DTrace for Storage Development

Dominic Kay, Sun Microsystems

Agenda

- What is dtrace.
- How does it work: theory & simple examples.
- Examples in the storage domain.
- scsi.d
- Providers of note.
- Further work.
- Resources
- Q&A

Dtrace is...

- A dynamic troubleshooting and analysis tool first introduced in the Solaris 10 and OpenSolaris
  - Subsequently ported to Mac OS X and FreeBSD
- DTrace is many things, in particular:
  - A tool
  - A programming language interpreter
  - An instrumentation framework
- Provides observability across entire software stack allowing you to examine software execution as never before.

Dtrace Strengths

- A new powerful framework for real-time analysis and observability. System and process centric
- Hard to debug transient problems with: truss(1), pstack(1), prstat(1M), iostat(1M)
- Only mdb(1) designed for systemic problems but only for postmortem analysis
- Designed for live production systems: a totally safe way to inspect live data on production systems

Dynamic Tracing

- Ease-of-use and instant gratification engenders serious hypothesis testing
- Instrumentation directed by high-level control language (not unlike AWK or C) for easy scripting and command line use
- Build your DTrace toolbox
- Comprehensive probe coverage and powerful data management allow for concise answers to arbitrary questions
- What is this system doing ...?

Dynamic Tracing (cont.)

- Safe and comprehensive: tens-of-thousands of data monitoring points
- Inspect kernel and user space level
- Reduced costs: problems usually found in minutes or hours, not days or weeks
- Flexibility: DTrace lets you create your own custom programs to dynamically instrument the system
- No need to instrument your applications via source code modifications, no need to stop or restart them

Probes

- A point of instrumentation, made available by a provider, which has a name
- A four-tuple name uniquely identifies every probe
- provider:module:function:name
- Module and Function: places where you want to look
- Name: represents an entry point in that function (e.g. entry or return), or has a meaningful name (e.g. io:::start, proc:::exec)

Probes 2

- List probes
  - Use dtrace(1M) and '-l' option
  - For each probe the four-tuple will be displayed, probe components are ':' separated
  - List all probes:

```
$ dtrace -l | wc -l
39570
```

  - List all probes offered by syscall provider:

```
$ dtrace -lP syscall
```

  - List all probes offered by the ufs module:

```
$ dtrace -lm ufs
```

  - List all providers:

```
$ dtrace -l | awk '{print $2}' | sort -u
```

Probes 3

- List all read function probes:

```
$ dtrace -l -f read
```

- Enabling probes
  - Activate a probe by not using '-l' option
  - Default action with enabled probes- the CPU, the probe number and name are displayed whenever the probe fires
  - Enable all probes from nfs and ufs module:

```
$ dtrace -m nfs,ufs
```

  - Enable all read function probes:

```
$ dtrace -f read
```

  - Enable all probes from io provider:

```
$ dtrace -P io
```

Providers I

- A methodology for instrumenting the system
- Makes available all know probes
- Providers are offering all probes to the DTrace framework
- DTrace framework confirms to providers when a probe is activated
- Providers pass the control to DTrace when a probe is enabled
- Example of certain providers: syscall, lockstat, fbt, io, mib

Providers 2

- **syscall**
  - one of the most important providers
  - holds the entire communication from userland to kernel space
  - every system call on the system
- **proc**
  - handles: process, LWP creation and termination, signaling
- **sched**
  - CPU scheduling: why threads are sleeping, running
  - used usually to compute the CPU time, which threads are run by which CPU and for how long

Providers 3

- **io**
  - disk input and output requests
  - I/O by device, process, size, filename
- **mib**
  - counters for management information bases
  - IP, IPv6, ICMP, IPSec
- **profile**
  - time based probing at specific interval of times
  - low overhead
  - profile-<interval> and tick-<interval>

Actions I

- Taken when a probe fires
- Indicated by following a probe specification with "{ *action* }"
- Used to record data to a DTrace buffer
- Different types of actions:
  - data recording
  - destructive
  - special
- By default, data recording actions record data to the principal DTrace buffer

Actions 2

- Data Recording Actions
  - **trace(expression)**
    - records the result of trace to the directed buffer
    - `trace(pid)` traces the current process id
    - `trace(execname)` traces the current application name
  - **printf()**
    - traces a D expression
    - allows output style formatting
    - `printf("execname is %s", execname);`
  - **printa(aggregation)**
    - used to display and format aggregations
    - `printa(@agg1)`

Actions 3

- Data Recording Actions
  - **stack()**
    records a kernel stack trace
    dtrace -n 'syscall::open:entry{stack();}'
  - **ustack()**
    records a user process stack trace
    allows to inspect userland stack processes
    dtrace -n 'syscall::open:entry{ustack();}' -c ls
  - **jstack()**
    similar with ustack(), used for Java
    The stack depth frames is different than in ustack

Actions 4 (examples)

- Examples
  - `syscall:::`
  - `syscall:::entry`
  - `syscall:::return`
  - `syscall::read:entry{ printf("Process %d", pid); }`
  - `syscall::write:entry/execname=="firefox-bin"/{ @[probefunc] = count(); }`
  - `sysinfo:::readch{ trace(execname); exit(0); }`
  - `sysinfo:::writech`
  - `io:::`

Predicates I

- D expressions that define a conditional test
- Allow actions to only be taken when certain conditions are met. A predicate has this form: */predicate/*
- The actions will be activated only if the value of the predicate expression is true
- Used to filter and meet certain conditions: look only for a process which has the pid = 1203, match a process which has the name firefox-bin

Predicates 2

- Examples
  - `syscall:::`
  - `syscall:::entry`
  - `syscall:::return`
  - `syscall::read:entry{ printf("Process %d", pid); }`
  - `syscall::write:entry/execname=="firefox-bin"/{ @[probefunc] = count(); }`
  - `sysinfo:::readch{ trace(execname); exit(0); }`
  - `sysinfo:::writech`
  - `io:::`

Aggregations I

- Used to aggregate data and look for trends
- Simple to generate reports about: total system calls used by a process or an application, the total number of read or writes by process...
- Has the general form:

  @name[keys] = aggfunc(args)
- There is no need to use other tools like: awk(1), perl(1)

Aggregations 2

- Aggregating functions
  - count() : the number of times called, used to count for instance the total number of reads or system calls
  - sum() : the total value of the specified expressions
  - avg() : the arithmetic average of the specified expression
  - min() : the smallest value of the specified expression
  - max(): the largest value of the specified expression
  - quantize() : a power-of-two frequency distribution, simple to use to draw distributions
- Non-aggregating functions:mode and median

Aggregations 3

- What's going on with my system ?

```
dtrace -n syscall::::entry
```

- Difficult to read, start aggregating...

```
dtrace -n 'syscall::::entry{@[execname] =
  count();}'
```

- Filter on read system call

```
dtrace -n
  'syscall::read*:entry{@[execname]=count();}
  '
```

- Add the file descriptor information

```
dtrace -n
  'syscall::read*:entry{@[execname,arg0]=coun
  t();}'
```

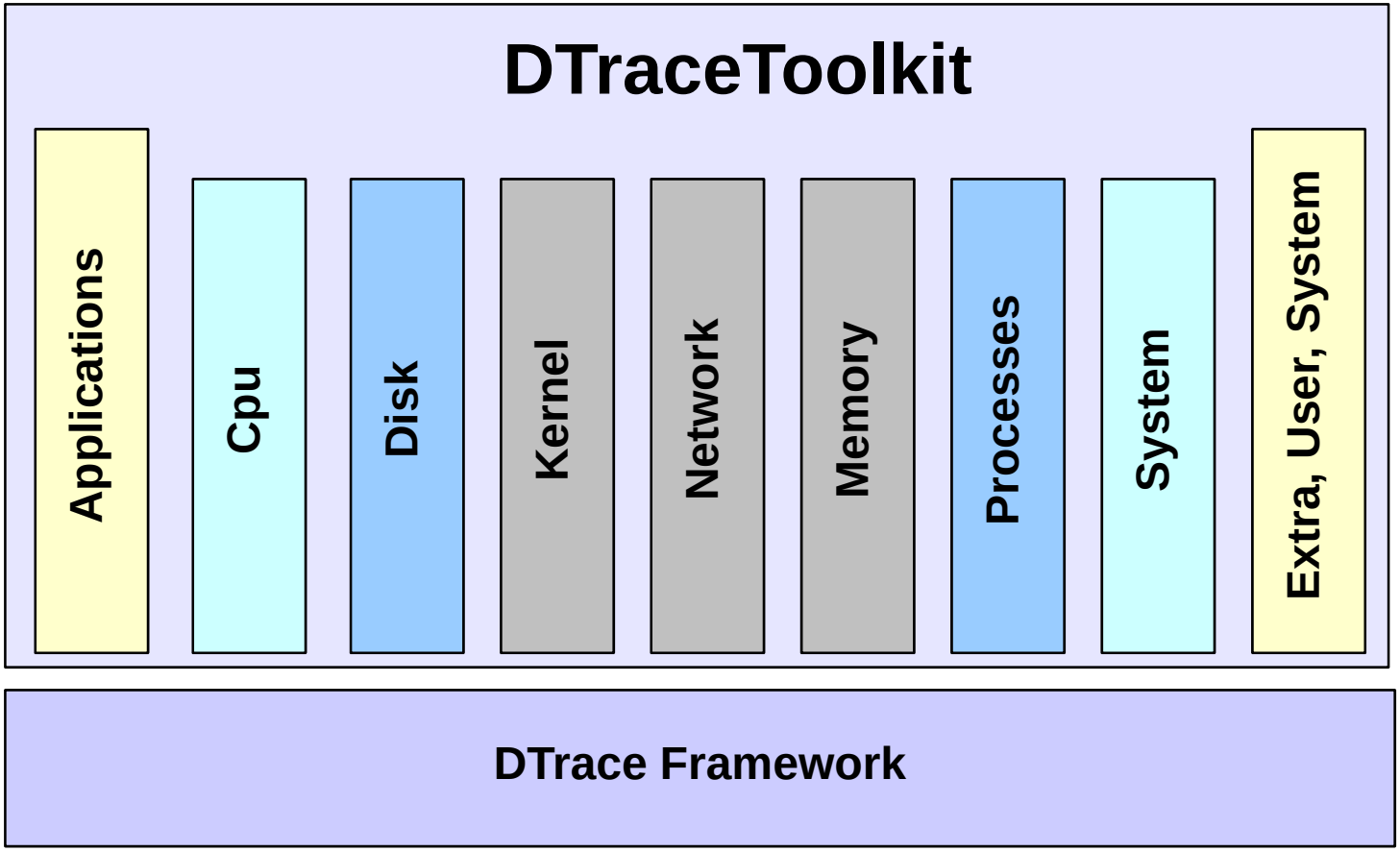
Aggregations 4

- Drill-down and get a distribution of each read by application name:

```
syscall::read*:entry
{
 self ->ts=timestamp;
}

syscall::read*:return
/self -> ts/
{
 @time[execname] = quantize(timestamp – self->ts);
 self->ts = 0;
}
```

Dtrace Toolkit



Toolkit Examples I

- **Disk**
  - Analyses I/O activity using the io provider from DTrace: disk I/O patterns, disk I/O activity by process, the seek size of an I/O operation
  - **iotop**: a top like utility which lists disk I/O events by processes
  - **iosnoop**: a disk I/O trace event application. The utility will report UID, PID, filename regarding for a I/O operation
  - **bitesize.d**: analyse disk I/O size by process
  - **seeksize.d**: analyses the disk I/O seek size by identifying what sort I/O operation the process is making: sequential or random

Toolkit Examples 2

- **Disk**
  - **iofile.d**: prints the total I/O wait times. Used to debug applications which are waiting for a disk file or resource
  - **iopattern**: computes the percentage of events that were of a random or sequential nature. Used easily to identify the type of an I/O operation and the average, totals numbers
  - **iopending**: prints a plot for the number of pending disk I/O events. This utility tries to identify the "serialness" or "parallelness" of the disk behavior
  - **diskhits**: prints the load average, similar to uptime
  - **iofileb.d**: prints a summary of requested disk activity by pathname, providing totals of the I/O events in bytes

Toolkit Examples 3

- **FS**
  - Analyses the activity on the file system level: write cache miss, read file I/O statistics, system calls read/write
  - **vopstat**: traces the vnode activity
  - **rfsio.d**: provides statistics on the number of reads: the bytes read from file systems (logical reads) and the number of bytes read from physical disk
  - **fspaging.d**: used to examine the behavior of each I/O layer, from the syscall interface to what the disk is doing
  - **rfileio.d**: similar with rfsio.d but reports by file

scsi.d

- The child of Chris Gerhard:
  - http://blogs.sun.com/chrisg/tags/scsi.d
  - 422 lines without the copyright header: when dtracers go mad!
- Touchstone for serious use.

Other Providers & Probes

- The providers of interest to storage folk (iterate).

Further work

- More providers
- The network stack (asynchronous events)
- Better documentation
- More scripts

Where to go from here:

- http://www.opensolaris.org/os/community/dtrace/
- http://www.sun.com/bigadmin/content/dtrace/
- Discussion Group:
  - http://www.opensolaris.org/os/community/dtrace/discussions/
- Book: Solaris Performance & Tools
- Book: Solaris Internals 2nd Ed.
- See Generally: http://www.solarisinternals.com/

Outro

- Q & A

DTrace for Storage Development

Dominic Kay, Sun Microsystems

dominic.kay@sun.com

blogs.sun.com/dom