DTrace Topics: DTraceToolkit

Brendan Gregg
Sun Microsystems
April 2007

```
 ragma D option quiet
pragma D option switchrate=10hz

/*
 * Print header
 */
dtrace:::BEGIN
{
        /* print optional headers */
        OPT_time    ? printf("%-14s ", "
        OPT_timestr ? printf("%-20s ", "
        OPT_zone    ? printf("%-10s ", "
        OPT_proj    ? printf("%5s ", "PR

        /* print main headers */
        OPT_dump    ? printf("%s %s %s %
            "TIME", "ZONE", "PROJ", "UID
            printf("%5s %6s %6s %s\n", "



        t exec event

        ::exec:return, syscall::execve:re
        ER == 0) || (OPT_cmd == 1 && COMM

        /* print optional fields */
        OPT_time ? printf("%-14d ", time
```

DTrace Topics: DTraceToolkit

- This presentation is about the DTraceToolkit, and is part of the “DTrace Topics” collection.
  - Difficulty: 
  - Audience: Everyone
- These slides cover:
  - What is the DTraceToolkit
  - What isn't the DTraceToolkit
  - Downloading
  - Contents
  - Testing & Impact
  - Quick Wins

What is the DTraceToolkit

- A collection of over 100 DTrace scripts for both the Solaris 10+ and OpenSolaris operating systems.
- The toolkit is intended to provide scripts for:
  - quick wins
  - performance observability
  - troubleshooting and debugging
  - examples of DTrace for both beginners and experts
- Not everyone has both the programming skills and the time to learn DTrace. The toolkit provides fast value from DTrace without needing to code.

What isn't the DTraceToolkit

- Magical
  - As with other tools, the DTraceToolkit helps fetch useful statistics, but you must draw the conclusions.
- All of DTrace
  - The field of DTrace is much bigger than the toolkit.
- Written by Sun
  - The DTraceToolkit became an OpenSolaris project, but is not an officially supported Sun product.

Downloading the DTraceToolkit

- The DTraceToolkit has an OpenSolaris URL, and can still be found by its original URL,
- http://www.opensolaris.org/os/community/dtrace/dtracetoolkit
- http://www.brendangregg.com/dtrace.html
- After downloading:
  1. gunzip and "tar xvf" the file. cd to the toolkit directory
  2. run ./install (optional, you can use the toolkit without doing this)
  3. read Guide to find out how to get started
  4. a list of scripts is in Docs/Contents

Contents

- This section discusses the toolkit components.
- Major Components:
  1. The scripts themselves
  2. A man page for every script
  3. An examples file for every script
- Important Directories:
  - Bin — symlinks to all the scripts
  - Man — man pages
  - Docs/Examples — examples

Contents

- The top level directory contains the top dozen or so most useful scripts. Other directories and files are:

```
DTraceToolkit-X.XX/
        Bin/                    Symlinks to the scripts
        Apps/                   Application specific scripts
        Cpu/                    Scripts for CPU analysis
        Disk/                   Scripts for disk I/O analysis
        Docs/                   Documentation
           Contents             Command list for the Toolkit
           Examples/            Examples of command usage
           Faq                  Frequently asked questions
           Links                Further DTrace links
           Notes/               Notes on Toolkit commands
           Readme               Readme for using the docs
        Extra/                  Misc scripts
        Guide                   This file!
[...continued...]
```

This is from the README file

Contents

```
[...continued...]
        Kernel/                 Scripts for kernel analysis
        License                 The CDDL license
        Locks/                  Scripts for lock analysis
        Man/                    Man pages
           man1m/               Man pages for the Toolkit commands
        Mem/                    Scripts for memory analysis
        Net/                    Scripts for network analysis
        Proc/                   Scripts for process analysis
        System/                 Scripts for system analysis
        User/                   Scripts for user based activity analysis
        Zones/                  Scripts for analysis by zone
        Version                 DTraceToolkit version
        install                 Install script, use for installs only
```

Scripts

- The scripts examine numerous areas of system behavior, including:
  - CPUs
  - disks
  - memory system
  - network interfaces
  - kernel
  - processes
  - user-land code

Script Naming

- If a script end in a ".d" suffix, then it is a pure DTrace script (and will start with #!/usr/sbin/dtrace):

```
DTraceToolkit-0.96$ more Disk/iofileb.d
#!/usr/sbin/dtrace -s
/*
 * iofileb.d - I/O bytes by filename and process.
 *             Written using DTrace (Solaris 10 3/05).
[...]

dtrace:::BEGIN
{
        printf("Tracing... Hit Ctrl-C to end.\n");
}
[...]
```

we begin in /usr/sbin/dtrace

> These scripts usually don't have command line options.

Script Naming

- Scripts that don't end in ".d" are DTrace wrapped in either Perl or shell for enhanced functionality:

```
DTraceToolkit-0.96$ more Proc/fddist
#!/usr/bin/sh
#
# fddist - file descriptor usage distributions.
[...]
### Process options
while getopts hrw name
[...]
# --- Main Program, DTrace ---
#
/usr/sbin/dtrace -n '
 #pragma D option quiet
```

here we are /usr/bin/sh

using the shell's getopts

now entering /usr/sbin/dtrace

> Try running these with "-h" for a USAGE message

Script Style

- The D scripts have been written to follow most of the standards from `cstyle` (Sun's C code checker).
- The scripts are intended to be read as a reference.
  - Since many are less than 100 lines of code, they are easy to read and will help you learn DTrace.
- The headers are also carefully written, and follow a toolkit standard to neatly convey essential details.

Tip:
If you are reading the scripts as a way to learn DTrace, it may be best to start with the smallest scripts first. Scripts larger than 10 Kbytes are usually very complex.

```
DTraceToolkit-0.96$ more Kernel/cpudists
#!/usr/bin/sh
# filename   sentence description
# cpudists - print CPU time distributions by Kernel/Idle/Processes.
#            Written using DTrace (Solaris 10 3/05).  language, OS
# date, version
# 22-Sep-2005, ver 0.73              (check for newer versions)
#                synopsis
# USAGE:         cpudists [-ahV] [-t top] [interval [count]]
#
#                -a                # print all processes     options
#                -V                # don't print timestamps
#                -t num            # print top num  only
#  eg,
#                cpudists 1        # print every 1 second    examples
#                cpudists -a 10    # print all processes every 10 secs
#
#
# FIELDS:        output fields
#                value             The following or the process name,
#                IDLE              Idle time - CPU running idle thread
#                KERNEL            Kernel time - Kernel servicing interrupts, ...
[...]
```

Oneliners

- Apart from scripts, the DTraceToolkit contains a list of useful one-liners. These are great because:
  - no towing scripts around, just copy-n-paste
  - helps you learn DTrace in small easy steps
  - one liners may have a faster site approval than scripts!
- They are in the toolkit as Docs/oneliners.txt.

  They were also listed as Appendix B in “Solaris Performance and Tools”, Prenctice Hall.

Anecdote:

Brendan has had many emails to the effect of “Thanks for all the scripts, although the one-liners were enough to solve all our issues.”

```
DTraceToolkit-0.96$ more Docs/oneliners.txt
#
#   DTrace OneLiners
#

DTrace One Liners,

# New processes with arguments,
dtrace -n 'proc:::exec-success { trace(curpsinfo->pr_psargs); }'

# Files opened by process name,
dtrace -n 'syscall::open*:entry { printf("%s %s",execname,copyinstr(arg0)); }'

# Files created using creat() by process name,
dtrace -n 'syscall::creat*:entry { printf("%s %s",execname,copyinstr(arg0)); }'

# Syscall count by process name,
dtrace -n 'syscall:::entry { @num[execname] = count(); }'

# Syscall count by syscall,
dtrace -n 'syscall:::entry { @num[probefunc] = count(); }'
[...]
```

Man Pages

- The Man directory has a man page for every script.

```
DTraceToolkit-0.96$ MANPATH=Man man iosnoop
Reformatting page.  Please Wait... done

USER COMMANDS                                              iosnoop(1m)

NAME
     iosnoop - snoop I/O events as they occur. Uses DTrace.

SYNOPSIS
     iosnoop [-a|-A|-Deghinostv] [-d device]  [-f  filename]  [-m
     mount_point] [-n name] [-p PID]

DESCRIPTION
     iosnoop prints  I/O  events  as  they  happen,  with  useful
     details such as UID, PID, block number, size, filename, etc.
[...]
```

Docs/Examples Directory

- This contains examples for every script in action, and discusses their output.
- Ever gone straight to the examples when reading a man page? The DTraceToolkit encourages this by providing seperate files.

Experience:
Some people have found the example files the best form of documentation in the toolkit; this includes the author of most of the scripts, who himself has a little difficulty remembering which of the 100+ scripts does what.

```
DTraceToolkit-0.96$ more Docs/Examples/errinfo_example.txt
This is an example of the errinfo program, which prints details on syscall
failures.

By default it "snoops" syscall failures and prints their details,

   # ./errinfo
              EXEC          SYSCALL  ERR  DESC
        wnck-applet             read   11  Resource temporarily unavailable
              Xorg             read   11  Resource temporarily unavailable
           nautilus             read   11  Resource temporarily unavailable
              Xorg             read   11  Resource temporarily unavailable
              dsdm             read   11  Resource temporarily unavailable
              Xorg             read   11  Resource temporarily unavailable
              Xorg           pollsys    4  interrupted system call
        mozilla-bin         lwp_park   62  timer expired
   gnome-netstatus-            ioctl   12  Not enough core
   [...]

which is useful to see these events live, but can scroll off the screen
somewhat rapidly.. so,
[...]
```

Docs/Notes directory

- This contains a collection of FAQ style files which document miscellaneous tool nuances.

```
DTraceToolkit-0.96$ more Docs/Notes/ALLsnoop_notes.txt
The following are additional notes on ALL of the *snoop programs (such as
execsnoop, iosnoop, ..., and dapptrace, dtruss),


* The output seems shuffled?

Beware - due to the way DTrace works, on multi-CPU systems there is no
guarentee that if you print traced events the output is in the same order that
the events occured.
[...]
```

> Known issues with tools are discussed in these files.

Bin Directory

- This directory contains symlinks to all scripts.
- This directory is handy for grepping for examples of DTrace functions, as it links to over 100 scripts. Here we search for examples of lquantize():

```
DTraceToolkit-0.96/Bin$ grep lquantize *
cpuwalk.d:      @sample[pid, execname] = lquantize(cpu, 0, MAXCPUID, 1);
dexplorer:              @length[cpu] = lquantize(this->num, 0, 100, 1);
diskhits:       @Location = lquantize(this->kb, 0, FILE_KB_MAX, SCALE_KB);
dispqlen.d:         lquantize(curthread->t_cpu->cpu_disp->disp_nrunnable, 0, 64,
dnlcps.d:       @Result[execname, pid] = lquantize(this->code, 0, 1, 1);
[...]
```

> Another place to grep is /usr/demo/dtrace

Future Contents

- So far the toolkit has been designed for the Solaris 10 3/05 release (so far meaning version 0.96).
- Newer versions of Solaris and OpenSolaris provide more DTrace probes; future versions of the toolkit should contains scripts that use these probes.
- Desired future category additions:
  - Java, JavaScript
  - NFS, iSCSI
  - Hardware (PIC observability)
  - Stable TCP/IP scripts

Testing

- Each script is tested for a variety of workloads on a variety of systems.
- Where possible, a known workload is created and the numbers are compared to what DTrace has measured.

Anecdote:
Far more effort goes into testing the scripts than actually writing them. Some scripts took around 15 minutes to write and over 3 hours to test.

Opinion:
If it isn't tested, it doesn't work.

Performance Impact

- Enabling DTrace to monitor events has little effect on the system, especially when compared to the behaviour of truss.
- The impact is proportional to how often the events occur that you are monitoring.
- DTrace will abort tracing if it detects it has consumed too much CPU. This is one of the DTrace safety measures.

Tip:
An event rate of over 1000/sec is when you may start to notice CPU cost.

Performance Impact

- The following numbers have been provided as an approximation:
- Fixed rate scripts
  - Usually scripts that use the profile::: provider.
  - For example, dispqlen.d samples at 1000 hz.
  - The impact will be negligible, close to 0% CPU. (in testing, 0.1% CPU).
  - While these have the advantage of low impact, they are usually fixed rate *sampling* scripts, which introduces a degree of error.

Performance Impact

- Demand rated scripts
  - The impact depends on the rate of events (per second).
  - Tracing “slow” disk events may cost less than 0.2% CPU.
  - Tracing process creation would expect even fewer events, costing closer to 0.0% CPU.
  - Tracing very rapid events can cost over 10% CPU. For example, running dapptrace on Xorg (over 6000 lines of output per second) consumed around 10% of a CPU.

Tip:
Fast scrolling output consumes CPU.
Do you really want that much output?
Try summarizing data with DTrace instead.

Performance Impact

- Heavyweight scripts
  - A few scripts in the toolkit must probe either a ton of different events, or very rapid events, or both. They are going to hurt and there is no way around it.
  - The worst would be cputimes and cpudists, they easily chew over 5% of the CPUs.

False:

The worst possible script is one that *DTraces DTrace* in a feedback loop. Due to exponential growth, the server would quickly consume enough energy to cause the heat death of the Universe. Our lives are spared by following lines of code from uts/*/dtrace/fbt.c:

```
if (strcmp(modname, "dtrace") == 0)
        return;  /* and save the universe */
```

What's next:

- We just covered:
  - *What is the DTraceToolkit*
  - *What isn't the DTraceToolkit*
  - *Downloading*
  - *Contents*
  - *Testing & Impact*
- Next up is:
  - Quick Wins
  - Cool Scripts

Quick Wins

- Start with the oneliners:

```
> more Docs/oneliners.txt
```

- Try the following scripts,

```
1../execsnoop -v
2../iosnoop
3../opensnoop -e
4../errinfo -c
5../procsystime -aT
6../iotop -PCt8
7../rwtop -Ct8
8../Disk/iopattern 1
```

Cool Scripts

- See Docs/Examples for demos of these scripts:
  - `bitesize.d`
  - `fddist.d`
  - `rwbbypid.d`
  - `fsrw.d`
  - `iofile.d`
  - `iopending.d`
  - `pathopens.d`
  - `pfilestat`
  - `rfileio.d`
  - `threaded.d`
  - `dispqlen.d`
  - `pridist.d`
  - `nfswizard.d`
  - `dvmstat`
  - `shellsnoop`

Finding Support

- As the DTraceToolkit is an open source product, and there is no official company offering support.
- If you post messages to the OpenSolaris DTrace discuss mailing list, a volunteer may help you out.
  - http://www.opensolaris.org/os/community/dtrace
- Many DTrace experts respond to the dtrace-discuss mailing list.

dtrace:::END

Brendan Gregg
brendan@sun.com