DTrace and Java™ Technology: Down and Dirty With Your Application

Adam H. Leventhal
Sun Microsystems

Jarod Jenson
Aeysis, Inc.

TS-1311

The State of Systemic Analysis

- Observability tools abound
  - Utilities for observing I/O, networking, applications written in the C, C++, Java™, perl, etc. 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 10
- Allows for dynamic instrumentation of the OS and applications (including Java-based 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 and Java Technology

- Initially, the only DTrace interaction with the Java platform 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-based 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 VM 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 Java VM services such a object allocation, class loading, garbage collection

Last Year's JavaOne℠ Conference

- Existing Java VM instrumentation interfaces allow for instrumentation at a number of points of interest
  - Java Virtual Machine Tool Interface (JVMTI)
  - Java Virtual Machine Profiler Interface (JVMPI)
- The DTrace agent creates DTrace probes using the JVMTI and JVMPI interfaces
- Initial Java technology observability via the **`dvm`** provider
- Good enough to make tremendous improvements
- ...But with some limitations

Areas of Improvement Since Last Year

- Significant probe effect even when not enabled—contrary to the principles of DTrace
- Required additional installation and configuration
- Needed to start the Java VM with special options
- Added probes to the Java VM itself and greatly improved the Java VM agents

DTrace Support in Mustang

- The **`hotspot`** provider is built in the Mustang Java VM (JDK™ 6 software)
  - Same probes as the **`dvm`** provider
- Many probes are always available for any Java technology-based application
- Some probes with overly onerous probe effect require an option at execution time:
  - **`-XX:+ExtendedDTraceProbes`**
- **`jinfo`** dynamically specify this flag to the Java VM

Improved Java VM Agents

- Enabled probe effect was halved by optimizing the agent code (using DTrace of course)
- Disabled probe effect was greatly reduced by allowing truly dynamic enabling
- New agents allow for a named pipe
  - `java -Xrundvmti:all,dynamic=/tmp/dvmpipe`
- Input on the pipe enables and disables instrumentation

`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 Technology-based Applications With DTrace

Still Not the Final Story

- The DTrace Java VM agent allows for extensive observability into the Java platform and leverages the power of the DTrace framework
- Great strides have been made in the past year
- Future work will reduce the disable and enabled probe effect, and allow for fine-grained instrumentation, tracing of arguments and Java VM data elements

Summary

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

For More Information

- The DTrace home page
  http://www.opensolaris.org/os/community/dtrace/
- DTrace Java VM 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 Java VM 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

DTrace and Java™ Technology: Down and Dirty With Your Application

Adam H. Leventhal
Sun Microsystems

Jarod Jenson
Aeysis, Inc.

TS-1311