DTrace for SysAdmins
An introduction to the DTraceToolkit

Ryan Matteson
matty91@gmail.com
http://prefetch.net

What is the DTraceToolkit?

- Collection of DTrace scripts written by Brendan Gregg to observe system and application behavior
- Over 105 scripts are currently available to observe CPU, memory, I/O, process scheduling, network activity, userland applications and much much more …

How is the toolkit organized?

- The toolkit is arranged as a series of directories, with each directory containing scripts to observe a specific subsystem (e.g., virtual memory)
- The “Bin” directory contains symbolic links to all of the scripts in the toolkit
- The “Docs” directory contains documentation, and a description of each script

Where can I grab the toolkit?

- The latest version can be retrieved from brendangregg.com, or the opensolaris.org DTrace community website
- The following alias is useful for retrieving and installing the latest version of the toolkit:

```
alias grabtoolkit="cd /opt && /usr/sfw/bin/wget -q -O - \
http://www.brendangregg.com/DTraceToolkit-latest.tar.gz \        |
/usr/sfw/bin/gtar xz"
```

Monitoring CPU activity

- The DTraceToolkit comes with several scripts to monitor interrupts, scheduling behavior, context switching and CPU utilization
- All of the scripts related to the CPU are stored in the “Cpu” directory

Observing the CPU dispatcher

- The CPU dispatcher maintains one or more queues of “runnable” processes, and schedules these onto available CPU resources
- The dispqlen.d script can be used to measure the number of “runnable” processes in each queue:

```
 $ dispqlen.d
Sampling... Hit Ctrl-C to end.
CPU 0
           value  ------------- Distribution ------------- count
               5 |                                         0
               6 |                                         55
```

Measuring CPU utilization

- The cputimes scripts can be used to measure how much CPU time is being consumed by each process:

```
$ cputimes -a 5
2006 Sep 10 20:16:22,
THREADS        TIME (ns)
      fmd          125016
   se.i386         127911
    dtrace         2088111
   fsflush         10904127
KERNEL             27022234
      orca         4932396085
      IDLE         14861393273
```

Monitoring virtual memory

- The DTraceToolkit comes with several scripts to monitor virtual memory usage
- There are also several scripts available to view and report on swap utilization
- All of the scripts related to virtual memory and swap are located in the “Mem” directory

Monitoring paging activity

- The vmstat “-p” option provides system wide paging activity, but there are times when you want to see how a specific process is impacting the virtual memory subsystem
- dvmstat can be used to retrieve paging activity for all processed with a specific name, or for a process with a specific process id:

```
$ dvmstat -n bash
   re   maj    mf   fr  epi  epo  api  apo  fpi  fpo    sy
  584    0  4496    0    0      0    0    0      0     0   577
   72    0  3272    0    0      0    0    0      0     0  1396
```

Monitoring network activity

- The DTraceToolkit comes with several scripts to monitor the TCP/IP and UDP/IP stacks
- There are also scripts to monitor HTTP requests and NFS activity
- These scripts are located in the “Net” and “Apps” directories

Monitoring TCP connections

- The connections script can be used to watch active connections on a system
- To view connection data in a “top”-like display, the tcptop script can be used
- To display TCP connections, the tcpsnoop script can be used
- To display UDP connections, the udpsnoop.d script can be used
- Due to Solaris bug #6315039, these scripts are currently broken in GA releases of Solaris (the bug is fixed in opensolaris)

Monitoring NFS client operations

- Monitoring NFSv3 client behavior prior to Solaris 10 was a chore (e.g., correlating truss, snoop and nfsstat was a nightmare!)
- I wrote the nfsclientstats.pl* to assist with correlating NFSv3 file system operations (also referred to as VOPs) to processes:

```
$ nfsclientstats.pl
process     read write readdir getattr setattr lookup access create remove
rename mkdir orca         3328  194          0   5496        6     6882   8246      12
0      0     0     0
 rm           0          0       760     950        0     2850    5320       0     190     0      0
190
touch        0          0         0     378      189     1512    1323     189       0       0      0
0
```

Tracing NFS operations

- Monitoring physical vs. logical NFS I/O was also a chore prior to Solaris 10 (anyone remember prex?)
- To determine how often an NFS operation caused a physical network I/O to occur, and to measure the latecy of each operation, I developed the nfstrace script:

```
$ nfstrace
Executable Operation     Type      Time     Size   Path
mkdir      nfs3_lookup   physical  359953   N/A    /opt/htdocs/test
mkdir      nfs3_getattr  logical   17481    N/A    /opt/htdocs/test
mkdir      nfs3_getattr  logical   7577     N/A    /opt/htdocs/test
cat        nfs3_read     logical   54848    8192   /opt/htdocs/test/1
```

* Script available at http://prefetch.net

Monitoring disk Activity

- The DTraceToolkit comes with several scripts to view physical and logical I/O
- There are also several scripts to profile application I/O behavior
- Scripts related to I/O are located in the “Disk” directory

Monitoring physical I/O

- The iotop utility can be used to view physical I/O in a “top”-like display
- It’s counterpart, iosnoop, can be used to display block I/O as it happens:

```
$ iosnoop
DEVICE    UID   PID D    BLOCK   SIZE      COMM PATHNAME
 cmdk0      100  3154 R    81824     3072       cat       /etc/default/nfs
cmdk0       100  3162 R  1050110     1024        ls       /etc/aliases
cmdk0       100  3242 R  1050634     2048       cat
/etc/default/inetinit
cmdk0         0     3    W    36726     1024       fsflush /var/cron/log
```

Monitoring logical I/O

- The rwtop utility can be used to view logical I/O in a “top”-like display:
- It’s counterpart, rwsnoop, can be used to display logical I/O as it happens:

```
$ rwsnoop
UID    PID CMD       D   BYTES FILE
  0   4536   more      W      41      /devices/pseudo/pts@0:1
100   2958 sshd       R      42       /devices/pseudo/clone@0:ptm
  0    540   orca      W    8192    /opt/data/foo1
  0    540   orca      W    8192     /opt/data/foo2
  0    540   orca      W    8192     /opt/data/foo3
```

Measuring application I/O patterns

- iopattern can be used to measure sequential and random I/O system wide (useful for tailoring file systems to suit specific workloads)
- seeksize.d can be used to determine if an individual process is performing sequential or random I/O
- bitesize.d can be used to measure the quantity and size of each I/O performed by an application

Measuring I/O wait

- The iopending and iofile.d scripts can be used to measure how much time an application spends waiting for I/O:

```
$ iofile.d
Tracing... Hit Ctrl-C to end.
PID CMD              TIME FILE
3442 cron             13995 /var/adm/lastlog
3442 cron             14374 /etc/default/login
3451 cron             16979 /var/adm/lastlog
255   cron            17310 /var/cron/log
386 syslogd           22261 /var/adm/messages
3456 sadc             24078 /var/adm/sa/sa10
3456 sadc             26327 /var/adm/sa/
```

Monitoring processes

- The DTraceToolkit comes with several scripts to observe processes and profile applications
- These scripts are located in the “Procs,” “Apps,” “Users” and “System” directories

Monitoring calls to exec*()

- The execsnoop script can be used to watch calls to the exec() family of functions:

```
$ execsnoop
UID    PID   PPID ARGS
  0   4676   3273 ls -l
  0   4677   3273 ps -ef
  0   4677   3273 ps -ef
  0   4678   3273 cat /etc/system
```

Monitoring calls to open*()

- The opensnoop script can be used to capture calls to the open() family of functions:

```
$ opensnoop
UID    PID COMM          FD PATH
    0    540 topen         69 /opt/data/foo1
    0    540 topen         69 /opt/data/foo2
    0    540 topen         69 /opt/data/foo3
    0    540 topen         69 /opt/data/foo4
```

Miscellaneous process scripts

- newproc.d can be used to watch processes as they are created
- errinfo can be used to watch errno values as they are generated
- procsystime can be used to determine how much CPU time is spent in each system call
- dapptrace and dappprof can be used to profile applications

Conclusion

- DTrace is an invaluable addition to Solaris
- There are over a hundred extremely useful scripts that can be used to derive useful debugging and profiling data
- The DTraceToolkit allows system adminisatrators to solve real problems without needing to crack open the Dtrace users guide, Solaris Systems Programming or Solaris Kernel Internals

References

- Dtrace users guide
  - http://www.opensolaris.org/os/community/dtrace/
- DTraceToolkit website
  - http://brendangregg.com
- Top Ten Dtrace scripts
  - http://prefetch.net
- Observing I/O behavior with the DTraceToolkit
  - http://prefetch.net
- Understanding vmstat and mpstat output with Dtrace
  - http://prefetch.net

Questions?