DTrace and Java:
Understanding the Application and the Entire Stack

Adam H. Leventhal
Solaris Kernel Development
Sun Microsystems

Jarod Jenson
Aeysis, Inc.

Session 5211

The State of Systemic Analysis

- Observability tools abound
  - Utilities for observing I/O, networking, applications written in C, C++, Java, perl, etc.
- 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 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

Some 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
- Do more with less – both intended and unintended
- Can be vital to understand how high level actions impact the underlying resources at the lowest level

DEMO

Observing low-level impact

DTrace and Java

- Initially, the only DTrace interaction with Java was an action to record a Java stack backtrace
- Rather limited, but still extremely useful
- Use the `jstack()` from any DTrace probe to deduce the Java call chain
- Especially useful to understand I/O and scheduler behavior and interaction with the underlying system libraries

DEMO

Using the `jstack()` action

DTrace/Java Limitations

- The jstack() action is *very* useful
- Would like to be able to instrument method entry and return – as the `pid` provider enables for C and C++ applications
- Also need some probes for JVM services such a object allocation, class loading, garbage collection

DTrace JVM Agent

- Existing JVM instrumentation interfaces allow for instrumentation at a number of points of interest
  - JVM Tool Interface (JVMTI)
  - JVM Profiler Interface (JVMPI)
- The DTrace agent creates DTrace probes using the JVMTI and JVMPI interfaces
- Brings Java observability into the folds of the DTrace framework
- Offers DTrace probes via the **dvm** provider

`dvm` Provider Probes

- Some basic Java "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

Some examples of the `dvm` provider

Not the Final Story

- The DTrace Java agent allows for *extensive* observability into Java and leverages the power of the DTrace framework
- Unfortunately requires the application to be started with special options to the JVM
  - No **`dvm`** provider unless the application was started with these options
  - Some performance impact even when probes are not enabled – violates one of the constraints of DTrace

Future Work

- Work in the JVM and the DTrace framework is ongoing to improve the interaction
- Goal to provide similar functionality of the DTrace Java agent
- Ensure zero overhead when disabled
- Expose more information from the JVM such as method arguments

Summary

- DTrace allows for unprecedented systemic analysis – critical for increasingly complex systems
- The Java agent fills the gap in DTrace's coverage of the system
- Java developers can optimize applications for system performance
- System administrators can identify system bottlenecks in Java applications

For More Information

List

- The DTrace home page
  http://www.sun.com/bigadmin/content/dtrace/
- DTrace JVM 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 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 Leventhal
ahl@sun.com

Jarod Jenson
jarod@aeysis.com

DTrace and Java:
Understanding the Application and the Entire Stack

Adam H. Leventhal
Solaris Kernel Development
Sun Microsystems

Jarod Jenson
Aeysis, Inc.

Session 5211