DTrace

Christopher Beal
Senior Staff Engineer
Sun Microsystems
http://opensolaris.org

Agenda

- **Why DTrace**
- What is DTrace
- Providers Probes and Stuff
- Using Dtrace – Some examples
- Dtrace resources

Why DTrace?

- Transient problems are hard to debug.
- Example.
  - Who sent a kill signal to my process
  - Thread gets preempted when it should not
  - In live production system my application does not scale above 30,000 user
  - Why are there so many threads in run queue when the CPU is idle

Current Options

- Reproduce problem outside of production
  - Not easy & Expensive
- Convert it into a fatal problem
  - Causes more downtime than the transient problem
  - Not easy to debug a transient problem with a snapshot
- Use tools like truss or pstack
  - Per process tools – hard to debug systemic issues
  - Too intrusive for production

Current Options

- Custom instrumented application or kernel
  - Too intrusive for production
  - Takes too many iteration to get to the root cause
  - Huge QA cost
  - Expensive production interruptions

A Much Better Solution

- **A Dynamically Instrumentable System**
  - have enough instrumentation to permit collecting any arbitrary data
  - permit dynamically turning on/off instrumentation
  - be performant to run in production
  - ensures safety

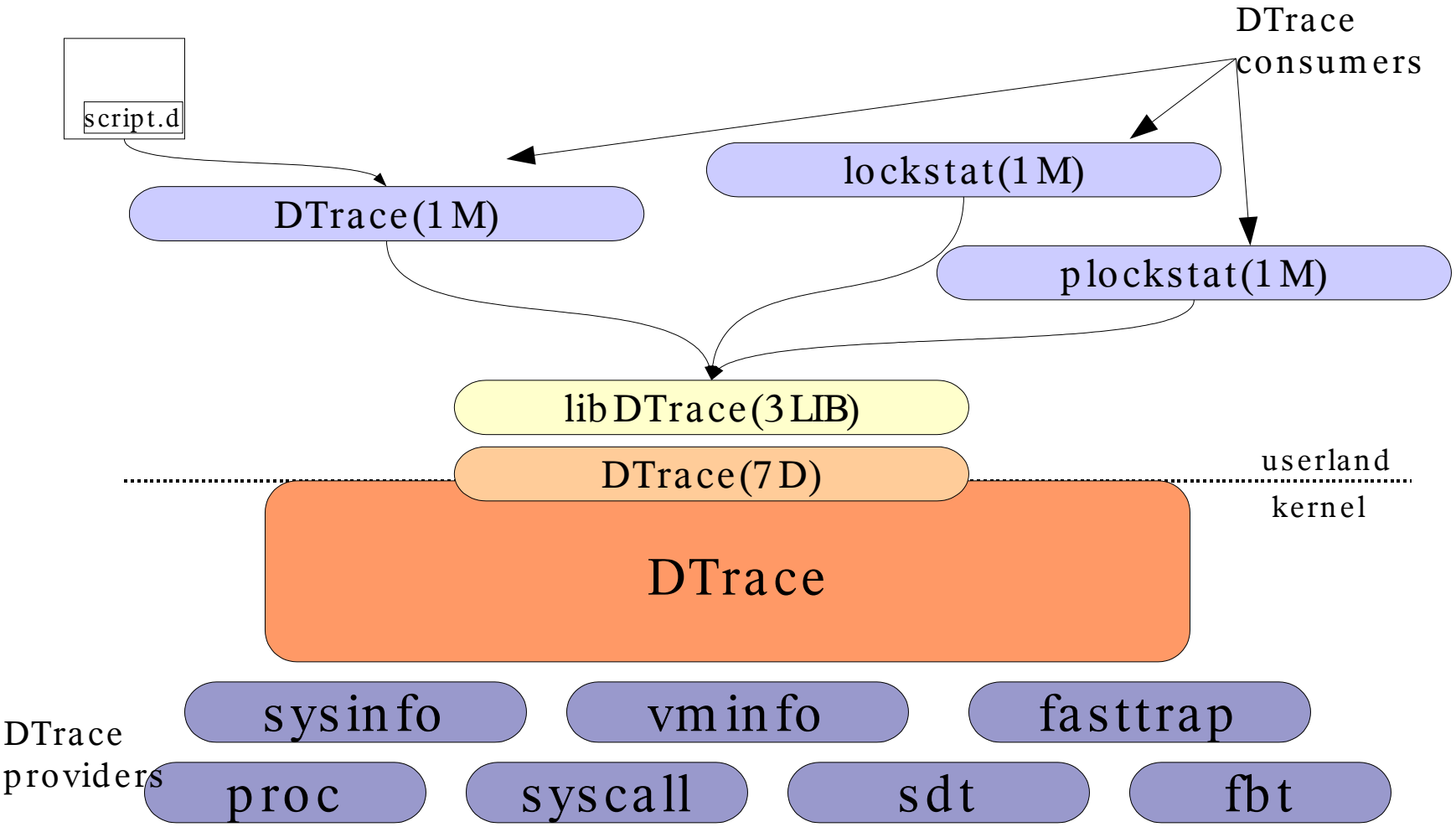
Agenda

- Why DTrace
- **What is DTrace**
- Providers Probes and Stuff
- Using Dtrace – Some examples
- Dtrace resources

Dtrace – What is it?

- Over 30K probes built into Solaris 10
- Can create more probes on the fly
- New powerful, dynamically interpreted language (D) to instantiate probes
- Probes are light weight and low overhead
- **No** overhead if probe not enabled
- Safe to use on **“live”** system

Dtrace Architecture



How it works

- Driven through the D Language
  - dtrace command compiles the D language Script
  - Intermediate code checked for safety (like java)
  - The compiled code is executed in the kernel by DTrace
  - DTrace instructs the provider to enable the probes
- As soon as the D program exits all instrumentation removed
- No limit (except system resources) on number of D scripts that can be run simultaneously
- Different users can debug the system simultaneously without causing data corruption or collision issues

D Language - Format.

```
probe description
/
        predicate
/
{
        action statements
}
```

- When a probe fires then action is executed if predicate evaluates true

D Language - Example

- Example:
  - > “Print all the system calls executed by ksh”

syscalls.d

```
#!/usr/sbin/dtrace -s
syscall:::entry
/
        execname==”ksh”
/
{
        printf(“%s called\n”,probefunc);
}
```

Agenda

- Why DTrace
- What is DTrace
- **Providers Probes and Stuff**
- Using Dtrace – Some examples
- Dtrace resources

Providers

- Providers make probes available to the DTrace framework
- DTrace controls when a provider enables a probe
- Providers transfer control to DTrace when a probe is fired

Some Available Providers

- syscall: provides probes in every system call
- fbt: provides probe into every function in the kernel
- pid: Provides fbt and instruction tracing for user programs
- proc: probes relating to process lifecycle
- profile: probes that will fire at regular intervals
- sdt: static probe points for your programs
- lockstat: probes to look at lock contention events

Probe

- Probes are points of **instrumentation**
- Each probe has a name
- These four attributes define a tuple that uniquely identifies each probe
  - *provider:module:function:name*
  - Example

    syscall::open:entry
- *Listed by dtrace -l*
  - *eg. dtrace -l -P proc*

Predicates

- A predicate is a D expression
  - Like a D if statement
- Actions will only be executed if the predicate expression evaluates to true
- *Examples*
  - Print the pid of every “ls” process that is started

```
#!/usr/sbin/dtrace -s
proc:::exec-success
/execname == "ls"/
{
}
```

pred.d

Actions

- Actions are executed when a probe fires
- Most actions record some specified state in the system
- Some actions change the state of the system in a well-defined manner
  - These are called destructive actions and are disabled by default.
- Probes may provide parameters that can be used in the actions.

Variables and Operators

- All standard 'C' types and operators available
- External variables are available using the ` symbol
  - eg. `physmem – is the kernel physmem variable
- Many builtin variables for you to use
  - eg. pid, arg0, errno
- Thread local data
  - self->variable = expression;
  - Essential for multi-threaded debugging

Aggregation

- Think of a case when you want to know the "**total**" time the system spends in a function.
  - We can save the amount of time spent by the function every time it is called and then add the total.
    - If the function was called 1000 times that is 1000 bits of info stored in the buffer just for us to finally add to get the total.
  - Instead if we just keep a running **total** then it is just one piece of info that is stored in the buffer.
  - We can use the same concept when we want **averages**, **count**, **min** or **max**.
- **Aggregation** is a D construct for this purpose.

Aggregation - Format

- **@name[keys] = aggfunc(args);**
- '@' - key to show that name is an aggregation.
- keys – comma separated list of D expressions.
- **aggfunc** could be one of...
  - sum(expr) – total value of specified expression
  - count() – number of times called.
  - avg(expr) – average of expression
  - min(expr)/max(expr) – min and max of expressions
  - quantize()/lquantize() - power of two & linear distribution

Aggregation Example

aggr.d

```
#!/usr/sbin/dtrace -s
pid$target:libc:malloc:entry
{
        @["Malloc Distribution"]=quantize(arg0);
}
```

```
  $ aggr.d -c who
  dtrace: script './aggr.d' matched 1 probe
  ...
  dtrace: pid 6906 has exited

    Malloc Distribution
           value  ------------- Distribution -------------------------------------------------count
               1 |                                                                               0
               2 |@@@@@@@@@@@@@@@@@@@                                                            3
               4 |                                                                               0
               8 |@@@@@@                                                                         1
              16 |@@@@@@                                                                         1
              32 |                                                                               0
              64 |                                                                               0
             128 |                                                                               0
             256 |                                                                               0
             512 |                                                                               0
            1024 |                                                                               0
            2048 |                                                                               0
            4096 |                                                                               0
            8192 |@@@@@@@@@@@@                                                                   2
           16384 |                                                                               0
```

DTrace and User Process

- DTrace provides a lot of features to investigate user processes
- We will look at features in DTrace that is useful to examine user process
- Uses the PID provider

Granting privilege to run DTrace

- A system admin can grant any user privileges to run DTrace using the Solaris Least Privilege facility privileges(5).
- DTrace provides for three types of privileges.

  dtrace_proc - provides access to process level tracing no kernel level tracing allowed. (pid provider is about all they can run)

  dtrace_user – provides access to process level and kernel level probes but only for process to which the user has access. (ie) they can use syscall provider but only for syscalls made by process that they have access.

  dtrace_kernel – provides all access except process access to user level procs that they do not have access.

- Enable these priv by editing **/etc/user_attr**. Eg
  - user::::defaultpriv=basic,privileges,dtrace_proc,dtrace_user,dtrace_kernel

Agenda

- Why DTrace
- What is DTrace
- Providers Probes and Stuff
- **Using Dtrace – Some examples**
- Dtrace resources

The pid Provider

- The **pid** Provider is extremely flexible and allows you to instrument any instruction in **user land** including entry and exit
- pid provider creates probes on the fly when they are needed
- This is why they **do not** appear in the dtrace -l listing
- We will see how to use the pid provider to trace
  - Function Boundaries
  - Any arbitrary instruction in a given function

pid – Function Boundary probes

- The probe is constructed using the following format

  pid<processid>:<library or executable>:<function>:<entry or return>

- Examples:

  pid1234:date:main:entry

  pid1122:libc:open:return

- Following code counts all libc calls made by a program

```
#!/usr/sbin/dtrace -s
pid$target:libc::entry
{
        @[probefunc]=count()
}
```

pid1.d

pid – Function Offset probes

- The probe is constructed using the following format

  pid<processid>:<library or executable>:<function>:<offset>

- Examples:

  pid1234:date:main:*16*

  pid1122:libc:open:*4*

- Following code prints all instructions executed in the programs main

  offs.d

```
#!/usr/sbin/dtrace -s
pid$target:a.out:main:
{
}
```

pid – Instruction Level Tracing

- The function offset tracing is a very powerful mechanism.
- The following example prints code path followed by a particular func.

trace_code.d

```
pid$1::$2:entry
{
        self->trace_code = 1;
        printf("%x %x %x %x %x", arg0, arg1, arg2, arg3, arg4);
}
pid$1:::
/self->trace_code/
{ }

pid$1::$2:return
/self->trace_code/
{
        exit(0);
}
```

Execute.

trace_code.d 1218 printf

proc Provider

- The proc Provider has probes for **process/lwp lifecycle**
  - **create** – fires when a proc is created using fork and its variants
  - **exec** – fires when exec and its variants are called
  - **exec-failure** & **exec-success** – when exec fails or succeeds
  - **lwp-create**, **lwp-start**, **lwp-exit** – lwp life cycle probes
  - **signal-send**, **signal-handle**, **signal-clear** – probes for various signal states
  - **start** – fires when a process starts before the first instruction is executed.

Who killed the process?

sig1.d

```
#!/usr/sbin/dtrace -qs

proc:::signal-send
/args[1]->pr_fname == $$1/
{
        printf("%s(pid:%d) is sending signal %d to %s\n", execname, pid, args[2],args[1]->pr_fname);
}
```

```
$ ./sig1.d bc
sched(pid:0) is sending signal 24 to bc
sched(pid:0) is sending signal 24 to bc
bash(pid:3987) is sending signal 15 to bc
bash(pid:3987) is sending signal 15 to bc
bash(pid:3987) is sending signal 9 to bc
```

The above program prints out process that is sending the signal to the program “bc”.

Note: $$1 is argument 1 as string

The signal-send probe has arg1 that has info on signal destination

The signal-send probe has args2 that has the signal number

Agenda

- Why DTrace
- What is DTrace
- Providers Probes and Stuff
- Using Dtrace – Some examples
- **Dtrace resources**

DTrace Resources

- Here are a few of the many DTrace resources available for you
  - "Solaris Dynamic Tracing Guide" is an excellent resource. Many of the examples in this presentation are from the Guide. http://docs.sun.com/db/doc/817-6223
  - The BigAdmin DTrace web page has a lot of good info http://www.sun.com/bigadmin/content/dtrace/
  - Open Solaris DTrace community page http://www.opensolaris.org/os/community/dtrace/
  - DTrace toolkit contains a lot of very useful scripts http://www.opensolaris.org/os/community/dtrace/dtrace

More DTrace Resources

- Read the Blogs from Bryan Cantrill, Adam Leventhal, Mike Shapiro

  http://blogs.sun.com/roller/page/bmc

  http://blogs.sun.com/roller/page/ahl

  http://blogs.sun.com/mws

  > They often speak about DTrace related issues.

- Of course you can google DTrace.

  http://www.google.com/search?q=dtrace

Thank you!

Christopher Beal
Senior Staff Engineer
Sun Microsystems
http://opensolaris.org