Cross Platform Java Analysis with DTrace

Jason Brazile and Stefan Tramm
Netcetera
240

Contents

Part I: The what/why/how of DTrace

Part II: Cross Platform Java Analysis

Contents

Part I: The what/why/how of DTrace

DTrace: What and Why?

- **What**?
  - It is an awkable `truss(1)` for more than just system calls
- **Why**?
  - JMX can only monitor what someone wrote an MBean for
  - `truss(1)` can only snapshot system calls/signals/faults
  - `vmstat(1)` can only look at the vm subsystem
  - `iostat(1)` can only look at the io subsystem
  - `pstack(1)` can only look at a stack

Dtrace does these and more… and is scriptable…

DTrace: History

- 1996: conceived of by Bryan Cantrill while an undergrad at Brown
- 2001: Started work on Dtrace wth Michael W. Shapiro
- 2002: Early prototype, Adam H. Leventhal joined development
- 2004: Appears in Solaris
- 2005: Sun releases DTrace source code, initial java support (dvm)
- 2006: Ported to FreeBSD by John Birrell
- 2007: To be released in MacOSX 10.5 “Leopard” and basis of Xray

DTrace: Design features

- Provides (>30k) instrumentation points…
  - …in the kernel
  - …in the C runtime library (libc.so)
  - …in the Java VM (libjvm.so)
  - …in Java itself (starting with Java 1.6)
- Safe (probes not allowed to crash system)
- Extensible (users can implement their own instrumentation points)
- Predicates avoid retaining (copying, and storing) unneeded data
- Provides scalable aggregation (sum, min, avg, quantize, count, etc.)
- High level awk-like language, called D

Motivation: Example one-liners

- **System call count, by process**
  - dtrace -n 'syscall:::entry { @num[pid,execname] = count(); }'
- **Files opened by process**
  - dtrace -n 'syscall::open*:entry { printf("%s %s",execname,copyinstr(arg0)); }'
- **Number of bytes read, by process**
  - dtrace -n 'sysinfo:::readch { @bytes[execname] = sum(arg0); }'
- **Write size distribution, by process**
  - dtrace -n 'sysinfo:::writech { @dist[execname] = quantize(arg0); }'
- **Sample java stack at 1001 Hertz (aka profile)**
  - dtrace -n 'profile-1001 /pid == $target/ { @num[jstack()] = count(); }' -p PID

DTrace instrumentation in Java SE (>= 1.6)

> Probes enabled by default (no performance impact)
  - Garbage collection
  - Method compilation
  - Thread lifecycle
  - Class loading

> Probes disabled by default
  - Object allocation `java -XX:+DTraceAllocProbes`
  - Method invocation `java -XX:+DTraceMethodProbes`
  - Java monitor events `java -XX:+DTraceMonitorProbes`

Can also be enabled dynamically with jinfo e.g.

```
jinfo –flag +ExtendedDTraceProbes <pid>
```

Two (Stolen) Examples

- **Problem: Excessive garbage collection**
  - What are largest (aggregate) object allocations?
  - What are the call stacks when a particular allocation type occurs?
  - Strategy: use **`object-alloc`** probe + sum/count
  - **Solution: 2 one-liners**
- **Problem: Contended monitor**
  - List (quantized) time between requested entrance and actual entrance
  - Strategy: use **`monitor`** probe + **`timestamp`** and **`quantize`**
  - **Solution: a 10-liner** using **`monitor`** + quantize

Examples: Jarod Jenson, DTrace and Java: Exposing Performance Problems That Once Were Hidden

DTrace Java Example: Excessive GC (I)

> Show object allocations and total size of those allocations

```
$ dtrace -n hotspot116977:::object-alloc'{@[copyinstr(arg1, arg2)] = sum(arg3)},
dtrace: description 'hotspot116977:::object-alloc' matched 1 probe
^C
[…]
java/awt/geom/LineIterator                19608
java/util/HashMap$Entry                   21936
java/awt/geom/Point2D$Double              26160
sun/java2d/pipe/Region                    39072
java/awt/geom/Path2D$Double               94368
java/awt/geom/Rectangle2D$Double          106720
sun/java2d/SunGraphics2D                  112200
[B                                        154624
java/awt/Rectangle                        166656
[F                                        216824
java/awt/geom/RectIterator                242928
java/awt/geom/AffineTransform             500224
[I                                        1609512
[D                                        2207120
```

>1MB

Examples: Jarod Jenson, DTrace and Java: Exposing Performance Problems That Once Were Hidden

DTrace Java Example: Excessive GC (II)

> Dump the stack (20 frames) whenever an Integer array is allocated

```
$ dtrace -n hotspot116977:::object-alloc'/copyinstr(arg1, arg2) == "[I"/{@[jstack(20,2048)] = count()}'
dtrace: description 'hotspot116977:::object-alloc' matched 1 probe
^C

libjvm.so`__1cNSharedRuntimeYdtrace_object_alloc_base6FpnGThread_pnHoopDesc__i_+0x7d
libjvm.so`__1cNSharedRuntimeTdtrace_object_alloc6FpnHoopDesc__i_+0x4f
libjvm.so`__1cNCollectedHeapbCpost_allocation_setup_common6FnLKlassHandle_pnIHeapWord_I_v_+0x121
libjvm.so`__1cOtypeArrayKlassIallocate6MipnGThread__pnQtypeArrayOopDesc__+0x19b
libjvm.so`__1cKoopFactoryNnew_typeArray6FnJBasicType_ipnGThread__pnQtypeArrayOopDesc__+0x2f
libjvm.so`__1cSInterpreterRuntimeInewarray6FpnKJavaThread_nJBasicType_i_v_+0x33
sun/java2d/pipe/DuctusShapeRenderer.renderPath(Lsun/java2d/SunGraphics2D;Ljava/awt/Shape;Ljava/awt/BasicStroke;)V
sun/java2d/pipe/DuctusShapeRenderer.fill(Lsun/java2d/SunGraphics2D;Ljava/awt/Shape;)V
sun/java2d/pipe/PixelToShapeConverter.fillRect(Lsun/java2d/SunGraphics2D;IIII)V sun/java2d/SunGraphics2D.fillRect(IIII)V
sun/java2d/SunGraphics2D.clearRect(IIII)V java2d/Intro$Surface.paint(Ljava/awt/Graphics;)V
javax/swing/JComponent.paintToOffscreen(Ljava/awt/Graphics;IIIIII)V
javax/swing/BufferStrategyPaintManager.paint(Ljavax/swing/JComponent;Ljavax/swing/JComponent;Ljava/awt/Graphics;IIII)Z
javax/swing/RepaintManager.paint(Ljavax/swing/JComponent;Ljavax/swing/JComponent;Ljava/awt/Graphics;IIII)V
javax/swing/JComponent._paintImmediately(IIII)V javax/swing/JComponent.paintImmediately(IIII)V
javax/swing/RepaintManager.paintDirtyRegions(Ljava/util/Map;)V
javax/swing/RepaintManager.paintDirtyRegions()V javax/swing/RepaintManager.seqPaintDirtyRegions()V
94
```

Examples: Jarod Jenson, DTrace and Java: Exposing Performance Problems That Once Were Hidden

DTrace Java Example: Monitor Contention (I)

> Bucketed distribution of monitor acquisition wait times, per-thread

```
$./monitor-contend.d `pgrep java`
dtrace: script './monitor-contend.d' matched 2 probes
^C
    30
    value ------------- Distribution ------------- count
[…]
    14
    value ------------- Distribution ------------- count
      1024 |                                         0
      2048 |@@@@                                     12
      4096 |@                                        4
      8192 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           84
     16384 |                                         0
     32768 |@                                        2
     65536 |@                                        2
    131072 |@@@                                      7
    262144 |                                         1
    524288 |                                         0
```

8 times >= 131ms

Examples: Jarod Jenson, DTrace and Java: Exposing Performance Problems That Once Were Hidden

DTrace Java Example: Monitor Contention (II)

> And the D script used in the previous slide…

```
$ cat monitor-wait.d
#!/usr/sbin/dtrace -s
hotspot$1:::monitor-contended-enter
{
   self->ts = timestamp;
}
hotspot$1:::monitor-contended-entered
/ self->ts /
{
   @[tid] = quantize(timestamp - self->ts); self->ts = 0;
}
```

Examples: Jarod Jenson, DTrace and Java: Exposing Performance Problems That Once Were Hidden

Contents

Part II: Cross Platform Java Analysis

Problem Statement

"In the early days […], we would build a Solaris 10 system [port the customer's application to it] and do the [DTrace] analysis. Suggestions were given and changes made to the app, which was then run on its original system [to see if] the gains transferred"

– Jarod Jenson, Chief Systems Architect, AEYSUS
  (co-author of early DTrace Java instrumentation)

Does this work for Java across OSes?

Problem Statement: details

- Dtrace now available on **3 platforms** (Solaris, FreeBSD, MacOS X)
- Are **Java characteristics** as revealed by Dtrace **equivalent**?

Experiment:

- Controlled hardware/software, ***only OS varies***
  - Same hardware: MacBook Pro (triple boot)
  - Same Java version (i.e. not Java 6)
  - Same application

Expectations:

- Different OS threading, scheduling, locking, memory mgmt

Question: How significant are these differences?

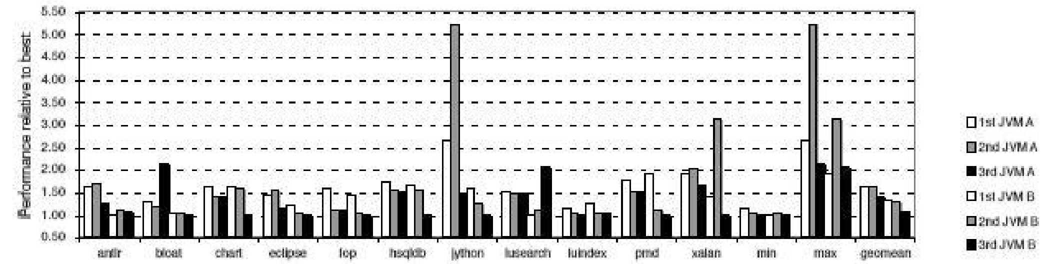
Problem Statement: Targets for Analysis?

> Whole Apps?
  – Broad Coverage, accounts for JVM (dynamic compilation, GC, etc)
> Micro benchmarks?
  – Is there a Java equivalent of lmbench ?

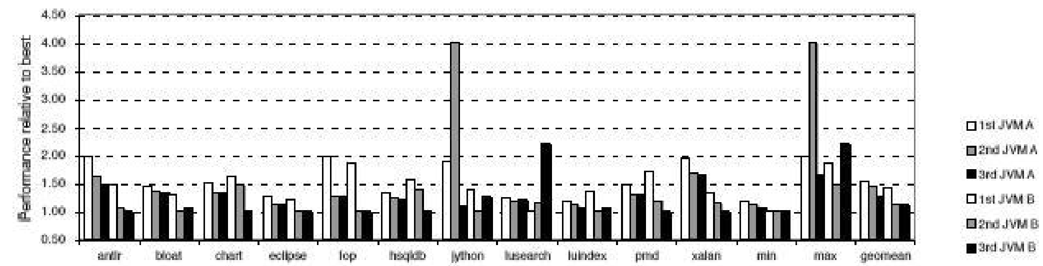
Findings:

> Most scientifically defensible macro benchmarks: **DaCapo**
> No accepted java equivalent of lmbench (roll our own)

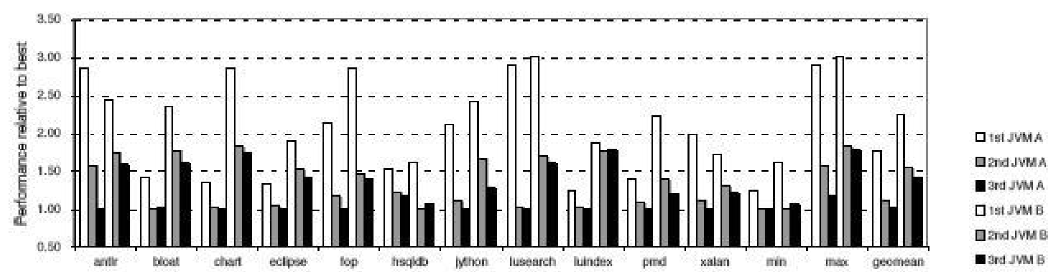
2 JVMs across 3 archs: Results after 3 iterations



(a) AMD



(b) Pentium M



(c) Sparc

The DaCapo Benchmarks

- Like SPEC, suite of 11 constituent benchmarks
- Diverse applications (parser generator, IDE, language interpreter, etc)
- Diverse code complexity
- Diverse objects and memory behavior
- Diverse GC behavior
- Diverse JIT behavior
- Deterministic Reproducibility

Diversity quantified via Principle Component Analysis

Blackburn et al, The DaCapo Benchmarks: Java Benchmarking Development and Analysis, OOPSLA'06

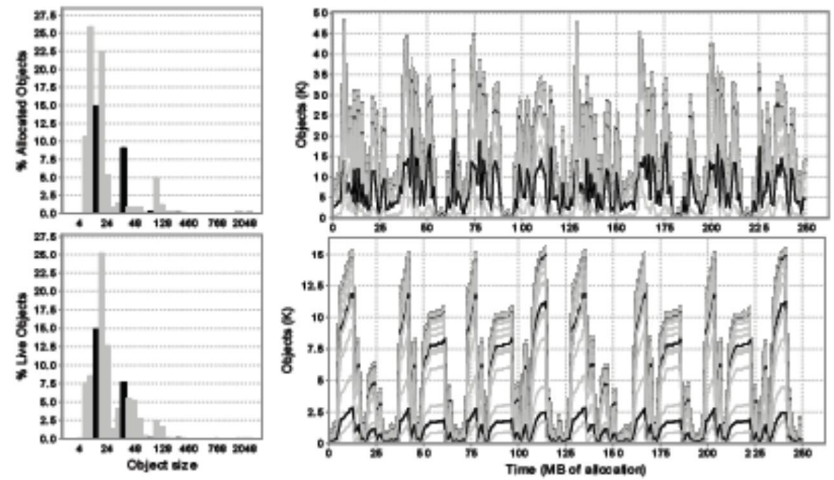
DaCapo: characteristics of antlr component

Benchmark Description and Origin

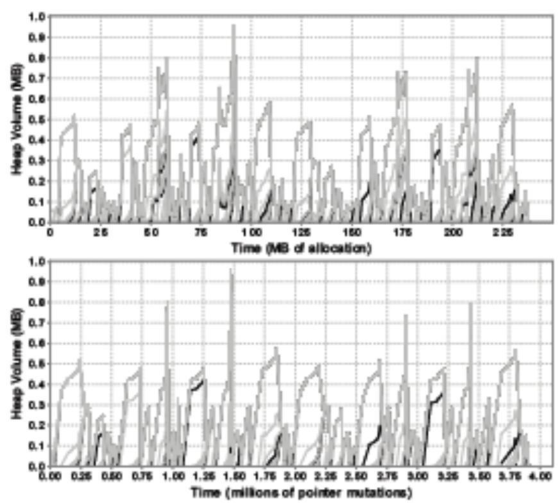
Short Description	A parser generator and translator generator
Long Description	ANTLR parses one or more grammar files and generate a parser and lexical analyzer for each.
Threads	Single threaded
Repeats	Two iterations, each parses 44 distinct grammar files
Version	2.7.2
Copyright	Public Domain
Author	Terence Parr
License	Public Domain

Benchmark Characteristics

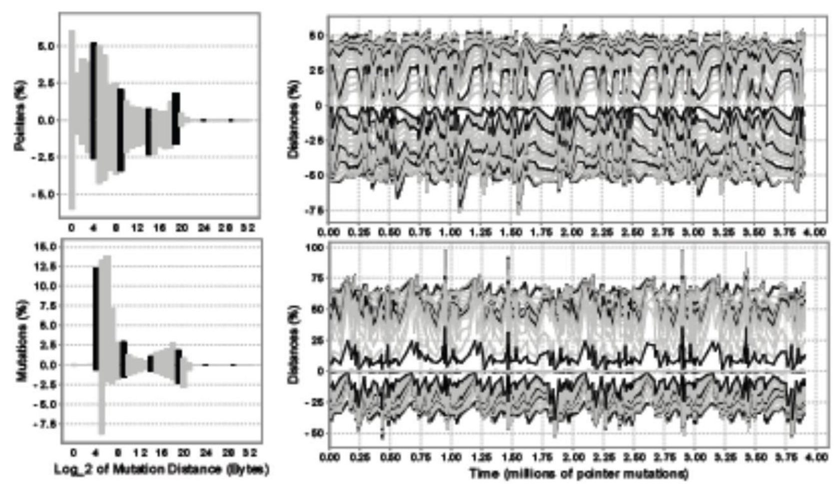
Total Allocation (MB)	237.9
(Obj)	4,208,403
Maximum Live (MB)	1.0
(Obj)	15,566
Pointer Mutations (M)	3.91
Classes Loaded	126



(a) Allocated (above) and Live (below) Object Size Histograms and Time-series



(b) Heap Composition Time-series, in Allocations (above) and Mutations (below)



(c) Snapshot (above) and Mutation (below) Pointer Distance Histograms and Time-series

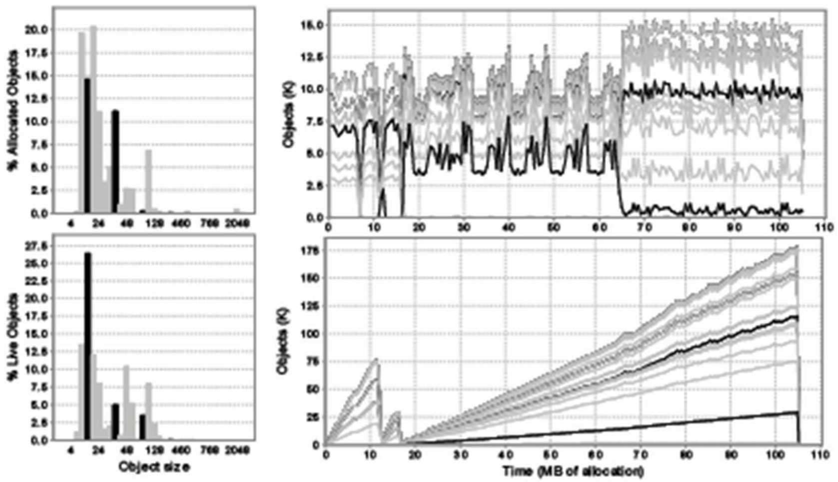
DaCapo: characteristics of fop component

Benchmark Description and Origin

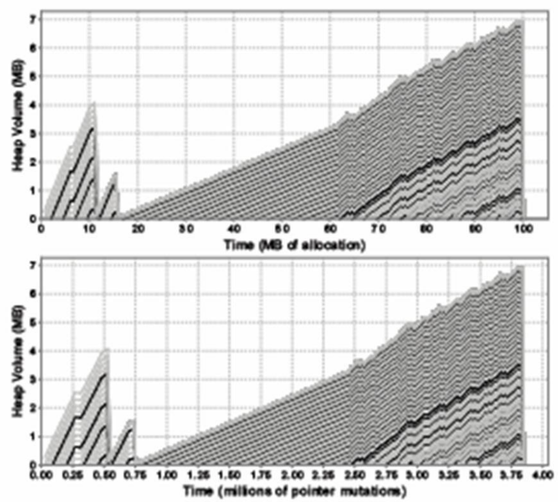
Short Description	An output-independant print formatter
Long Description	fop takes an XSL-FO file, parses it and formats it, generating an encrypted pdf file
Threads	Single threaded
Repeats	Single iteration, renders a single XSL-FO file
Version	0.20.5
Copyright	Copyright (C) 1999-2003 The Apache Software Foundation
Author	Apache Software Foundation
License	Apache Public License

Benchmark Characteristics

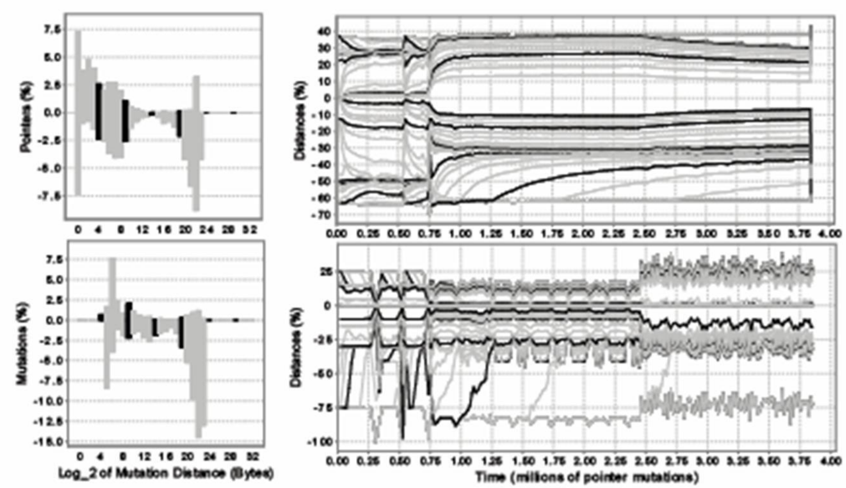
Total Allocation (MB)	100.3
(Obj)	2,402,403
Maximum Live (MB)	6.9
(Obj)	177,718
Pointer Mutations (M)	3.85
Classes Loaded	231



(a) Allocated (above) and Live (below) Object Size Histograms and Time-series



(b) Heap Composition Time-series, in Allocations (above) and Mutations (below)

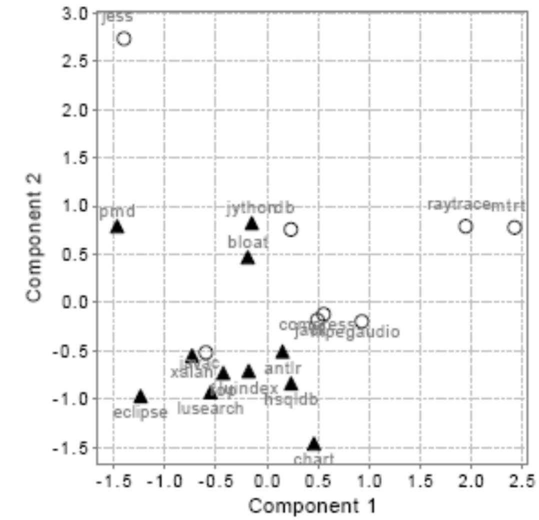


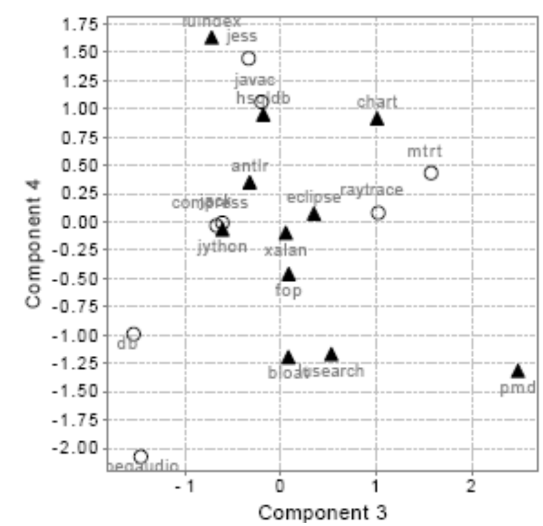
(c) Snapshot (above) and Mutation (below) Pointer Distance Histograms and Time-series

DaCapo: Metrics used for Primary Comp. Anal.

Metric	Rank PC1	PC2	PC3	PC4
Architecture				
Instruction mix – ALU	-9	-15	7	-3
Intruction mix – branches	-10	**-4**	16	1
Instruction mix – memory	**1**	13	-11	-13
Branch mispreds/instruction for a PPM predictor	8	8	12	14
Register dependence distance up to 16	**-5**	-12	-5	12
Register dependence distance between 16 and 64	-15	**-1**	6	-7
Register dependence distance above 64	**6**	**3**	13	-15
Code				
Instruction cache misses in miss/msec	-14	14	**1**	11
Bytecode compiled in KB	-7	11	8	-6
Methods compiled	**-2**	7	9	-9
Memory				
Pointer distance – mean	**3**	-16	10	-5
Pointer distance – standard deviation	**-4**	9	14	16
Mutation distance – mean	16	10	**3**	4
Mutation distance – standard deviation	-12	**2**	15	2
Incoming pointers per object – standard deviation	-13	5	**-2**	-8
Outgoing pointers per object – standard deviation	-11	6	**-4**	-10

Table 5. Metrics Used for PC Analysis and Their PC Rankings





Our hand-rolled microbenchmarks

- Memory exerciser — Mem.java
  - allocates memory in a loop
  - parameter 1: loop count
  - parameter 2: size of memory
- Thread creation/context switching — Zog.java
  - Thread creation and shared data structures
  - parameter 1: number of threads
  - parameter 2: number of messages
- Synchronized data access — SharedVar.java
  - exercise synchronized access
  - parameter 1: number of threads
  - parameter 2: number of calls per thread

Hardware platform

Triple boot MacBook Pro:

- http://blogs.sun.com/ptelles/entry/multiboot_solaris_on_a_macbook
- http://blogs.sun.com/paulm/entry/dual_partitioning_a_macbook_pro

Boot with rEFIt:

- http://refit.sourceforge.net/

Appropriate OSes:

- Mac OS X 10.5 Leopard developer build
- FreeBSD: http://wiki.bsdforen.de/DTrace http://wiki.freebsd.org/AppleMacbook
- Solaris Express: http://opensolaris.org/os/downloads/on/

Results: ETIMEDOUT

- Solaris on MacBook Pro only now (Q2 2007) possible
  - See links
- Authors access to MacOS X 10.5 developer snapshot too limited
  - In Nov 2006, expected Leopard in May 2007 -> Fall (maybe)
- FreeBSD support also not yet fully integrated (FreeBSD 7.0)
- Out of time for this presentation anyway…

Monitor authors' blog for updates…

Conclusions

- Irreplaceable for system visibility
- Not (yet) completely available on FreeBSD or MacOS X
- Still useful for Java even without Java specific probes (JDK >= 1.6)
- If desperate for java probes, can try dvm agent for JDK < 1.6
- Important/critical results achievable with series of one-liners
- Knowledge of OS, C runtime, JVM runtime, and Java app helpful

Links

- DTrace
  - McDougall et al, *Solaris Performance and Tools*, Prentice Hall, 2006
  - http://java.sun.com/javase/6/docs/technotes/guides/vm/dtrace.html
  - http://www.brendangregg.com/dtrace.html (D scripts)
  - http://www.opensolaris.org/os/community/dtrace/dtracetoolkit/
  - http://www.devx.com/Java/Article/33943 (Stolen examples)
  - https://solaris10-dtrace-vm-agents.dev.java.net/ (dvm for JDK <1.6)
- DaCapo
  - http://dacapobench.org/
  - Blackburn et al, The DaCapo Benchmarks: Java Benchmarking Development and Analysis, OOPSLA '06

Jason Brazile

Stefan Tramm

Netcetera

http://netcetera.ch

jason.brazile@netcetera.ch