

【总结】第三章：探索（profiling）服务器表现(HP-MySQL 3rd)

2016-08-22 | 10

High Performance MySQL, 3rd Edition

实践中常被咨询的三个问题：

- 服务器是否以最优状态运行？
- 为什么这个查询不够快？
- 如何定位很奇怪的异常？

这章主要解决这三个问题，并介绍一些工具。

有个简单的办法解决这些问题，就是测试服务器主要在做什么，把时间花在了哪，这个过程就叫做profiling，打印出服务器的表现。这一章我们主要就是学会如何探索，无论是一个应用还是一个查询。

下面先解释下晦涩的理论，后面会有例子。

性能优化的简介

每个人对性能的有着自己的定义。这里我们标准化为：完成任务的时间，也就是反应时间。对于数据库服务器而言，任务应是一堆SELECT、UPDATE、INSERT，而非一个。当然也有对于一个查询的反应时间。

显然，优化就是指尽可能的降低反应时间。

有些人会误解，比如会认为需要降低CPU使用资源，这是错误，CPU的资源就是用来被消耗，新版本的MySQL提供的反应时间的同时也增加了CPU的使用。这是正常的。

还有一种误解，就是改进每秒能进行的查询数，这叫做吞吐量优化，而非性能优化。

为了改进任务的反应时间，首先要知道为了任务，时间都花在哪了。我们必须知道时间花哪了，才能改进。

有一种常见的错误，有人喜欢改动，却不测量时间具体的花费。相反，我们尽可能的多测时间的花费，确定不了问题就不会乱改动。往往我们尽可能了解的时候，解决方案就直接出来了。但实际上也是困难，因为尽管我们知道时间花哪了，却不知道为什么花那里，一样无从下手。

花费的时间主要有两部分：等待或执行。分别需要不同的工具。

等待比较复杂，涉及与别的任务的交互，也许是资源争夺方面的问题，往往不易解决。

执行方面，可以测试、优化子任务的时间，哪优化哪个子任务最有效呢？这就是profiling出现的原因。

不要完成相信测试的结果，测试不能完全代表真正的质量、性能，只是性能侧面的表现。

通过profiling优化性能

profiling分两步：测试一堆任务时间，再排序研究时间耗费最多的。

下面是一个使用pt-query-digest工具产生的 profiling 的报告，内容简单，好理解。

Rank	Response time	Calls	R/Call	Item
1	11256.3618	68.1%	78069	0.1442 SELECT InvitesNew
2	2029.4730	12.3%	14415	0.1408 SELECT StatusUpdate
3	1345.3445	8.1%	3520	0.3822 SHOW STATUS

最后一列是Query的简述，在本次报告中query即是任务。

profiling报告也分两种，执行时间的报告和等待分析。执行时间报告显示了哪些任务最耗时，而等待分析显示了任务在哪卡住。

一般哪个时间长就分析哪部分，也可以同时产出两种报告。

在本例中，SELECT InvitesNew 表耗费较长时间，从更底层来将，也许是等待是I/O完成。

除了使用工具直接得到结论，也许从系统的外部表现、日志来分析，

解析profile报告

上面的报告很简单，没有显示更多的有价值的信息，比如：

- 有优化空间的查询：选择有优化空间的查询需要技巧，注意有两类不值得优化，第一是占整体时间少的查询，无论怎么优化，也无法提供整体速度。第二是优化成本太高的查询。
- 异常点：有些查询发生少，但是偶然发生了，确极其慢，这对用户来说是无法接受的。但这样的任务却很难排到顶部引起人们注意。
- 未被统计到的时间：好的profiling工具能测到未被统计的时间。比如，整个线程运行了10秒，但是你的profile的子任务耗费时间之和加起来只有9.7秒，那么有300毫秒的时间未被统计到。出现的原因可能是四舍五入的错误、测量工具造成的。请注意这一点。
- 隐藏的数据：使用多种统计学技巧来了解数据的真实情况，而非单纯依靠一个指标，如平均值。常用的指标有：直方图、百分位数、标准差、离散指数（index of dispersion）。

实际上，pt-query-digest就是一个非常好的工具。这里为了举例简化了很多。

另外，上面的例子还忽略了查询语句之间的联系。我们不应将语句割裂来看，也可以在事务甚至应用的高层来观测，下一节将讲述这个内容。

profiling应用

profiling整个应用能让我们从头到尾了解用户与服务器交互时各个阶段时间的消耗，更利于发现真正的瓶颈，而非错怪了MySQL。常见的瓶颈如下：

- 请求外部资源，如请求web service或者搜索引擎。
- 对大文件、大数据的操作，如解析XML。
- 迭代次数太多的操作，如滥用正则。
- 错用算法。

在旧应用中加入profiling代码比较吃力，但是为新的应用插入profiling代码十分简单且有必要，所以请引起重视。

profiling代码是否会降低服务器速度（可以简单的理解profiling就是在各个阶段的节点插入日志）

会降低，但是带来的价值无可估量，从评估服务器表现到硬件购买、从预测流量高峰到debug等等，oracle如果去掉所有profiling的代码会让整体性能提高10%，但是还是profiling代码更有价值。

生产环境中有些小技巧：

```
<?php
$profiling_enabled = rand(0, 100) > 99;
?>
```

这样即可可以保留重要的历史数据，又不至于对用户体验造成太大的影响。

目前已经有比较成熟的工具：New Relic，能够非常好完成这个过程。

profiling PHP 应用

推荐使用facebook开发和开源的工具：xhprof，质量好，可用于生产环境中。

此外除了xhprof，还有xdebug、Valgrind、cachegrind能够从各个角度审视你的代码，但是这些工具消耗较多资源，不宜于生产环境中使用。

instrumentation-for-php (IfP)也值得推荐，但非针对PHP，它主要用于测试数据库的请求所消耗的时间，可用于生产环境。

另外，使用IfP，我们可以选择记录PHP的函数调用。但是三种重要的请求会被自动记录：网页请求、数据库语句、缓存请求。并且结果会自动写入Apache服务器。

在web应用开始前加入一下代码即可使用IfP：

```
require_once('Instrumentation.php');
Instrumentation::get_instance()->start_request();
```

IfP还自动对SQL添加评论，这样更方便了每一句SQL的分析。如下所示，添加的评论包括请求页面、函数等等信息，非常方便追踪效果。

```
- File: index.php Line: 118 Function: fullCachePage request_id: ABC session_id: XYZ
SELECT * FROM ...
```

具体如何使用IfP根据情况各有不同。

但是我查了一下，发现非常冷门，没什么人用。而常见的Yii框架已经提供了内置的profiling功能了。使用框架的话，完全不用考虑这些细节。

对于打印日志的抽取和分析可以使用pt-query-digest工具，具体如何使用可以参见[官网](#)。

Profiling MySQL 语句

可以从两个方面Profiling SQL语句：从整个服务器找出哪个SQL最耗时和优化一个特定的SQL语句。

profiling整个服务器的SQL

主要依靠slow query log，可以设置long_query_time确定大于多少时长的query被记录，如果设置为0，也就是记录所有的query。单位是微妙。
slow query log给服务器带来的负担可以忽略不计，但是需要注意日志的大小以免打满磁盘。

相比slow query log，general log只会记录到达MySQL的查询，但是不会打印响应时间、执行计划等 profiling 的信息。

MySQL某版本允许把query打印进表里，以供分析，但是非常耗费资源，不推荐使用。

Percona服务器能打印更详细的调试信息，比如query执行计划、锁、I/O活动等，以应对不同的查询场景。

当没办法登录到数据库服务器查询slow query log时，

对于Percona服务器有一个工具：pt-query-digest 可以帮助打印信息。

第二种方式捕获TCP网络流量然后分析它，用MySQL客户端与服务器之间的信息解析流量。具体说来，先用tcpdump保存流量到磁盘，再用pt-query-digest -type=tpcdump解析query。当使用高级协议时同样有效。

分析 query log

建议至少经历一次打印所有query到slow query log，然后逐一分析。当然不能一直开着，在高峰时期打印一小时或者一分钟都有利于找到劣质query。

不要打开日志一个一个看，而是用代码生成汇总（统计）信息，再找到从里面的找到劣质query。

推荐使用pt-query-digest工具生成汇总信息，使用简单，功能齐全，还可以将query导入到数据库，再跟踪变化。

观察上面是示例可以发现第一列是标识符，并且会整合有相同目的的查询。但是因为3、4有不同的标识符所以3和4其实是不同的查询。

V/M列是离散指数，离散指数越高，表明多次执行的时间不稳定，表明此查询越值得优化。

最后一列表明了其它17种查询，工具认为这些查询不值得优化，所以没有单独报告。

可以根据排名序号或者唯一标识符打印出某个查询更详细的信息：

```
# Query 1: 24.28 QPS, 3.50x concurrency, ID 0xBFCF8E3F293F6466 at byte 5590079
# This item is included in the report because it matches --limit.
# Scores: V/M = 0.21
# Query_time sparkline: | _^_.^_ |
# Time range: 2008-09-13 21:51:55 to 22:45:30
# Attribute      pct     total      min      max      avg      95%    stddev   median
# ======      ==      =====      =====      =====      =====      ======    =====   =====
# Count          63    78069
# Exec time     68   11256s     37us      1s    144ms    501ms    175ms    68ms
# Lock time     85    134s       0    650ms      2ms    176us    20ms    57us
# Rows sent      8   70.18k       0       1    0.92    0.99    0.27    0.99
# Rows examine    8   70.84k       0       3    0.93    0.99    0.28    0.99
# Query size     84   10.43M    135    141  140.13  136.99    0.10   136.99
# String:
# Databases     production
# Hosts
# Users         fbappuser
# Query_time distribution
#   1us
# 10us #
# 100us #####
# 1ms #####
# 10ms #####
# 100ms #####
# 1s #
# 10s+
# Tables
#   SHOW TABLE STATUS FROM `production` LIKE 'InvitesNew82'\G
#   SHOW CREATE TABLE `production`.`InvitesNew82`\G
# EXPLAIN /*!50100 PARTITIONS*/
SELECT InviteId, InviterIdentifier FROM InvitesNew82 WHERE (InviteSetId = 87041469)
AND (InviteeIdentifier = 1138714082) LIMIT 1\G
```

信息已经列举的非常清楚了，比如还有查询的分布，在最后还列出了表现最差的SQL语句。

pt-query-digest 是非常重要的分析工具，尽可能熟练掌握。

profiling 单个SQL语句

优化SQL的内容贯穿了本书，在这里只是讲述如何测量一个SQL在做什么和SQL在执行过程中每一步耗费的时间，这将帮助我们选择优化技术。

实践中，主要使用SHOW STATUS、SHOW PROFILE和检查slow log日志。下面对这三种方式逐一介绍。

SHOW PROFILE

对于一个会话，使用如下命令打开PROFILE：

```
mysql> SET profiling = 1;
```

之后执行任何语句都会被记录详细的测试信息。比如：

```
mysql> SELECT * FROM sakila.nicer_but_slower_film_list;
[query results omitted]
```

997 rows in set (0.17 sec)

然后我们可以列举所有被记录测试的query：

```
mysql> SHOW PROFILES;
+-----+-----+-----+
| Query_ID | Duration | Query
+-----+-----+-----+
|      1 | 0.16767900 | SELECT * FROM sakila.nicer_but_slower_film_list |
+-----+-----+-----+
```

可以看到执行时间表示的精度更高了，还可以获得更详细的信息：

```
mysql> SHOW PROFILE FOR QUERY 1;
+-----+-----+
| Status          | Duration |
+-----+-----+
| starting        | 0.000082
| Opening tables | 0.000459
| System lock    | 0.000010
| Table lock     | 0.000020
| checking permissions | 0.000005
| checking permissions | 0.000004
| checking permissions | 0.000003
| checking permissions | 0.000004
| checking permissions | 0.000560
| optimizing      | 0.000054
| statistics      | 0.000174
| preparing       | 0.000059
| Creating tmp table | 0.000463
| executing       | 0.000006
| Copying to tmp table | 0.090623
| Sorting result  | 0.011555
| Sending data   | 0.045931
| removing tmp table | 0.004782
| Sending data   | 0.000011
| init            | 0.000022
| optimizing      | 0.000005
| statistics      | 0.000013
| preparing       | 0.000008
| executing       | 0.000004
| Sending data   | 0.010832
| end             | 0.000008
| query end       | 0.000003
| freeing items   | 0.000017
| removing tmp table | 0.000010
| freeing items   | 0.000042
| removing tmp table | 0.001098
| closing tables  | 0.000013
| logging slow query | 0.000003
| logging slow query | 0.000789
| cleaning up     | 0.000007
+-----+-----+
```

但这个表没有按执行时长排序，我们可以通过查询INFORMATION_SCHEMA来获得更佳的表达方式：

```
mysql> SET @query_id = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT STATE, SUM(DURATION) AS Total_R,
```

Chapter 3: Profiling Server Performance

```
->     ROUND(
->         100 * SUM(DURATION) /
->             (SELECT SUM(DURATION)
->                 FROM INFORMATION_SCHEMA.PROFILING
->                 WHERE QUERY_ID = @query_id
->                     ), 2) AS Pct_R,
->     COUNT(*) AS Calls,
->     SUM(DURATION) / COUNT(*) AS "R/Call"
->     FROM INFORMATION_SCHEMA.PROFILING
->     WHERE QUERY_ID = @query_id
->     GROUP BY STATE
->     ORDER BY Total_R DESC;
```

STATE	Total_R	Pct_R	Calls	R/Call
Copying to tmp table	0.090623	54.05	1	0.0906230000
Sending data	0.056774	33.86	3	0.0189246667
Sorting result	0.011555	6.89	1	0.0115550000
removing tmp table	0.005890	3.51	3	0.0019633333
logging slow query	0.000792	0.47	2	0.0003960000
checking permissions	0.000576	0.34	5	0.0001152000
Creating tmp table	0.000463	0.28	1	0.0004630000
Opening tables	0.000459	0.27	1	0.0004590000
statistics	0.000187	0.11	2	0.0000935000
starting	0.000082	0.05	1	0.0000820000
preparing	0.000067	0.04	2	0.0000335000
freeing items	0.000059	0.04	2	0.0000295000
optimizing	0.000059	0.04	2	0.0000295000
init	0.000022	0.01	1	0.0000220000
Table lock	0.000020	0.01	1	0.0000200000
closing tables	0.000013	0.01	1	0.0000130000
System lock	0.000010	0.01	1	0.0000100000
executing	0.000010	0.01	2	0.0000050000
end	0.000008	0.00	1	0.0000080000
cleaning up	0.000007	0.00	1	0.0000070000
query end	0.000003	0.00	1	0.0000030000

现在我们一眼就能看出是复制临时表太费事了，所以我们应重写SQL语句，使其不使用临时表。

第二耗时的是“Sending data”，这代表了很多行为，一般很难优化。

尽管我知道了时长，但是并不知道为什么如此。所以，我们必须再进一步对各个部分的子任务进行研究。

SHOW STATUS

这个命令返回一堆计数器，全局的或者是会话级的，注意混合了两类。比如Queries计数器，返回就是会话级的执行Query的个数。如果只看全局的计数器，可以使用SHOW GLOBAL STATUS。

SHOW STATUS只是展示了各种活动发生的次数，比如从读索引发生了多少次，但是不会显示消耗的时间。除了Innodb_row_lock_time，这是一个全局的计数器。

通过SHOW STATUS，我们可以从某项活动发生的次数来估算它对查询时间的影响。最重要的计数器有：the handler * 相关计数器和the temporary file and table counters。附录B有详细的解释。下列展示这两个计数器：

```
mysql> FLUSH STATUS;
mysql> SELECT * FROM sakila.nicer_but_slower_film_list;
[query results omitted]
mysql> SHOW STATUS WHERE Variable_name LIKE 'Handler%'
      OR Variable_name LIKE 'Created%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Created_tmp_disk_tables | 2      |
| Created_tmp_files     | 0      |
| Created_tmp_tables    | 3      |
| Handler_commit         | 1      |
| Handler_delete         | 0      |
| Handler_discover       | 0      |
| Handler_prepare        | 0      |
| Handler_read_first     | 1      |
+-----+-----+
```

. If you own the second edition of this book, you'll notice that we're

Chapter 3: Profiling Server Performance

Handler_read_key	7483
Handler_read_next	6462
Handler_read_prev	0
Handler_read_rnd	5462
Handler_read_rnd_next	6478
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	6459

可以看见创建了三次临时表，并且有两个是在磁盘上创建的。由Handler_read_rnd_next可见，进行了多次无索引的读。如果没有更多信息，我们可以推断这个语句做了一次无索引的join，这可能是因为子语句创建了一个临时表，接着又做了一次右连接。注意由子语句创建的临时表是没有索引的。

另外，SHOW STATUS也会创建临时表，所以会增加一些handler的次数，通过SHOW PROFILES观察，大约会增加2次。

与SHOW STATUS相比，EXPLAIN是服务器以为它自己会做什么，而SHOW STATUS是实际做了什么，EXPLAIN无法发现临时表是否在磁盘上创建。更多EXPLAIN的内容请见目录D。

使用show query log

这里主要介绍了在Percona Server里slow query log和pt-query-digest对slow query log的使用。对我暂时没有什么用处。

使用Performance Schema

本书写作期间是MySQL5.5，目前此工具没有开发的很好，但是以后可能很实用。

使用Profile优化

Profile也许指名了问题，但是没有给出方案：如何优化query和服务器。解决方案需要大量的知识，比如MySQL如何执行一个query。以后更多章节将会讨论这个问题。

一般情况下我们的优化都基于不完全的Profile信息，比如上面的例子中，我们可以看出建立临时表和未索引的读耗费了最多的时间（但实际上这是无法证实的）。很多问题难于解决，正是因为我们无法测量所有的信息。有时我们也会忽略了服务器的状况，而过于专注优化语句。比如我们只测量语句执行中的情况，而忽略了query执行前的瞬间。

有时一个query千百次执行时都很快，但是某一次很慢，这种情况我们更多的考虑是服务器别的活动占用了资源，比如：备份等。

诊断突发的问题

突发问题是指出服务器偶然卡住或其他不是慢查询的问题。甚至包括一些比较虚幻问题：你找它的时候他根本不出现，也就是很难复现的问题。不要采用试错的方法来解决这些问题。因为根本就不知道在哪的话，乱试可能使结果更糟。所以关键是定位问题，使用不的方法和工具。

下面是一些常见的突发问题：

- Web应用在执行curl，这是一个费时的操作。
- memcached重要的缓存到期了，导致应用对MySQL发起了许多请求用于重新生成缓存项。
- DNS查询偶尔会失败。
- query的缓存偶尔会失效，因为MySQL间歇性发生了锁竞争或删除缓存query的算法低效。
- 并发过高时，InnoDB稳定性的限制将导致Query优化时间过长。

所以有时我们一开始就定位错了方向，尽可能的测试的更加完整。

下面将介绍一些方法来解决突发问题。

单个query的问题还是整个服务器的问题

如果有证据确认一个问题，那么首先需要确认这是一个query的问题还是真个服务器的问题。如果整个服务器都受到影响，那么就不太可能是一个query的问题。很多慢查询都是别的问题造成的，当然，如果就一个query很慢，那么就由必要仔细看看这个query了。

通过升级可以避免一些常见的服务器的问题。

当问题能够重复发生时，通过下面的三种方式我们可以轻易识别是单个query的问题还是服务器的问题。

使用SHOW GLOBAL STATUS

这个技术不需要什么特权，只需要高频执行（每分钟）SHOW GLOBAL STATUS即可。主要观察Threads_running, Threads_connected, Questions, Queries的数量随时间的变化。

```
$ mysqladmin ext -i1 | awk '
/Queries/{q=$4-qp;qp=$4}
/Threads_connected/{tc=$4}
/Threads_running/{printf "%5d %5d %5d\n", q, tc, $4}'
2147483647    136      7
 798    136      7
 767    134      9
 828    134      7
```

Again, don't do that without a good reason to believe that it's the solution.

Dia

683	134	7
784	135	7
614	134	7
108	134	24
187	134	31
179	134	28
1179	134	7
1151	134	7
1240	135	7
1000	135	7

其中Threads_running:目前执行的query个数。如果queries per second低于正常水平，则说明服务器有异常，此外另外两个指标可能会暴涨或者保持水平。

两种常见的情况可以引发监控的指标发生剧烈的变化。一种是服务器发生了瓶颈，旧query拿着一堆锁，而新query需要这些锁而堆积。另一种是缓存集体过期，那么发起了大量请求。

因为每秒钟只有一条信息，所以这个程序可以长期运行。

使用 SHOW PROCESSLIST

这个命令会展现所有进程，并列举基本信息。我们可以统计进程的state来判断是否有异常。进程不该长期处于非常特别的状态，比如statistics，这个状态只是为了优化join的顺序。还有也不该有大量用户处于Unauthenticated user，所以也不会有大量进程处于connection

handshake的状态。

```
$ mysql -e 'SHOW PROCESSLIST\G' | grep State: | sort | uniq -c | sort -rn
 744  State:
  67  State: Sending data
  36  State: freeing items
   8  State: NULL
   6  State: end
   4  State: Updating
   4  State: cleaning up
   2  State: update
   1  State: Sorting result
   1  State: logging slow query
```

换grep的行就可以统计不同的信息。从上面的例子中我们可以看到大量query处于查询结束的阶段，如：“freeing items”，“end”，“cleaning up”，“logging slow query”。如果大量进程处于freeing items阶段，则说明有问题。

除了使用命令行模式外，也可以查询INFORMATION_SCHEMA数据库中的PROCESSLIST表，或者innovate工具进行统计。对于一种常见的异常，就是使用MyISAM引擎时，有大量query处于Locked的状态，这是因为MyISAM进行的是表锁，当有大量的写操作时，将会造成服务器的暂时停顿（卡）。

使用query logging

使用query logging分析问题时，将slow log打开、设置long_query_time为0，并且确定所有的连接都使用了新的设置。如果没办法接触到query log，那可以使用tcpdump和pt-query-digest来模拟query log。

主要是使用日志寻找到吞吐量下降的时间段，query一般会在完成时记录在slow query log中，所以如果服务器卡住的话，slow query log中记录的完成量也会下降。

下面是一个例子：

```
$ awk '/^# Time:/ {print $3, $4, c; c=0} /^# User/{c++}' slow-query.log
080913 21:52:17 51
080913 21:52:18 29
080913 21:52:19 34
080913 21:52:20 33
080913 21:52:21 38
080913 21:52:22 15
080913 21:52:23 47
080913 21:52:24 96
080913 21:52:25 6
080913 21:52:26 66
080913 21:52:27 37
080913 21:52:28 59
```

可以看到吞吐量在达到峰值后又有一个明显下降。在没有时间戳的情况下很难说清楚发生了什么。经过详细的调查，我们发现了达到峰值是因为所有的连接中断了，而这是因为应用服务器发生了重启。所以也不都是MySQL的问题。

解释所有的发现

发现异常最好使用图表（比如我就可以使用python），可以先从SHOW STATUS或SHOW PROCESSLIST开始，这里两个都比较简单，而slow log query则复杂的多。

发现问题之后，下面主要是找原因。

Capturing Diagnostic Data

（感觉和现在需要的有点远，先不看了）

#MySQL

◀ 【总结】第二章：MySQL基准测试(HP-MySQL 3rd)

用SQL完成推荐：推荐相似用户喜爱的物品 ▶

0条评论

还没有评论，沙发等你来抢

社交帐号登录：[微信](#) [微博](#) [QQ](#) [人人](#) [更多»](#)



说点什么吧...

发布

zhangyang's blog正在使用多说

© 2017 ❤ Zhang Yang

由 [Hexo](#) 强力驱动 | 主题 - [NexT.Pisces](#)

