SystemTap/DTrace with MySQL & Drizzle

Padraig O'Sullivan
Software Engineer, Akiban Tech.
posullivan@akiban.com
http://posulliv.github.com/

About Padraig

- Software Engineer at Akiban Technologies in Boston
- Drizzle contributor for over a year now
- Ported the DTrace probes from MySQL to Drizzle...hence this talk!
- Actually going to spend most of this talk discussing SystemTap since I'm assuming everyone is familiar with DTrace

What is DTrace?

- Observability technology that allows the behaviour of systems and applications to be investigated easily
- Developed at Sun and runs on Solaris and OSX
- Zero probe effect when not enabled and system acts as if DTrace not present at all
- Has a scripting language called “D”
- It doesn't work on Linux currently

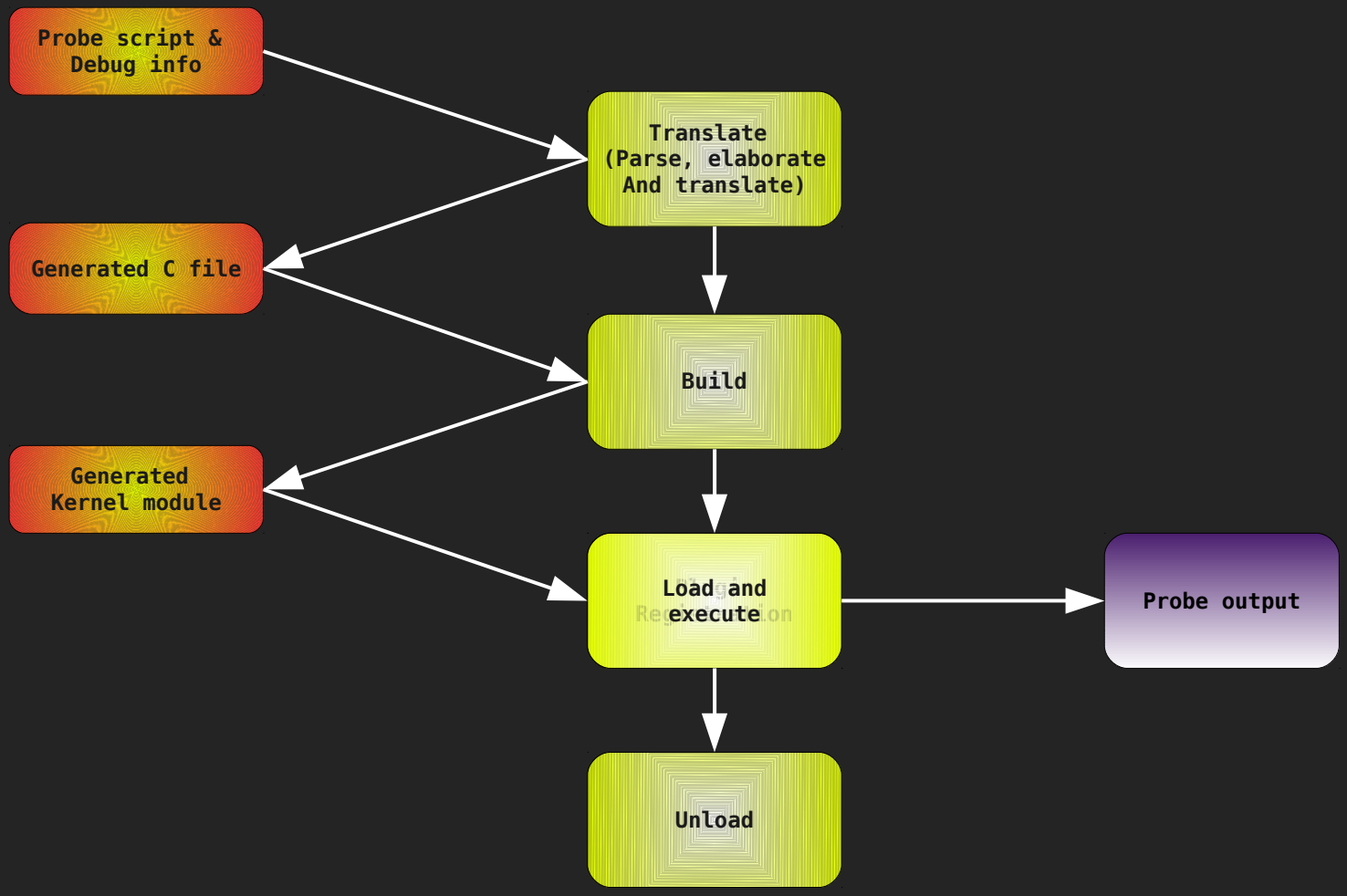
What is SystemTap?

- Project under development since 2005 for dynamic instrumentation of Linux systems
  - Contributions come from Red Hat, IBM, Oracle, etc.
- SystemTap can accomplish the same tasks as DTrace
- Static DTrace probes in an application can be used with SystemTap
  - Latest version comes with python script named dtrace for this
- Has production support since RHEL 5.4

How SystemTap Works

- SystemTap has a command-line interface named stap
  - Takes script in C-like language
- Basic steps Systemtap performs are:
  - Translates the .stp script into C source code
  - Compiles the C code into a kernel module (.ko file)
  - Loads the module into the kernel
  - Module runs
  - Module is unloaded when CTRL-C is given or module decides it is done

How SystemTap Works



Probes

- Essential idea behind a SystemTap script is to name “events” and give them “handlers”
- When a specified event occurs, the kernel runs the handler and then resumes execution
- Combination of an event and a handler is referred to as a probe in SystemTap

Simple Example – syscall probing

```
# stap -e 'probe syscall.open
> {
>   log(execname() . “: “ . filename)
> }'
```

```
bash: /
bash: /tmp
cat: /etc/ld.so.cache
cat: /tmp/padraig
sendmail: /proc/loadavg
cat: <unknown>
...
```

iotop in SystemTap

```
global reads, writes, total_io

probe vfs.read
{
    reads[execname()] += $count
}

probe vfs.write
{
    writes[execname()] += $count
}

# print top 10 IO processes every 5 seconds
probe timer.s(5)
{
    foreach (name in writes)
        total_io[name] += writes[name]
    foreach (name in reads)
        total_io[name] += reads[name]
    printf ("%16s\t%10s\t%10s\n", "Process", "KB Read", "KB Written")
    foreach (name in total_io- limit 10)
        printf("%16s\t%10d\t%10d\n", name,
                reads[name]/1024, writes[name]/1024)
    delete reads
    delete writes
    delete total_io
    print("\n")
}
```

iotop in SystemTap

```
# stap iotop.sty

       Process  KB Read KB Written
       firefox     3200          0
Gnome-terminal     3060          0
         mysql      528         22
        mysqld       23         63
hald-addon-inpu      14          0
gnome-power-man      12          0

#
```

Getting SystemTap

- Recommend going with Fedora 11/12 or RHEL if you want to use SystemTap easily. What's needed:
  - systemtap
  - systemtap-sdt-devel
  - systemtap-runtime
  - Kernel debuginfo
- Quite stable on these platforms and usually works just as advertised

SystemTap on Ubuntu/Debian

- This is not as easy! What's needed:
  - systemtap
  - systemtap-sdt-dev
  - A kernel compiled with debug information
  - A kernel that supports utrace if you want to perform user level probing
- In order to perform user-level probing, you will need to compile/install a custom kernel that supports utrace

Dtrace/SystemTap Comparison

- Obviously there is the licensing difference but we won't discuss that here
- Both tools are said to be safe to use in production
- DTrace comes by default in Solaris. Not so with SystemTap on Linux distributions
- From a user's perspective, SystemTap can be a pain to get working (particularly on Ubuntu/Debian)
- DTrace just works straight out of the box

MySQL & Drizzle with SystemTap

- Symbolic probing in which function names can be used for probes:
  - Relies on knowledge of the source code of the application
- Instrument the application with marks
  - Only requires us to know what the markers for the application are
- Luckily, MySQL and Drizzle have static Dtrace probes which correspond to the second method listed above

MySQL & Drizzle with SystemTap

- MySQL comes with static DTrace probes
  - I ported these probes to drizzle
- SystemTap is compatible with static DTrace probes
  - Any probe we define in our application for Dtrace can be used by SystemTap also
- In order to build MySQL and Drizzle with SystemTap support the systemtap-sdt-dev package needs to be installed

MySQL & Drizzle with SystemTap

- To build MySQL and Drizzle for use with SystemTap, it's pretty straightforward:

Drizzle

```
$ ./config/autorun.sh
$ ./configure –with-debug –enable-dtrace
$ make
```

MySQL

```
$ ./BUILD/autorun.sh
$ ./configure –with-debug –enable-dtrace
$ make
```

Static Probes in MySQL & Drizzle

- Once MySQL/Drizzle is configured and built and we start the server, we can list the markers that are present in the server:

```
# stap -l 'process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("*")'
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("net__write__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("net__write__done")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("net__read__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("net__read__done")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("connection__done")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("connection__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("query__parse__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("query__parse__done")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("update__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("update__done")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("multi__update__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("multi__update__done")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("insert__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("insert__done")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("insert__select__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("insert__select__done")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("delete__start")
process("/home/posulliv/repos/mysql/uc/sql/mysqld").mark("delete__done")
...
```

MySQL & Drizzle Static Probes

- Approximately 60 probes, 200 locations in MySQL
- Query cache (not in drizzle)
- MyISAM key cache (not in drizzle)
- Network I/O
- Query execution
- Drizzle has started inserting probes to the optimizer
- Zero overhead when probes disabled
  - Overhead can get high if lots of probes firing

Measuring Time in Storage Engine

```
global query_times
global engine_times
global queries

probe process(@1).mark("query__start")
{
  query_times[tid()] = gettimeofday_us()
  engine_times[tid()] = 0
  queries[tid()] = user_string($arg1)
}

probe process(@1).mark("*row__start")
{
  engine_times[tid()] = gettimeofday_us()
}

probe process(@1).mark("*row__done")
{
  engine_times[tid()] = gettimeofday_us() - engine_times[tid()]
}

probe process(@1).mark("query__done")
{
  query_times[tid()] = gettimeofday_us() - query_times[tid()]
  printf("%s\n", queries[tid()])
  printf("Total: %dus Engine: %dus\n",
         query_times[tid()],
         engine_times[tid()])
}
```

Measuring Time in Storage Engine

```
# stap se_time.sty /home/posulliv/repos/mysql/mysql-5.5.2-m2/5.5.2/libexec/mysqld
select @@version_comment limit 1
Total: 1395us Engine: 0us
select USER()
Total: 228us Engine: 0us
SELECT DATABASE()
Total: 586us Engine: 0us
show databases
Total: 753us Engine: 5us
show tables
Total: 893us Engine: 5us
show tables
Total: 3545us Engine: 18us
select * from tables
Total: 22756us Engine: 5us
^C
#
```

CPU Cycles in Storage Engine

```
global query_times
global engine_times
global queries

probe process(@1).mark("query__start")
{
  # get_cycles() returns the processor cycle counter value
  query_times[tid()] = get_cycles()
  engine_times[tid()] = 0
  queries[tid()] = user_string($arg1)
}

probe process(@1).mark("*row__start")
{
  engine_times[tid()] = get_cycles()
}

probe process(@1).mark("*row__done")
{
  engine_times[tid()] = get_cycles() - engine_times[tid()]
}

probe process(@1).mark("query__done")
{
  query_times[tid()] = get_cycles() - query_times[tid()]
  printf("%s\n", queries[tid()])
  printf("Total: %d Engine: %d\n",
         query_times[tid()],
         engine_times[tid()])
}
```

CPU Cycles in Storage Engine

```
# stap se_cycles.sty /home/posulliv/repos/mysql/mysql-5.5.2-m2/5.5.2/libexec/mysqld
select @@version_comment limit 1
Total: 370496 Engine: 0
select USER()
Total: 117047 Engine: 0
SELECT DATABASE()
Total: 659163 Engine: 0
show databases
Total: 1189115 Engine: 8348
show tables
Total: 353008 Engine: 8390
show tables
Total: 2659210 Engine: 31185
insert into t1 values (2, 'sarah')
Total: 8745185 Engine: 6257979
select * from t1
Total: 17086400 Engine: 9733
select b from t1 where a = 2
Total: 20495092 Engine: 61660
^C
#
```

Query Tracing – Lock & Filesort

```
global lock_times
global filesort_times

probe process(@1).mark(“*lock__start”)
{
  lock_times[tid()] = gettimeofday_us()
}

probe process(@1).mark("*lock__done")
{
  lock_times[tid()] = gettimeofday_us() - lock_times[tid()]
}

probe process(@1).mark("filesort__start")
{
  filesort_times[tid()] = gettimeofday_us()
}

probe process(@1).mark("filesort__done")
{
  filesort_times[tid()] = gettimeofday_us() - filesort_times[tid()]
}
```

Putting It All Together

```
# stap query_trace.sty /home/posulliv/repos/mysql/mysql-5.5.2-m2/5.5.2/libexec/mysqld
select @@version_comment limit 1
Total: 287us Locks: 0us Engine: 0us Network: 6us Filesort: 0us
select USER()
Total: 127us Locks: 0us Engine: 0us Network: 6us Filesort: 0us
SELECT DATABASE()
Total: 562us Locks: 0us Engine: 0us Network: 21us Filesort: 0us
show databases
Total: 642us Locks: 0us Engine: 5us Network: 6us Filesort: 0us
show tables
Total: 1148us Locks: 0us Engine: 6us Network: 6us Filesort: 0us
show tables
Total: 6201us Locks: 0us Engine: 19us Network: 23us Filesort: 0us
SELECT DATABASE()
Total: 564us Locks: 0us Engine: 0us Network: 21us Filesort: 0us
show databases
Total: 81us Locks: 0us Engine: 5us Network: 6us Filesort: 0us
show tables
Total: 293us Locks: 0us Engine: 5us Network: 6us Filesort: 0us
select * from t1
Total: 1202us Locks: 32us Engine: 20us Network: 23us Filesort: 0us
select * from t1 where a = 1
Total: 1134us Locks: 30us Engine: 56us Network: 21us Filesort: 0us
select * from t1 order by a desc
Total: 4727us Locks: 26us Engine: 20us Network: 22us Filesort: 3182us
insert into t1 values (3, 'tomas')
Total: 3406us Locks: 146us Engine: 858us Network: 30us Filesort: 0us
select * from t1 order by a desc
Total: 1340us Locks: 27us Engine: 20us Network: 21us Filesort: 228us
^C
#
```

Hot Tables - Reads

```
global tables

probe process(@1).mark("*read__row__start")
{
  tables[user_string($arg1), user_string($arg2)]++
}

probe end
{
  foreach ([schema, table] in tables)
  {
    printf("%s.%s %d\n", schema, table, tables[schema, table])
  }
}
```

```
# stap hot_reads.sty /home/posulliv/repos/mysql/mysql-5.5.2-m2/5.5.2/libexec/mysqld
information_schema./opt/mysql_sandboxes/stap/tmp/#sql_1ecd_0 6
test.t1 8
^C
#
```

Hot Tables - Updates

```
global tables

probe
  process(@1).mark("update__row__start"),
  process(@1).mark("insert__row__start"),
  process(@1).mark("delete__row__start")
{
  tables[user_string($arg1), user_string($arg2)]++
}

probe end
{
  foreach ([schema, table] in tables)
  {
    printf("%s.%s %d\n", schema, table, tables[schema, table])
  }
}
```

```
# stap hot_updates.sty /home/posulliv/repos/mysql/mysql-5.5.2-m2/5.5.2/libexec/mysqld
information_schema./opt/mysql_sandboxes/stap/tmp/#sql_901_0 4
test.t1 3
test.t2 100
^C
#
```

Queries Accessing Particular Table

```
global schema_name
global table_name
global queries
global used

probe begin
{
  schema_name = @1
  table_name = @2
}

probe process(@3).mark("query__exec__start")
{
  queries[tid()] = user_string($arg1)
  used[tid()] = 0
}

probe process(@3).mark("*lock__start")
{
  if (schema_name == user_string($arg1) &&
      table_name == user_string($arg2))
  {
    used[tid()] = 1
  }
}

probe process(@3).mark("query__exec__done")
{
  if (used[tid()])
  {
    printf("%s;\n", queries[tid()])
  }
}
```

Queries Accessing Particular Table

```
# stap par_table.sty test t1 /home/posulliv/repos/mysql/mysql-5.5.2-
m2/5.5.2/libexec/mysqld
select * from t1;
insert into t1 values (4, 'gearoid');
insert into t1 values (5, 'domhnall');
select * from t1 order by a desc;
^C
#
```

Timing Locks

```
global rdlocks
global wrlocks

probe process(@1).mark("handler__rdlock__start")
{
  rdlocks[tid()] = get_cycles()
  printf("Start: Lock->Read %s.%s\n",
         user_string($arg1), user_string($arg2))
}

probe process(@1).mark("handler__rdlock__done")
{
  printf("End: Lock->Read %d cycles\n",
         get_cycles() - rdlocks[tid()])
}

probe process(@1).mark("handler__wrlock__start")
{
  wrlocks[tid()] = get_cycles()
  printf("Start: Lock->Write %s.%s\n",
         user_string($arg1), user_string($arg2))
}

probe process(@1).mark("handler__wrlock__done")
{
  printf("End: Lock->Write %d cycles\n",
         get_cycles() - wrlocks[tid()])
}
```

Timing Locks

```
# stap locktime.stp /home/posulliv/repos/mysql/mysql-5.5.2-m2/5.5.2/libexec/mysqld
Start: Lock->Read test.t1
End: Lock->Read 418788 cycles
Start: Lock->Read test.t1
End: Lock->Read 173971 cycles
Start: Lock->Write test.t1
End: Lock->Write 402910 cycles
Start: Lock->Write test.t1
End: Lock->Write 704391 cycles
Start: Lock->Read test.t1
End: Lock->Read 171652 cycles
Start: Lock->Read test.t1
End: Lock->Read 251286 cycles
^C
#
```

Resources and Thank You!

- SystemTap wiki
  - Lots of documentation and tutorials on how to get started with stap
- SystemTap examples
  - http://sourceware.org/systemtap/examples/
- My blog has some articles on how to get SystemTap up and running on Ubuntu and how to configure MySQL/Drizzle for SystemTap
- SystemTap script repository
  - http://github.com/posulliv/stap