

DTrace & MySQL

MySQL Users Conference 2008

Ben Rockwood
Director of Systems
Joyent

DTrace

- Dynamic Tracing Framework
- The ultimate observability tool.
- Created by Sun for Solaris 10
- Open Source! (CDDL License)
- Ported to OS X (10.5), FreeBSD, and QNX

Advantages

- You don't have to login to the database
- Not limited to data provided by SHOW ...;
- Ability to observe and profile discrete actions down to the function level
- Observe interaction between dependancies (Apache/PHP/MySQL, etc.) or system (I/O, etc)
- Queries don't slip through the cracks due to a polling interval (processlist, sleep, processlist,..)
- ... and much more.

Basics

- Tens of thousands of “probes” embedded into the kernel
- Organized as: `provider:module:function:name`
- Example: `syscall::write:entry`
- When probe “fires” an action can conditionally (predicates) be run
- D language (“D Script”) borrows from C & AWK
- Zero impact when probe not enabled;
Production safe!

Probes

- List all probes via ‘dtrace -l’
- Most probes include at least an “entry” and “return” (or “start”/”done”, etc.)

```
root@ultra ~$ dtrace -l | wc -l  
127574
```

```
root@ultra ~$ dtrace -l
```

ID	PROVIDER	MODULE	FUNCTION NAME
1	dtrace		BEGIN
2	dtrace		END
3	dtrace		ERROR
4	fbt	smbios	smb_info entry
5	fbt	smbios	smb_info return

Example: Syscall Provider

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

```
syscall::entry  
{  
  
}
```

```
$ sudo ./syscall-simple.d  
dtrace: script './syscall-simple.d' matched 427 probes
```

CPU	ID	FUNCTION:NAME
I	17702	ioctl:entry
I	17702	ioctl:entry
I	17998	__sysctl:entry
I	17998	__sysctl:entry
I	17686	sigaction:entry
I	17686	sigaction:entry
I	17686	sigaction:entry
I	17686	sigaction:entry
I	17690	sigprocmask:entry

Syscall Part 2: Variables

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

```
syscall::entry  
/ execname != "dtrace" /  
{  
    trace(execname);  
}
```

```
$ sudo ./syscall-simple2.d
```

```
dtrace: script './syscall-simple2.d' matched 427 probes
```

CPU	ID	FUNCTION:NAME
0	17988	mmap:entry iTerm
0	17740	munmap:entry iTerm
0	18320	kevent:entry Stickies
0	17974	lstat:entry fseventsd
0	17974	lstat:entry fseventsd
0	17974	lstat:entry fseventsd
0	17974	lstat:entry fseventsd
0	18034	getattrlist:entry Stickies

Syscall Part 3: Aggregations

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

```
syscall::entry  
/ execname != "dtrace" /  
{  
    @foo[probefunc] = count();  
}
```

```
$ sudo ./syscall-simple3.d  
dtrace: script './syscall-simple3.d' matched 427 probes  
^C
```

__disable_threadsignal	1
accept	1
access	1
...	
sigaltstack	73
sigprocmask	82
close	2574

Syscall Part 4: Drilling Down

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

```
syscall::close:entry  
/ execname != "dtrace" /  
{  
    @foo[execname] = count();  
}
```

```
$ sudo ./syscall-simple4.d  
dtrace: script './syscall-simple4.d' matched 1 probe  
^C
```

syslogd	4
airport	7
iStumbler	2565

Syscall Part 4: ustack

```
$ sudo ./syscall-simple5.d
```

```
dtrace: script './syscall-simple5.d' matched 1 probe
```

```
^C
```

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

```
syscall::close:entry
/ execname == "iStumbler" /
{
    @close[ustack()] = count();
}
```

```
libSystem.B.dylib`close$UNIX2003+0xa
Foundation`-[NSConcretePipe dealloc]+0x3f
CoreFoundation`CFRelease+0x5a
CoreFoundation`__CFDictionaryDeallocate+0x119
CoreFoundation`_CFRelease+0xd8
Foundation`-[NSConcreteTask dealloc]+0x26
CoreFoundation`CFRelease+0x5a
Foundation`_taskDied+0xd9
CoreFoundation`__CFFileDescriptorPerform+0x55
CoreFoundation`CFRunLoopRunSpecific+0xf5
CoreFoundation`CFRunLoopRunInMode+0x58
Foundation`-[NSRunLoop(NSRunLoop) runMode:beforeFirst
Foundation`-[NSRunLoop(NSRunLoop) run]+0x54
RadioStore`-[RSScanner run:]+0x143
Foundation`-[NSThread main]+0x2d
Foundation`__NSThread__main__+0x134
libSystem.B.dylib`_pthread_start+0x141
libSystem.B.dylib`thread_start+0x22
```

|



Free Your Mind.

Statically Defined User Probes

- Probes can be explicitly embedded in your code
- Static probes improves user experience by providing them with a pre-defined listing of relevant probe points
- Simple to do:
 - Create .d source file defining your probes
 - Inserted `DTRACE_PROBEn(arg0, arg1,...);` macro at desired points
 - Generate a header file (`dtrace -h..`)
 - Recompile, running '`dtrace -G`' prior to final link.

The PID Provider

- PID Provider allows tracing of entry and return of functions within user applications (“Function Boundary Tracing”)
- NO MODIFICATIONS REQUIRED, PERIOD!
- Probe format example:
 - pid123:a.out:some_function:entry
 - pid123:mysql:some_function:entry
 - pid123:::entry
 - pid123::*write*:return

MySQL Internals... Buy it.

- PID provider allows us to peer into the entry or return of any function inside MySQL (or anything else), any version, no mods.
- .. you do know what all those functions do right?
- Copies of “MySQL Internals” (O’Reilly) or “Expert MySQL” (Apress) recommended.

Watching Queries

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

```
#pragma D option quiet
```

```
pid$target:*dispatch_command*:entry  
{  
    printf("Query: %s\n", copyinstr(arg2));  
}
```

```
$ ./query_watch.d -p `pgrep -x mysqld`
```

```
Query: show tables
```

```
Query: select * from Country LIMIT 10
```

```
Query: explain user
```

```
Query: select Host,User>Password,Select_priv from user where User = 'benr'
```

```
Query: blah blah blah....
```

Counting Queries

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

```
#pragma D option quiet
```

```
dtrace:::BEGIN
```

```
{  
    printf("Tracing... Hit Ctrl-C to end.\n");  
}
```

```
pid$target::*mysql_parse*:entry
```

```
{  
    @query[copyinstr(arg1)] = count();  
}
```

```
$ ./querycounts.d -p `pgrep -x mysqld`  
Tracing... Hit Ctrl-C to end.  
^C
```

```
select * from CountryLanguage LIMIT 5  
show tables  
select * from City
```

```
1  
1  
2
```

Queries, Qcache, & Time

```
pid$target:mysql:*dispatch_command*:entry
{
    self->query = copyinstr(arg2);
    self->start = timestamp;
}
```

```
pid$target:mysql:*send_result_to_client*:entry
{
    self->cache = "Yes";
}
```

```
pid$target:mysql:*do_select*:entry
{
    self->cache = "No";
}
```

```
pid$target:mysql:*dispatch_command*:return
{
    this->end = timestamp;
    this->elapsed = (this->end - self->start)/1000000;
    printf("From Query Cache?: %s in %d ms \t| Query: %s\n", self->cache, this->elapsed, self->query);
    self->start = 0;
    self->query = 0;
}
```

Queries, Qcache & Time: Output

```
$ ./my_qcache.d -p `pgrep -x mysqld`  
Waiting for queries... hit ^c to quit.
```

```
From Query Cache?: No in 1 ms | Query: SELECT * FROM Country  
From Query Cache?: Yes in 0 ms | Query: SELECT * FROM Country  
From Query Cache?: No in 48 ms | Query: select * from City where CountryCode = 'USA'  
From Query Cache?: Yes in 0 ms | Query: select * from City where CountryCode = 'USA'  
From Query Cache?: No in 16 ms | Query: select * from City where CountryCode LIKE 'USA%'  
^C
```

Watching MySQL Flow

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

```
#pragma D option flowindent
```

```
pid$target:mysqld::entry
```

```
{
```

```
pid$target:mysqld::return
```

```
}
```

```
| -> _Z34create_thread_to_handle_connectionP3THD
```

```
| -> my_micro_time
```

```
| <- my_micro_time
```

```
| <- _Z34create_thread_to_handle_connectionP3THD
```

```
| -> handle_one_connection
```

```
| -> my_micro_time
```

```
| <- my_micro_time
```

```
| -> _Z34init_new_connection_handler_threadv
```

```
| -> my_thread_init
```

```
| <- my_thread_init
```

```
| <- _Z34init_new_connection_handler_threadv
```

```
| -> _Z31setup_connection_thread_globalsP3THD
```

```
| -> _ZN3THDI3store_globalsEv
```

```
| -> _my_thread_var
```

```
| <- _my_thread_var
```

```
| -> thr_lock_info_init
```

```
| -> _my_thread_var
```

```
| <- _my_thread_var
```

```
| <- thr_lock_info_init
```

```
| <- _ZN3THDI3store_globalsEv
```

```
| <- _Z31setup_connection_thread_globalsP3THD
```

```
| -> _Z9lex_startP3THD
```

```
| -> _ZN18st_select_lex_unit10init_queryEv
```

```
| -> _ZN18st_select_lex_node10init_queryEv
```

Broadening the Scope

- DTrace and see the whole system at the same time allowing for correlations, such as...
- Trace physical IO, paging, etc, as a result of some action in the database
- Trace Firefox, Apache, PHP, and MySQL together to profile end-to-end. (On one system obviously.)

Static Probes in MySQL & THD

- PID provider is awesome, but can't look inside THD which can be limiting (May be possible, but no ones done it yet...)
- Tracing requires a knowledge of what your tracing... choosing which function to trace can be daunting.
- Static probes in MySQL would give us visibility into interesting aspects of THD where appropriate and make tracing much more straight forward.

Status of Probe Integration

- Sun frequently demo's probes in MySQL 5.0 but has not released a patch. They are not present in any release from anyone (Coolstack, etc.)
- Probes currently in the 6.0 tree, but... only a handful of probes and no arguments passed, making them effectively useless for anything but aggregate counting.
- Probes currently in Falcon, will be coming to more engines soon.

Resources

- DTrace Toolkit (System) by Brendan Gregg
- MySQL DTrace Toolkit (MySQL, PID) by Derek Crudgington
- “Solaris Dynamic Tracing Guide” (docs.sun.com)
- DTrace “Google Talk” (youtube)
- OpenSolaris DTrace Community (opensolaris.org)
- Search for “DTrace MySQL”

Thank You.

Ben Rockwood

Joyent, Inc.

benr@joyent.com

<http://www.cuddletech.com/>