Cranking Up Java™ Application Performance With DTrace

Adam H. Leventhal
Sun Microsystems, Inc.

Jarod Jenson
Aeysis, Inc.

TS-2698

The State of Systemic Analysis

- Observability tools abound
  - Utilities for observing I/O, networking, applications written in the C, C++, perl, and Java programming languages
- Application-centric tools extremely narrow in scope and not designed for use on production systems
- Tools with system-wide scope present a static view of system behavior—no way to dive deeper

Introducing DTrace

- DTrace is the dynamic tracing facility new in Solaris™ Operating System 10
- Allows for dynamic instrumentation of the OS and applications (including Java applications)
- Available on stock systems—typical system has more than 30,000 probes
- Dynamically interpreted language allows for arbitrary actions and predicates

Introducing DTrace (Cont.)

- Designed explicitly for use on production systems
- Zero performance impact when not in use
- Completely safe—no way to cause panics, crashes, data corruption, or pathological performance degradation
- Powerful data management primitives eliminate need for most postprocessing
- Unwanted data is pruned as close to the source as possible

Providers

- A provider allows for instrumentation of a particular area of the system
- Providers make probes available to the framework
- Providers transfer control to the DTrace framework when an enabled probe is hit
- DTrace has several providers, e.g.:
  - The pid provider for C and C++ applications
  - The syscall provider for system calls
  - The io provider for system I/O

The D Language

- D is a C-like language specific to DTrace with some constructs similar to awk(1)
- Global, thread-local and probe-local variables
- Built-in variables like **`execname`** and **`timestamp`**
- Predicates can use arbitrary expressions to select which data is traced and which is discarded
- Actions to trace data, record stack backtraces, stop processes at points of interest, etc.

DEMO

Simple DTrace Invocations

Aggregations

- Often the patterns are more interesting than each individual datum
- Want to aggregate data to look for larger trends
- DTrace supports the aggregation of data as a first-class operation
- An aggregation is the result of an aggregating function
  - count(), min(), max(), avg(), quantize()
- May be keyed by an arbitrary tuple

DEMO

Using Aggregations

Systemic Analysis With DTrace

- DTrace is a powerful tool for finding problems with the correctness or performance of an application
- Real strength is in understanding the interaction between the application and the rest of the system
- Business solutions are increasingly constructed from heterogeneous components
- Finding the system bottleneck requires understanding the interaction between those components and the operating system

Systemic Analysis With DTrace

- Higher layers of abstraction allow for greater leverage
- Complex tasks can be easily performed—both by design and by accident
- Can be vital to understand how high-level actions impact the underlying resources at the lowest level

DEMO

Observing Low-Level Impact

DTrace for Java Applications

- The simplest use of DTrace for Java applications is to record the call stack
- Rather limited, but still extremely useful
- Use the `jstack()` from any DTrace probe to record the Java application's stack trace
- Especially useful to understand I/O and scheduler behavior and interaction with the underlying system libraries

DEMO

The `jstack()` Action in Action

Tracing Java Application Behavior

- Probes for method entry and return
  - Add-on **`dvm`** provider on Java 2 Platform, Standard Edition (J2SE™ platform) 5.0
  - Built-in **`hotspot`** provider on Java Platform, Standard Edition (Java SE platform) 6
- No performance impact when not in use—dynamically enabled
- Some probes with overly onerous probe effect require an option at execution time:
  - **`-XX:+ExtendedDTraceProbes`**
- Or on a running Java Virtual Machine (JVM™) from the command-line:
  - **`jinfo -XX:+ExtendedDTraceProbes`**

The terms “Java Virtual Machine” and “JVM” mean a Virtual Machine for the Java™ platform.

`hotspot/dvm` Provider Probes

- Some basic Java technology “lifecycle” probes
  - vm-init, vm-death, thread-start, thread-end
- Class loading probes
  - class-load, class-unload
- GC and memory allocation probes
  - gc-start, gc-finish, object-alloc, object-free
- Probes dealing with method invocation
  - method-entry, method-return

DEMO

Tracing Java Applications With DTrace

Since Last Year

- DTrace allows C/C++ developers to add probes to their applications and libraries
  - Empowers users without requiring low-level knowledge
- Called USDT—user-land statically defined tracing
- Java statically defined tracing in the works for Java SE platform 7
- Here's what it's going to look like…

DEMO

For the First Time Ever: Java Statically Defined Tracing

Still Not the Final Story

- Every year DTrace support for Java applications gets better
- Still more in the works:
  - Fine-grained method probes (a la the pid provider)
  - Simplified dynamic enabling of expensive probes
  - More probes in the JVM software
  - Arguments to method probes
  - Java technology datatypes in DTrace
  - Improvements to `jstack()`
- Look for some of these in Java SE platform 7

Summary

- DTrace allows for unprecedented systemic analysis—critical for increasingly complex systems
- DTrace gives developers of Java applications the ability to understand their application's behavior and its interactions with the system
- System administrators can identify bottlenecks in Java applications
- Great support for Java technology—and getting better!

For More Information

- The DTrace home page
  http://www.opensolaris.org/os/community/dtrace/
- DTrace JVM software agent
  https://solaris10-dtrace-vm-agents.dev.java.net/
- The Solaris Dynamic Tracing Guide
  http://docs.sun.com/app/docs/doc/817-6223
- Some blog entries about the DTrace JVM software agent
  - http://blogs.sun.com/roller/page/ahl/20050418#dtracing_java
  - http://blogs.sun.com/roller/page/bmc/20050418#your_java_fell_into_my
  - http://blogs.sun.com/roller/page/ahl/20050529#java_debugging_w_dtrace
  - http://blogs.sun.com/roller/page/kto/20050413#java_vm_agents_and_solaris1

Q&A

Adam H. Leventhal
ahl@sun.com

Jarod Jenson
jarod@aeysis.com

Cranking Up Java™ Application Performance With DTrace

Adam H. Leventhal
Sun Microsystems, Inc.

Jarod Jenson
Aeysis, Inc.

TS-2698