Application Instrumentation for MySQL

What Why and How

Peter Zaitsev, CEO
Percona Inc
18/04/12

Agenda

- Importance of Instrumentation of Application
- What needs to be Instrumented
- How can you do it

Secret Agenda

- Convince you to spend time implementing Instrumentation for your application
  - Before lightning strikes
- … or convince your Boss to let you spend time doing it

Our Focus

- Assuming Web Applications
- Focus on Performance Instrumentation
- Focus on Backend performance
- Assume MySQL is the database
- Use PHP for examples

- **Lets start with Why ?**

Can’t I just look at MySQL ?

- MySQL might not be leading cause of Performance Problems
- Mapping from MySQL queries to user actions can be not trivial

Eliminate Guessing

- Instrumented Applications means you **Know** where response time comes from

Predictable and Focused efforts

- 80% of response time comes from given query
  - Focusing on this query is highest priority
- Can half this query response time ?
  - You know how to reduce backend response time 40%

Get data for Performance SLA

- Know how system is Performing in Numbers
- Have data to analyze SLA violations
  - Want 99.9% responses in 0.5 sec
  - Analyze requests taking over 0.5 sec and see how they can be optimized

Why Instrument in Production ?

- Because you need to know how your real system operates
- In Test system you can have more in depth instrumentation
  - Profiling

Why Instrument All the Time ?

- Should not you just enable instrumentation when system is in trouble ?
- You want point of comparison
- You want to know “good” state of things

  … to understand what have changed

Methods of Sampling

- Most systems can instrument every single request
- “Random Sampling” – good big picture overview
- Dedicated Web Server – hides Web server specific issues
- Random User Sessions
- Do not forget background jobs

- What should we instrument ?

Focus on Response Time

- Total Response Time (Wall Clock Time)
- CPU usage (user and system)
- Response Time for external calls
  - MySQL, Memcache, MongoDB, Web Services
- Potentially expensive DiskIO
- Potentially expensive pieces of Code

Look at Resource Usage

- Memory being the most important these days
- … running out of resources can cause performance problems and failures

Events

- Number of MySQL
  - Connections
  - Selects ? Updated ? Deletes ?
- Memcache hits and Misses
- Errors
  - Deadlocks ?

Additional Information

- "Glue" between the components
  - RequestID which we can pass as comment in MySQL Queries
- Other information helpful for analyses
  - User_id ?
  - Logged in user vs anonymous account ?
  - Information about user interaction
  - Was user Search engine Bot ?

- How
- Lets finally get to meaty stuff

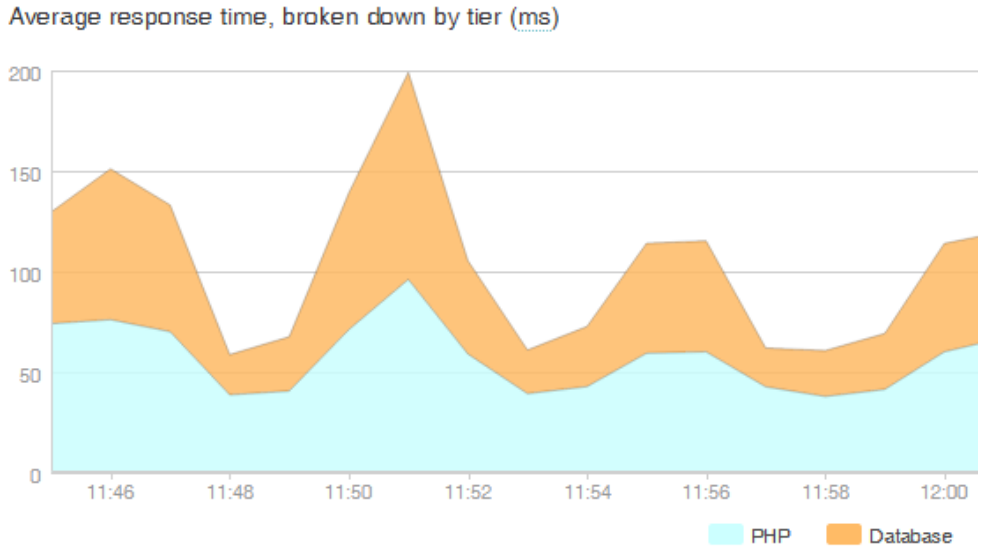
Three Main Choices

- Use SaaS solutions
  - NewRelic, AppDynamics, etc
- Use Instrumentation tools
  - XHProf is good example
- Roll your own
  - http://code.google.com/p/instrumentation-for-php / can provide good start

NewRelic

- Hugely popular tool among Ruby,PHP, Java Developers
- Has good support for MySQL
- Easy to use
- Measures both end user and App Server
- Provided for free with Percona's MySQL Support contracts

Throughtput and ResponseTime

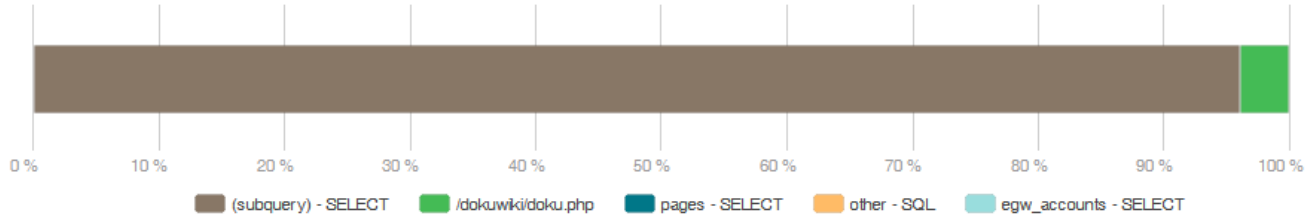


Transaction Traces

- Analyze Slow transactions not complete set

Start time	URL	Resp. time	Average
04/04/12 04:49:51	/dokuwiki/doku.php /dokuwiki/	46,650 ms	1,759 ms
04/03/12 05:35:55	/dokuwiki/lib/exe/fetch.php /dokuwiki/_media/prs:nikki_morton.ai	38,199 ms	3,065 ms
04/03/12 13:27:05	/dokuwiki/lib/exe/fetch.php /dokuwiki/_media/prs:yves_trudeau.ai	30,026 ms	3,065 ms

Summary | Trace details | SQL statements



Slowest components	Count	Duration	%
(subquery) - SELECT	1	44,724 ms	96%
/dokuwiki/doku.php	1	1,708 ms	4%
pages - SELECT	4	216 ms	0%
other - SQL	4	1 ms	0%
egw_accounts - SELECT	3	1 ms	0%
Total		**46,650 ms**	**100%**

Trace details

Summary | Trace details | SQL statements

Expand all Collapse all

Duration (ms)	Duration (%)	Segment	Drilldown	Timestamp
46,650	100.00%	/dokuwiki/doku.php		0.000 s
11.0	0.02%	7 fast method calls		0.006 s
10.0	0.02%	pages - SELECT		0.780 s
192	0.41%	pages - SELECT		0.790 s
44,724	95.87%	(subquery) - SELECT		1.200 s
0.0	0.00%	pages - SELECT		46.627 s
14.0	0.03%	pages - SELECT		46.627 s

SQL query

```
SELECT * FROM ( SELECT assigned, CONCAT('?', iss_id, '?', iss_id, '?') AS iss_id, CASE WHEN TIMESTAMPDIFF(MINUTE, iss_last_public_action_date, NOW()) < ? THEN CONCAT(TIMESTAMPDIFF(MINUTE, iss_last_public_action_date, NOW()), '?') WHEN TIMESTAMPDIFF(HOUR, iss_last_public_action_date, NOW()) < ? THEN CONCAT(TIMESTAMPDIFF(HOUR, iss_last_public_action_date, NOW()), '?') WHEN TIMESTAMPDIFF(DAY, iss_last_public_action_date, NOW()) < ? THEN CONCAT(TIMESTAMPDIFF(DAY, iss_last_public_action_date, NOW()), '?') ELSE CONCAT(TIMESTAMPDIFF(MONTH, iss_last_public_action_date, NOW()), [see the rest…]
```

Timestamp	Duration	SQL
1.200 s	44,724 ms	SELECT * FROM (SELECT assigned, CONCAT('?', iss_id, '?', iss_id, '?') AS iss_id, CASE WHEN TIMESTAMPDIFF(MINUTE, iss_last_public_action_date, NOW()) < ? THEN CONCAT(TIMESTAMPDIFF(MINUTE, iss_last_public_action_date, NOW()), '?') WHEN TIMESTAMPDIFF(HOUR, iss_last_public_action_date, NOW()) < ? THEN CONCAT(TIMESTAMPDIFF(HOUR, iss_last_public_action_date, NOW()), '?') WHEN TIMESTAMPDIFF(DAY, iss_last_public_action_date, NOW()) < ? THEN CONCAT(TIMESTAMPDIFF(DAY, iss_last_public_action_date, NOW()), '?') ELSE CONCAT(TIMESTAMPDIFF(MONTH, iss_last_public_action_date, NOW()) [see the rest…]

SQL details

Duration 44,724 ms

Stack trace

```
…y called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/parser/xhtml.php(370) : eval()'d code (82)
… called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/parser/xhtml.php (370)
in Doku_Renderer_xhtml::php called at ? (?)
…y called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/parserutils.php (555)
…intranet.percona.com/htdocs/dokuwiki/lib/plugins/sphinxsearch/SphinxSearch.php (130)
…a/www/intranet.percona.com/htdocs/dokuwiki/lib/plugins/sphinxsearch/action.php (146)
…a/www/intranet.percona.com/htdocs/dokuwiki/lib/plugins/sphinxsearch/action.php (95)
…nknown called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/events.php (171)
…_event called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/events.php (56)
…before called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/events.php (85)
…rigger called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/events.php (195)
…vent called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/template.php (48)
…led at /data/www/intranet.percona.com/htdocs/dokuwiki/lib/tpl/default/main.php (91)
…clude called at /data/www/intranet.percona.com/htdocs/dokuwiki/inc/actions.php (156)
…act_dispatch called at /data/www/intranet.percona.com/htdocs/dokuwiki/doku.php (102)
```

XHPROF

- Profiler for PHP
  - Stores data in format usable as instrumentation
- Lightweight for production use
- Developed by Facebook for Facebook
- Open Source
- Latest release 0.9.2, about 3 years ago

Profile Summary

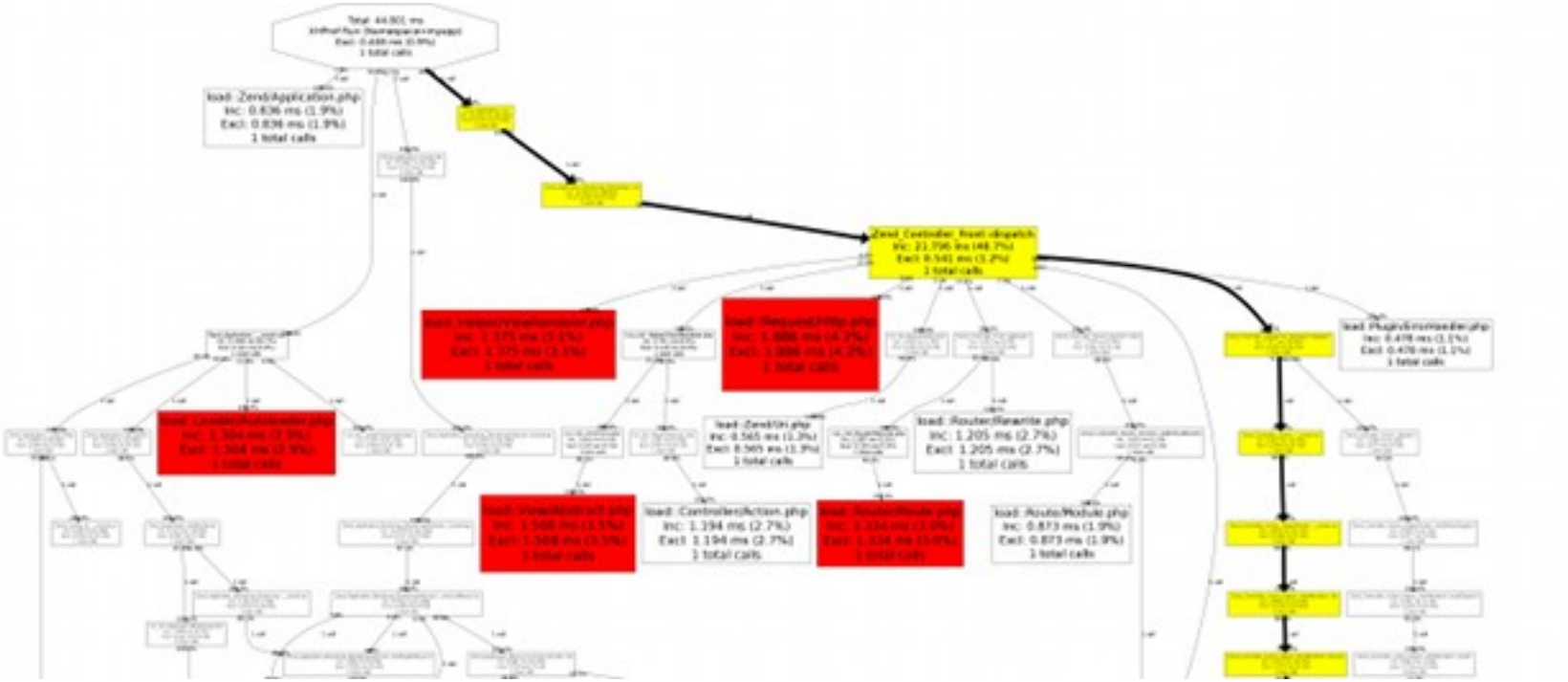
Overall Summary	
Total Incl. Wall Time (microsec):	44,801 microsecs
Total Incl. CPU (microsecs):	44,002 microsecs
Total Incl. MemUse (bytes):	2,459,584 bytes
Total Incl. PeakMemUse (bytes):	2,508,032 bytes
Number of Function Calls:	1,513

[View Full Callgraph]

Displaying top 100 functions: Sorted by Excl. CPU (microsec) [display all]

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Excl. Wall Time (microsec)	EWall%	Incl. CPU (microsecs)	ICpu%	Excl. CPU (microsec)	ECPU%	Incl. MemUse (bytes)	IMemUse%	Excl. MemUse (bytes)
run_init::Dispatcher/Standard.php	1	0.1%	3,899	8.7%	174	0.4%	8,001	18.2%	4,001	9.1%	192,096	7.8%	7,2
load::Loader/PluginLoader.php	1	0.1%	1,099	2.5%	1,099	2.5%	4,001	9.1%	4,001	9.1%	70,956	2.9%	70,9
ucfirst	23	1.5%	18	0.0%	18	0.0%	4,000	9.1%	4,000	9.1%	2,368	0.1%	2,3
load::Bootstrap/Bootstrap.php	1	0.1%	310	0.7%	310	0.7%	4,000	9.1%	4,000	9.1%	7,992	0.3%	7,9
load::Controller/Action.php	1	0.1%	1,194	2.7%	1,194	2.7%	4,000	9.1%	4,000	9.1%	76,392	3.1%	76,3
run_init::Dispatcher/Abstract.php	1	0.1%	3,228	7.2%	1,178	2.6%	4,000	9.1%	4,000	9.1%	140,752	5.7%	1,7
load::Controller/Front.php	1	0.1%	1,657	3.7%	1,657	3.7%	4,000	9.1%	4,000	9.1%	108,104	4.4%	108,1
load::Loader/Autoloader.php	1	0.1%	1,304	2.9%	1,304	2.9%	4,000	9.1%	4,000	9.1%	74,624	3.0%	74,6
load::controllers/IndexController.php	1	0.1%	188	0.4%	188	0.4%	4,000	9.1%	4,000	9.1%	10,744	0.4%	10,7
load::Router/Route.php	1	0.1%	1,334	3.0%	1,334	3.0%	4,000	9.1%	4,000	9.1%	81,500	3.3%	81,5
load::Request/Http.php	1	0.1%	1,886	4.2%	1,886	4.2%	4,000	9.1%	4,000	9.1%	135,104	5.5%	135,1
Zend_Controller_Plugin_Abstract::preDispatch	1	0.1%	1	0.0%	1	0.0%	0	0.0%	0	0.0%	480	0.0%	4
Zend_Controller_Dispatcher_Abstract::getParam	4	0.3%	6	0.0%	6	0.0%	0	0.0%	0	0.0%	1,572	0.1%	1,5
Zend_Controller_Dispatcher_Standard::isDispatchable	1	0.1%	438	1.0%	43	0.1%	0	0.0%	0	0.0%	16,188	0.7%	1,0
Zend_Controller_Request_Abstract::setDispatched	2	0.1%	4	0.0%	4	0.0%	0	0.0%	0	0.0%	984	0.0%	9
Zend_Controller_Plugin_Broker::preDispatch	1	0.1%	8	0.0%	7	0.0%	0	0.0%	0	0.0%	1,096	0.0%	6

See Call Graph



Analyze Callers

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Incl. CPU (microsecs)	ICpu%	Incl. MemUse (bytes)
Current Function							
Zend_Loader_PluginLoader::_formatName	18	31.0%	128	0.3%	4,000	9.1%	2,980
Exclusive Metrics for Current Function			115	89.8%	0	0.0%	2,048
Parent functions							
Zend_Loader_PluginLoader::load	7	38.9%	94	73.4%	4,000	100.0%	1,436
Zend_Loader_PluginLoader::getClassName	4	22.2%	12	9.4%	0	0.0%	692
Zend_Loader_PluginLoader::isLoaded	7	38.9%	22	17.2%	0	0.0%	852
Child function							
ucfirst	18	100.0%	13	10.2%	4,000	100.0%	932

Compare Traces

- Analyze Regressions

Overall Diff Summary

	Run #4af72e92462e5	Run #4af75948da32e	Diff	Diff%
Number of Function Calls	2,101	1,513	-588	-28.0%
Incl. Wall Time (microsec)	150,333	1,856,010	1,705,677	1134.6%
Incl. CPU (microsecs)	28,002	120,007	92,005	328.6%
Incl. MemUse (bytes)	364,584	2,459,644	2,095,060	574.6%
Incl. PeakMemUse (bytes)	917,100	2,508,092	1,590,992	173.5%

[View Regressions/Improvements using Callgraph Diff]

Top 100 *Regressions*/*Improvements*: Sorted by Incl. Wall Time (microsec) Diff [display all]

Function Name	Calls Diff	Calls Diff%	Incl. Wall Diff (microsec)	IWall Diff%	Excl. Wall Diff (microsec)	EWall Diff%	Incl. CPU Diff (microsec)	ICpu Diff%	Ex (mi
main()	0	0.0%	1,705,677	100.0%	13,347	0.8%	92,005	100.0%	
Zend_Application::__construct	1	0.2%	794,810	46.6%	39,236	2.3%	44,002	47.8%	
Zend_Loader_Autoloader::autoload	6	1.0%	689,138	40.4%	19,449	1.1%	44,003	47.8%	
Zend_Loader_Autoloader::_autoload	6	1.0%	661,052	38.8%	66	0.0%	44,003	47.8%	
call_user_func@1	6	1.0%	660,965	38.8%	44	0.0%	44,003	47.8%	
Zend_Loader::loadClass	6	1.0%	660,921	38.7%	6,294	0.4%	44,003	47.8%	
call_user_func	-23	-3.9%	660,777	38.7%	-128	-0.0%	44,003	47.8%	

PHP Instrumentation Framework

- Set of Classes to instrument your PHP applications
- Extends MySQL(i) Classes for instrumentation
- Augments MySQL Queries
- Logging of profiling Data
  - Integrates with Apache logs

Augmented query example

The query comment consists of key value pairs:

```
-- File: index.php Line: 118 Function: fullCachePage request_id: ABC session_id: XYZ
select Socialeventid,
        Title,
        Summary,
        Imagethumburl,
        Createdtimestamp,
        Eventdate,
        Submitterusername
from SOCIALEVENT
where  eventtimestamp>=CURRENT_TIMESTAMP
ORDER BY eventdate ASC limit 0,10;
```

mk-query-digest can use the pairs as attributes:

```
--embedded-attributes '^-- [^\n]+','(\w+): ([^\t]+)'
```

Apache configuration Example

```
SetEnvIf Request_URI \.php instrumented  # SET instrumented to ON for PHP reqs
#
# Instrumented application logging
# May be loaded with LOAD DATA INFILE
# Use %D for request time instead of %T because %D is microseconds
# mod_logio is required for %I %O
# memcache counters omitted for brevity
LogFormat "\"%{%Y-%m-%d %H:%M:%S}t\" %a %I %O %D %f %H %m \"%q\" %>s %V %
{CTR_total_cpu_time}e %{CTR_cpu_user}e %{CTR_cpu_system}e %{CTR_memory_usage}e \"%
{CTR_request_id}e\" \"%{CTR_SESSION_uname}e\" \"%{CTR_session_id}e\" %
{CTR_mysql_query_count}e %{CTR_mysql_prepare_count}e %{CTR_mysql_prepare_time}e %
{CTR_mysql_connection_count}e %{CTR_mysql_query_exec_time}e  performance

#regular access log for all requests
CustomLog logs/access_log common

#performance data  goes in this log ONLY for PHP apps (see SetEnvIf above)
CustomLog logs/performance_log performance  env=instrumented
```

Configuring MySQL Logging

Configure Slow Query Log for MySQL (Using Percona Extensions)

```
long_query_time=0              #zero=all, .001, .01,.1, 5, 10
slow_query_log=on              #on|off
slow_query_log_file=slow.log
log_slow_verbosity=full        #full, innodb,microsecond,query_plan
slow_query_rate_limit=N        #only log every Nth session
```

Percona Toolkit's pt-query-digest can be used to analyze the slow query log
It supports embedded attributes in SQL queries, which the example query augmenter provides.
It can record stats from the slow log into review tables

LOAD DATA INFILE can be used to load the Apache performance log into a table

Create Table to store profiling data

```
CREATE TABLE `performance_log` (
  `access_time` datetime DEFAULT NULL,
  `remote_address` varchar(25) DEFAULT NULL,
  `bytes_in` bigint(20) unsigned DEFAULT NULL,
  `bytes_out` bigint(20) unsigned DEFAULT NULL,
  `service_time` bigint(20) DEFAULT NULL,
  `file` varchar(100) DEFAULT NULL,
  `protocol` char(10) DEFAULT NULL,
  `action` char(10) DEFAULT NULL,
  `query_string` text,
  `status` smallint(5) unsigned DEFAULT NULL,
  `virtualhost` varchar(50) DEFAULT NULL,
  `total_cpu_time` float DEFAULT NULL,
....
```

Load Data from Log File

- Can implement script for incremental loading

```
mysql> load data
    infile '/var/log/httpd/performance_log'
    into table performance_log
    fields terminated by ' '
    optionally enclosed by '"' ;
Query OK, 5471 rows affected, 1946 warnings (0.12 sec)
Records: 5471  Deleted: 0  Skipped: 0  Warnings: 0

mysql> show warnings;
+---------+------+-----------------------------------------------------------------------------+
| Level   | Code | Message                                                                     |
+---------+------+-----------------------------------------------------------------------------+
| Warning | 1366 | Incorrect integer value: '-' for column 'memory_usage' at row 366

....
```

VIEWS can be great for ease of use

```
CREATE ALGORITHM=MERGE VIEW `performance_view` AS SELECT
                `mysql_query_count` + `mysql_prepare_count` AS `mysql_ops`,
from `performance_log`
mysql> desc performance_view;
```

Field	Type	Null	Key	Default	Extra
access_time	datetime	YES		NULL	
remote_address	varchar(25)	YES		NULL	
bytes_in	bigint(20) unsigned	YES		NULL	
bytes_out	bigint(20) unsigned	YES		NULL	
service_time	bigint(20)	YES		NULL	
file	varchar(100)	YES		NULL	
protocol	char(10)	YES		NULL	
action	char(10)	YES		NULL	
query_string	text	YES		NULL	
status	smallint(5) unsigned	YES		NULL	
virtualhost	varchar(50)	YES		NULL	
total_cpu_time	float	YES		NULL	
cpu_user	float	YES		NULL	
cpu_system	float	YES		NULL	
memory_usage	int(10) unsigned	YES		NULL	
request_id	char(40)	YES		NULL	
SESSION_uname	varchar(25)	YES		NULL	
session_id	char(32)	YES		NULL	
mysql_ops	int(9)	YES		NULL	
mysql_connection_count	mediumint(9)	YES		NULL	
mysql_time	double	YES		NULL	
memcache_connection_count	mediumint(9)	YES		NULL	
memcache_ops	bigint(15)	YES		NULL	
memcache_time	double	YES		NULL	

24 rows in set (0.00 sec)

Daily Performance Summary

```
create or replace view file_performance_day
as
select date(access_time) day,
      file,
      count(*) cnt,
      sum(bytes_in) / 1024 kb_in,
      sum(bytes_out) / 1024 kb_out,
      sum(bytes_in) / 1024 / 1024 mb_in,
      sum(bytes_out) / 1024 / 1024 mb_out,
      round(sum(service_time / 1e6),4) service_time,
      round(sum(total_cpu_time),4) total_cpu_time,
      round(sum(cpu_user),4) cpu_user,
      round(sum(cpu_system),4) cpu_system,
      sum(memory_usage) / 1024 / 1024 memory_usage_MB,
      round(sum(mysql_time),4) mysql_time,
      sum(mysql_ops) mysql_ops,
      sum(memcache_ops) memcache_ops,
      round(sum(memcache_time),4) memcache_time,
      sum(mysql_connection_count) mysql_connection_count,
      sum(memcache_connection_count) memcache_connection_count,
      round( sum(mysql_time) / sum(service_time/1e6) * 100, 2) mysql_pct,
      round( sum(memcache_time) / sum(service_time/1e6) * 100, 2) memcache_pct,
      round( sum(cpu_user + cpu_system) / sum(service_time/1e6) * 100, 2) cpu_pct,
      round( ( sum(service_time/1e6) - ( sum(mysql_time) + sum(memcache_time) + sum(cpu_user + cpu_system) ) ) /
sum(service_time / 1e6),4) * 100 other_pct
  from performance_view
group by day,file
```

What has largest combined service time ?

```
mysql> select * from file_performance_day order by service_time desc limit 1\G
*************************** 1. row ***************************
                    day: 2010-04-11
                   file: /var/www/html/oliophp/public_html/taggedEvents.php
                    cnt: 858
                  kb_in: 203.4248
                 kb_out: 14249.8896
                  mb_in: 0.19865704
                 mb_out: 13.91590786
           service_time: 265.7764
        total_cpu_time: 13.5320
              cpu_user: 7.6168
            cpu_system: 5.9151
       memory_usage_MB: 161.22107697
            mysql_time: 252.2444
             mysql_ops: 4805
          memcache_ops: 0
         memcache_time: 0.0000
  mysql_connection_count: 1566
memcache_connection_count: 0
              mysql_pct: 94.91
          memcache_pct: 0.00
               cpu_pct: 5.09
             other_pct: 0.0000
1 row in set (0.09 sec)
```

Now Look at MySQL Logs to find why

- Use pt-query-digest with embedded attributes and filter by file
  - Same works for request id, session etc

- pt-query-digest /var/lib/mysql/slow --embedded-attributes '^-- [^\n]+','(\w+): ([^\t]+)' --filter '$event->{File} && $event->{File} =~ m/taggedEvent/'

```
# Rank Query ID           Response time    Calls R/Call   Item
# ==== ================== ================ ===== ======== ===============
#    1 0x27A1E0E70C50976A   29.1573 51.7%   708   0.0412 SELECT SOCIALEVENT PERSON_SOCIALEVENT
#    2 0xFB3C3A24510F4D17   14.4064 25.5%   858   0.0168 SELECT SOCIALEVENTTAG SOCIALEVENTTAG_SOCIALEVENT
#    3 0xD5E65E7DEAA2876A    6.5342 11.6%   429   0.0152 SELECT SOCIALEVENTTAG
#    4 0xE00FB796CC8A5183    6.3538 11.3%  2810   0.0023 SELECT SOCIALEVENT
```

Summary

- Application Instrumentation is very important for performance management
- Multiple solutions are available
- Their functionality overlaps but different

Thank You !

- pz@percona.com
- http://www.percona.com
- @percona
  - Use #perconalive
- http://www.facebook.com/Percona
- Slides will be published on the conference site