops, ... ie teach perf more about the jvm – method tracing, …

Tony Printezis here, wrote most of the gc – and he laments that everyone blames the gc

Not the gc?

Can tell perf about interesting things – eg object allocations ….

Dtrace entries like from perf – see gc, whether full or not (a bool) method entry, exit, object tenure event, object alloc

Dtrace probes in hotspot – need extended dtrace probes and alloc probs enabled in hotspot - -XX:+ExtendedDTraceProbes …

Perf, sample object allocation when get one and profile on all cpus

Perf exports an interface scripting option – can write in python or perl

Strace but for java

Method entry and return can be done to!!
This bring a level of programmability to profiling – can choose and do what want – an api that is suddenly slow? Then entry when that method and exit -> stop and get stack …

**Improving dtrace hooks**

These have a cost – and are working to get this more efficient

**Plan to release**

Twitter jdk – want to get it all turnkey before publish it to their github