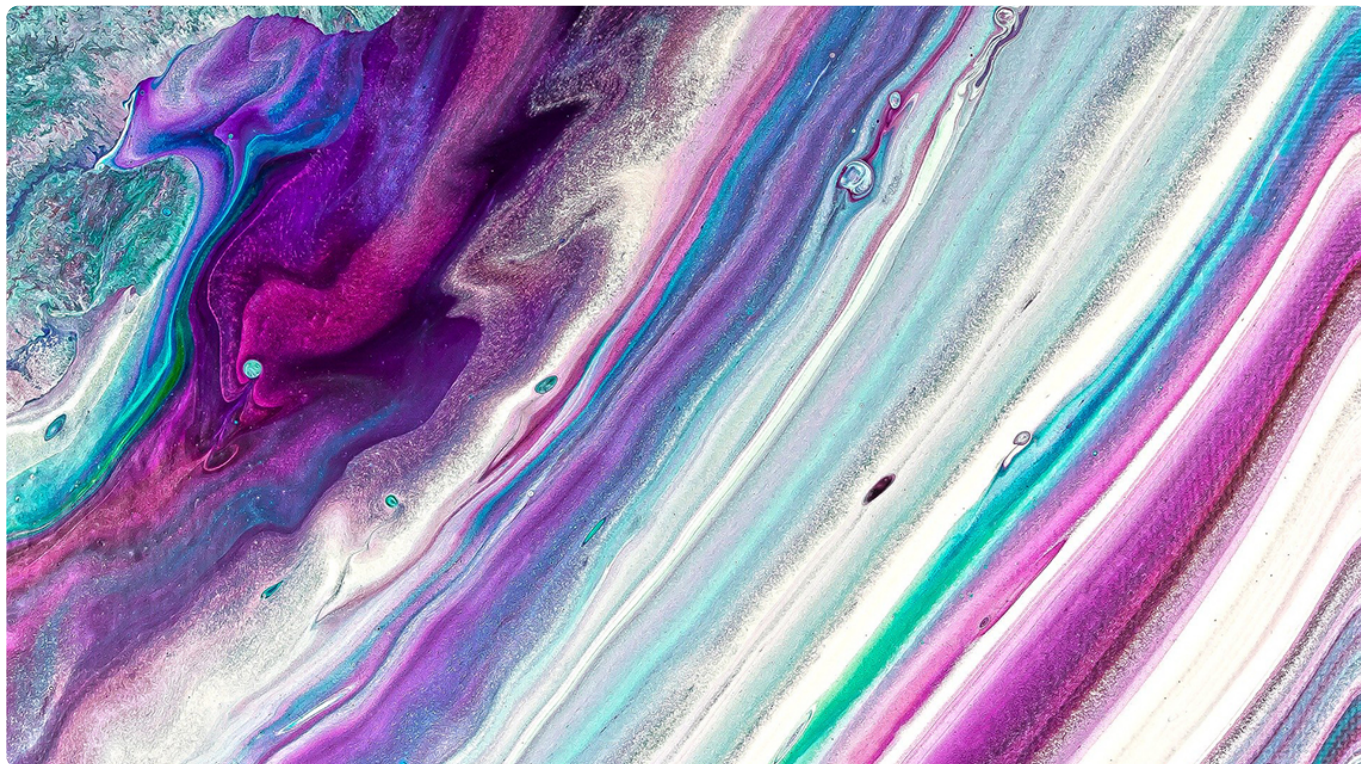


23 | MySQL：数据库级监控及常用计数器解析（下）

2020-02-12 高楼

性能测试实战30讲

[进入课程 >](#)



讲述：高楼

时长 20:37 大小 18.89M



上一篇文章中，我们讲了有关数据库的全局分析，那么在今天的文章中，我们继续看看在数据库中，如何做定向分析。

还记得我在上篇文章中提到的工具吗？mysqlreport、pt-query-digest 和 mysql_exportor+Prometheus+Grafana。我们在上一篇中已经讲完了 mysqlreport，今天来看看剩下的这几个。

定向抓取 SQL：pt-query-digest




pt-query-digest是个挺好的工具，它可以分析slow log、general log、binary log，还能分析 tcpdump 抓取的 MySQL 协议数据，可见这个工具有多强大。pt-

query-digest属于 Percona-tool 工具集，这个 Percona 公司还出了好几个特别好使的监控 MySQL 的工具。

pt-query-digest分析 slow log 时产生的报告逻辑非常清晰，并且数据也比较完整。执行命令后就会生成一个报告。

我来稍微解释一下这个报告。我们先看这个报告的第一个部分：

 复制代码

```
1 # 88.3s user time, 2.5s system time, 18.73M rss, 2.35G vsz
2 # Current date: Thu Jun 22 11:30:02 2017
3 # Hostname: localhost
4 # Files: /Users/Zee/Downloads/log/10.21.0.30/4001/TENCENT64-slow.log.last
5 # Overall: 210.18k total, 43 unique, 0.26 QPS, 0.14x concurrency -----
6 # Time range: 2017-06-12 21:20:51 to 2017-06-22 09:26:38
7 # Attribute          total        min         max         avg         95%        stddev    median
8 # =====
9 # Exec time          118079s      100ms         9s        562ms         2s        612ms      293ms
10 # Lock time           15s           0           7ms        71us        119us        38us       69us
11 # Rows sent           1.91M          0      48.42k       9.53       23.65      140.48       2.90
12 # Rows examine       13.99G          0      3.76M     69.79k    101.89k     33.28k     68.96k
13 # Rows affecte        3.36M          0      1.98M     16.76         0.99       4.90k         0
14 # Query size          102.82M         6     10.96k     512.99     719.66     265.43     719.66
```

从上表中可以看得出来，在这个慢日志中，总执行时间达到了 118079s，平均执行时间为 562ms，最长执行时间为 9s，标准方差为 612ms。

可见在此示例中，SQL 执行还是有点慢的。


这时也许会有人问，SQL 执行多长时间才是慢呢？之前在一个金融机构，我跟一个做核心系统的团队讨论他们的 SQL 执行时间指标。他们判断之后说，希望 SQL 平均执行时间指标定在 500ms。我说，你们要 500ms，那前面还有一连串的点才能到达最终的用户，如果每个环节都这样要求自己，那最终的用户不就明显感觉到很慢了吗？

经过一轮轮的讨论，最后定在了 100ms 以内。

其实从我的经验上来看，对于大部分实时的业务，一个 SQL 执行的平均时间指标定在 100ms 都多了。但是对性能来说就是这样，在所有的环节中都没有固定的标准，只有经验

数据和不断演化的系统性能能力。


我们再接着分析上面的数据。再来看pt-query-digest给出的负载报表：

 复制代码

```
1 # Profile
2 # Rank Query ID      Response time    Calls  R/Call V/M   Item
3 # =====
4 #      1 0x6A516B681113449F 73081.7989 61.9%  76338 0.9573  0.71 UPDATE mb_trans
5 #      2 0x90194A5C40980DA7 38014.5008 32.2% 105778 0.3594  0.20 SELECT mb_trans
6 #      3 0x9B56065EE2D0A5C8  3893.9757  3.3%   9709 0.4011  0.11 UPDATE mb_finan
7 # MISC 0xMISC          3088.5453  2.6%  18353 0.1683   0.0 <40 ITEMS>
```

从这个表中可以看到，有两个 SQL 的执行时间占了总执行时间的 94%，显然这两个 SQL 是要接下来要分析的重点。

我们再接着看这个工具给出的第一个 SQL 的性能报表：

 复制代码

```
1 # Query 1: 0.30 QPS, 0.29x concurrency, ID 0x6A516B681113449F at byte 12730358!
2 # This item is included in the report because it matches --limit.
3 # Scores: V/M = 0.71
4 # Time range: 2017-06-16 21:12:05 to 2017-06-19 18:50:59
5 # Attribute      pct   total      min      max      avg      95%   stddev  median
6 # =====
7 # Count          36    76338
8 # Exec time      61   73082s   100ms     5s    957ms     2s    823ms   672ms
9 # Lock time      19      3s     20us     7ms    38us     66us    29us   33us
10 # Rows sent       0      0        0        0        0        0        0      0
11 # Rows examine   36    5.06G   3.82k 108.02k 69.57k 101.89k 22.70k 68.96k
12 # Rows affecte   2   74.55k      1      1        1        1        0      1
13 # Query size     12  12.36M    161    263   169.75  192.76   11.55  158.58
14 # String:
15 # Databases      db_bank
16 # Hosts          10.21.16.50 (38297/50%)... 1 more
17 # Users          user1
18 # Query_time distribution
19 #    1us
20 #   10us
21 #  100us
22 #    1ms
23 #   10ms
24 #  100ms #####
25 #    1s  #####
```

```

26 # 10s+
27 # Tables
28 # SHOW TABLE STATUS FROM `db_bank` LIKE 'mb_trans'\G
29 # SHOW CREATE TABLE `db_bank`.`mb_trans`\G
30 UPDATE mb_trans
31 SET
32   resCode='PCX00000',resultMes='交易成功',payTranStatus='P03',payRouteCode='CN
33 WHERE
34   seqNo='20170619PM010394356875'\G
35 # Converted for EXPLAIN
36 # EXPLAIN /*!50100 PARTITIONS*/
37 select
38   resCode='PCX00000',resultMes='交易成功',payTranStatus='P03',payRouteCode='CN
39   seqNo='20170619PM010394356875'\G

```

从查询时间分布图上来看，这个语句的执行时间在 100ms~1s 之间居多，95% 的执行时间在 2s 以下。那么这个 SQL 就是我们接下来要调优的重点了。

第二个 SQL 我就不赘述了，因为逻辑是完全一样的。

通过对慢日志的分析，我们可以很快知道哪个 SQL 是慢的了。当然你用mysqldumpslow分析，也会得到一样的结果。

SQL 剖析：profiling

在分析数据库的性能时，显然对 SQL 的分析是绕不过去的一个环节。但是我之前也说了，上来就对 SQL 进行全面剖析也是不合逻辑的，因为 SQL 那么多，如果对每个 SQL 都进行详细的执行步骤解析，显然会拖慢整个系统，而且，对一些执行快的 SQL 进行分析也没有什么必要，徒增资源消耗。

通过上面的分析过程，我们已经定位到了具体是哪个 SQL 执行得慢，那么下面就是要知道 SQL 的执行细节。无论是在 Oracle 还是在 MySQL 中，我们都要去看执行计划。

比如说下面这样的：

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	blog	ALL					49648	Using where
2	SUBQUERY	blog_statistics	const	PRIMARY	PRIMARY	8	const	1	Using index

上图中select_type是子句类型的意思，有简单有复杂，但是它不能说明什么成本的问题。在这里，最重要的内容是 type，因为 type 可以告诉你访问这个表的时候，是通过什么样的方式访问的。上图中的 ALL 是全表扫描的意思。type 还有如下几个值：

type	含义	备注
ALL	全表扫描	没啥好备注的。
index	全索引扫描	index与ALL的区别就是走了索引，对查数据来说都是全表查。
range	索引范围扫描	如果你用了between、(<)、(>)，就会看到这个类型。
ref	非唯一性索引扫描	
eq_ref	唯一性索引扫描	像主键、唯一索引扫描就属于这种情况。
const或system	常量查询	如果将主键放到where列表，MySQL就可以将主键转化为常量。
NULL	空值	执行时不用访问表或索引。

执行计划中的possible_keys会列出可能使用到的索引值。key 这一列会列出执行时使用到的索引值。

以上信息就是 MySQL 的执行计划中比较重要的部分了。这些信息可以帮助我们做 SQL 的分析，为优化提供证据。


除了执行计划外，MySQL 还提供了profiling，这个有什么用呢？它可以把 SQL 执行的每一个步骤详细列出来，从一个 SQL 进入到数据库中，到执行完这整个生命周期。

MySQL 的profiling在session级生效，所以当你用了慢日志，知道哪个 SQL 有问题之后，再用这个功能是最见成效的。如果想一开始就把所有session的SQL profiling功能打开，那成本就太高了。

下面我来详细解释一下 profiling 的用法和功能。

profiling 操作步骤

profiling 的操作步骤比较简单，如下所示：

 复制代码

```
1 步骤一：set profiling=1; //这一步是为了打开profiling功能
2 步骤二：执行语句          //执行你从慢日志中看到的语句
3 步骤三：show profiles;     //这一步是为了查找步骤二中执行的语句的ID
4 步骤四：show profile all for query id; //这一步是为了显示出profiling的结果
```

我们实际执行一下上面的步骤：

 复制代码

```
1 // 步骤一：打开profiling功能
2 mysql> set profiling=1;
3 Query OK, 0 rows affected, 1 warning (0.00 sec)
4 // 这一步只是为了确认一下profiles列表有没有值，可以不用执行。
5 mysql> show profiles;
6 Empty set, 1 warning (0.00 sec)
7 // 步骤二：执行语句
8 mysql> select * from t_user where user_name='Zee0355916';
9 +-----+-----+-----+-----+-----+
10 | id                | user_number | user_name | org_id | ei
11 +-----+-----+-----+-----+-----+
12 | 00000d2d-32a8-11ea-91f8-00163e124cff | 00009496    | Zee0355916 | NULL    | t
13 | 77bdb1ef-32a6-11ea-91f8-00163e124cff | 00009496    | Zee0355916 | NULL    | t
14 | d4338339-32a2-11ea-91f8-00163e124cff | 00009496    | Zee0355916 | NULL    | t
15 +-----+-----+-----+-----+-----+
16 3 rows in set (14.33 sec)
17 // 步骤三：查看profiles列表中，有了我们刚才执行的语句
18 mysql> show profiles;
19 +-----+-----+-----+-----+-----+
20 | Query_ID | Duration      | Query
21 +-----+-----+-----+-----+-----+
22 | 1         | 14.34078475   | select * from t_user where user_name='Zee0355916' |
23 +-----+-----+-----+-----+-----+
24 1 row in set, 1 warning (0.00 sec)
25 // 步骤四：看这个语句的profile信息
26 mysql> show profile all for query 1;
27 +-----+-----+-----+-----+-----+
28 | Status                | Duration      | CPU_user | CPU_system | Context
29 +-----+-----+-----+-----+-----+
30 | starting              | 0.0000024    | 0.000012 | 0.000005   |
31 | Waiting for query cache lock | 0.0000004    | 0.000003 | 0.000001   |
32 | init                  | 0.0000003    | 0.000002 | 0.000001   |
33 | checking query cache for query | 0.0000052    | 0.000036 | 0.000015   |
34 | checking permissions  | 0.0000007    | 0.000005 | 0.000002   |
35 | Opening tables        | 0.0000032    | 0.000023 | 0.000009   |
36 | init                  | 0.0000042    | 0.000029 | 0.000013   |
```

```

37 | System lock | 0.000016 | 0.000011 | 0.000004 |
38 | Waiting for query cache lock | 0.000003 | 0.000002 | 0.000001 |
39 | System lock | 0.000020 | 0.000014 | 0.000006 |
40 | optimizing | 0.000012 | 0.000009 | 0.000004 |
41 | statistics | 0.000019 | 0.000013 | 0.000005 |
42 | preparing | 0.000015 | 0.000010 | 0.000005 |
43 | executing | 0.000004 | 0.000003 | 0.000001 |
44 | Sending data | 14.324781 | 4.676869 | 0.762349 |
45 | end | 0.000015 | 0.000007 | 0.000002 |
46 | query end | 0.000006 | 0.000005 | 0.000001 |
47 | closing tables | 0.000016 | 0.000013 | 0.000003 |
48 | freeing items | 0.000013 | 0.000010 | 0.000003 |
49 | Waiting for query cache lock | 0.000003 | 0.000002 | 0.000000 |
50 | freeing items | 0.000014 | 0.000012 | 0.000003 |
51 | Waiting for query cache lock | 0.000003 | 0.000002 | 0.000000 |
52 | freeing items | 0.000003 | 0.000002 | 0.000001 |
53 | storing result in query cache | 0.000004 | 0.000002 | 0.000000 |
54 | logging slow query | 0.015645 | 0.000084 | 0.000020 |
55 | cleaning up | 0.000034 | 0.000024 | 0.000006 |
56 +-----+-----+-----+-----+
57 26 rows in set, 1 warning (0.02 sec)

```

非常长，从这样的数据中，我们就看到了一个语句在数据库中从开始到结束的整个生命周期。

对生命周期中的每个步骤进行统计之后，我们就可以看到每个步骤所消耗的时间。不仅如此，还能看到如下这些信息：

BLOCK IO

Context Switches

CPU

IPC

MEMORY

Page Fault


SOURCE

SWAPS

有了这些信息，我们基本上就可以判断语句哪里有问题了。


从上面这个示例语句中，你可以看到Sending data这一步消耗了 14 秒的时间，并且从后面的数据中，也可以看到主动上下文切换有 1316 次，被动的有 132 次，块操作的量也非常大。

碰到这样的情况，我们就得先知道这个Sending data到底是什么东西。下面我们结合之前说的到的执行计划，一起看一下：

 复制代码

```
1 mysql> explain select * from t_user where user_name='Zee0355916';
2 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 | id | select_type | table | type | possible_keys | key | key_len | ref | rows |
4 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
5 | 1 | SIMPLE | t_user | ALL | NULL | NULL | NULL | NULL | 3 |
6 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
7 1 row in set (0.00 sec)
```

这就是个典型的全表扫描，所以下一步就是检查有没有创建索引。

 复制代码

```
1 mysql> show indexes from t_user;
2 +-----+-----+-----+-----+-----+-----+-----+-----+
3 | Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality |
4 +-----+-----+-----+-----+-----+-----+-----+-----+
5 | t_user | 0 | PRIMARY | 1 | id | A | 1 |
6 +-----+-----+-----+-----+-----+-----+-----+-----+
7 1 row in set (0.00 sec)
8
9
10 mysql>
```

还是有一个主键索引的，但由于我们没用主键来查，所以用不到。

有些性能测试工程师面对这种情况可能会有这种想法：第一次没有查索引，但是把所有数据都调到缓存里了呀，所以第二次就快了嘛，于是有些人可能想尽快“完成”工作，就用重复的数据。

这里我再执行一遍，你可以看看是什么结果：



```
1 +-----+-----+-----+-----+
2 | Query_ID | Duration   | Query
3 +-----+-----+-----+-----+
4 |          1 | 14.34078475 | select * from t_user where user_name='Zee0355916'
5 |          2 | 0.00006675 | show profile all for 1
6 |          3 | 0.00031700 | explain select * from t_user where user_name='Zee03!
7 |          4 | 0.00040025 | show indexes from t_user
8 +-----+-----+-----+-----+
9 6 rows in set, 1 warning (0.00 sec)
10
11
12 mysql> select * from t_user where user_name='Zee0355916';
13 +-----+-----+-----+-----+-----+-----+
14 | id | user_number | user_name | org_id | ei
15 +-----+-----+-----+-----+-----+-----+
16 | 00000d2d-32a8-11ea-91f8-00163e124cff | 00009496 | Zee0355916 | NULL | t
17 | 77bdb1ef-32a6-11ea-91f8-00163e124cff | 00009496 | Zee0355916 | NULL | t
18 | d4338339-32a2-11ea-91f8-00163e124cff | 00009496 | Zee0355916 | NULL | t
19 +-----+-----+-----+-----+-----+-----+
20 3 rows in set (0.00 sec)
21
22
23 mysql> show profiles;
24 +-----+-----+-----+-----+-----+-----+
25 | Query_ID | Duration   | Query
26 +-----+-----+-----+-----+-----+-----+
27 |          1 | 14.34078475 | select * from t_user where user_name='Zee0355916'
28 |          2 | 0.00006675 | show profile all for 1
29 |          3 | 0.00031700 | explain select * from t_user where user_name='Zee03!
30 |          4 | 0.00040025 | show indexes from t_user
31 |          5 | 0.00027325 | select * from t_user where user_name='Zee0355916'
32 +-----+-----+-----+-----+-----+-----+
33 7 rows in set, 1 warning (0.00 sec)
34
35
36 mysql> show profile all for query 5;
37 +-----+-----+-----+-----+-----+-----+
38 | Status | Duration | CPU_user | CPU_system | Context_
39 +-----+-----+-----+-----+-----+-----+
40 | starting | 0.000029 | 0.000018 | 0.000004 |
41 | Waiting for query cache lock | 0.000006 | 0.000003 | 0.000001 |
42 | init | 0.000003 | 0.000003 | 0.000000 |
43 | checking query cache for query | 0.000008 | 0.000006 | 0.000002 |
44 | checking privileges on cached | 0.000003 | 0.000002 | 0.000000 |
45 | checking permissions | 0.000010 | 0.000192 | 0.000000 |
46 | sending cached result to clien | 0.000210 | 0.000028 | 0.000000 |
47 | cleaning up | 0.000006 | 0.000006 | 0.000000 |
48 +-----+-----+-----+-----+-----+-----+
49 8 rows in set, 1 warning (0.00 sec)
50
51
```

```
52 mys
53
```

看到没有，在用重复数据的时候确实会让响应时间快很多，因为数据直接从cache中发给client了。

但是，这种作法请你坚决制止，因为它不符合真实生产的样子。当你再换一个数据的时候，就会歇菜，还要再经过 14 秒的时间做全表扫描。

所以正确的做法是创建合适的索引，让语句在执行任何一条数据时都能快起来，那么，我们现在就创建一个索引，再看执行结果。

 复制代码

```
1 // 创建索引
2 mysql> ALTER TABLE t_user ADD INDEX username_idx (user_name);
3 Query OK, 0 rows affected (44.69 sec)
4 Records: 0 Duplicates: 0 Warnings: 0
5 // 分析表
6 mysql> analyze table t_user;
7 +-----+-----+-----+-----+
8 | Table      | Op      | Msg_type | Msg_text |
9 +-----+-----+-----+-----+
10 | pa.t_user | analyze | status   | OK       |
11 +-----+-----+-----+-----+
12 1 row in set (0.08 sec)
13 // 执行语句
14 mysql> select * from t_user where user_name='Zee0046948';
15 +-----+-----+-----+-----+-----+
16 | id                | user_number | user_name | org_id | ei
17 +-----+-----+-----+-----+-----+
18 | 000061a2-31c2-11ea-8d89-00163e124cff | 00009496    | Zee0046948 | NULL   | ti
19 | 047d7ae1-32a2-11ea-91f8-00163e124cff | 00009496    | Zee0046948 | NULL   | ti
20 | 1abfa543-318f-11ea-8d89-00163e124cff | 00009496    | Zee0046948 | NULL   | ti
21 | 671c4014-3222-11ea-91f8-00163e124cff | 00009496    | Zee0046948 | NULL   | ti
22 | 9de16dd3-32a5-11ea-91f8-00163e124cff | 00009496    | Zee0046948 | NULL   | ti
23 | dd4ab182-32a4-11ea-91f8-00163e124cff | 00009496    | Zee0046948 | NULL   | ti
24 | f507067e-32a6-11ea-91f8-00163e124cff | 00009496    | Zee0046948 | NULL   | ti
25 | f7b82744-3185-11ea-8d89-00163e124cff | 00009496    | Zee0046948 | NULL   | ti
26 +-----+-----+-----+-----+-----+
27 8 rows in set (0.02 sec)
28 // 查看Query_ID
29 mysql> show profiles;
30 +-----+-----+-----+
31 | Query_ID | Duration | Query
32 +-----+-----+-----+
```

```

33 |      1 | 14.34078475 | select * from t_user where user_name='Zee0355916'
34 |      2 | 0.00006675 | show profile all for 1
35 |      3 | 0.00031700 | explain select * from t_user where user_name='Zee03!
36 |      4 | 0.00005875 | show indexes for table t_user
37 |      5 | 0.00005850 | show indexes for t_user
38 |      6 | 0.00040025 | show indexes from t_user
39 |      7 | 0.00027325 | select * from t_user where user_name='Zee0355916'
40 |      8 | 0.00032100 | explain select * from t_user where user_name='Zee03!
41 |      9 | 12.22490550 | select * from t_user where user_name='Zee0046945'
42 |     10 | 0.00112450 | select * from t_user limit 20
43 |     11 | 44.68370500 | ALTER TABLE t_user ADD INDEX username_idx (user_nam
44 |     12 | 0.07385150 | analyze table t_user
45 |     13 | 0.01516450 | select * from t_user where user_name='Zee0046948'
46 +-----+-----+-----+-----+-----+
47 13 rows in set, 1 warning (0.00 sec)
48 // 查看profile信息
49 mysql> show profile all for query 13;
50 +-----+-----+-----+-----+-----+
51 | Status | Duration | CPU_user | CPU_system | Context_
52 +-----+-----+-----+-----+-----+
53 | starting | 0.000030 | 0.000017 | 0.000004 |
54 | Waiting for query cache lock | 0.000005 | 0.000004 | 0.000001 |
55 | init | 0.000003 | 0.000002 | 0.000000 |
56 | checking query cache for query | 0.000060 | 0.000050 | 0.000011 |
57 | checking permissions | 0.000009 | 0.000007 | 0.000002 |
58 | Opening tables | 0.000671 | 0.000412 | 0.000000 |
59 | init | 0.006018 | 0.000082 | 0.000899 |
60 | System lock | 0.000017 | 0.000011 | 0.000003 |
61 | Waiting for query cache lock | 0.000003 | 0.000003 | 0.000000 |
62 | System lock | 0.000019 | 0.000015 | 0.000004 |
63 | optimizing | 0.000012 | 0.000010 | 0.000002 |
64 | statistics | 0.001432 | 0.000167 | 0.000037 |
65 | preparing | 0.000026 | 0.000043 | 0.000009 |
66 | executing | 0.000034 | 0.000005 | 0.000001 |
67 | Sending data | 0.006727 | 0.000439 | 0.001111 |
68 | end | 0.000014 | 0.000007 | 0.000002 |
69 | query end | 0.000009 | 0.000008 | 0.000001 |
70 | closing tables | 0.000015 | 0.000012 | 0.000003 |
71 | freeing items | 0.000010 | 0.000008 | 0.000002 |
72 | Waiting for query cache lock | 0.000003 | 0.000002 | 0.000000 |
73 | freeing items | 0.000027 | 0.000022 | 0.000005 |
74 | Waiting for query cache lock | 0.000003 | 0.000002 | 0.000001 |
75 | freeing items | 0.000003 | 0.000002 | 0.000000 |
76 | storing result in query cache | 0.000004 | 0.000004 | 0.000001 |
77 | cleaning up | 0.000015 | 0.000012 | 0.000003 |
78 +-----+-----+-----+-----+-----+
79 25 rows in set, 1 warning (0.01 sec)
80
81
82 mysql>

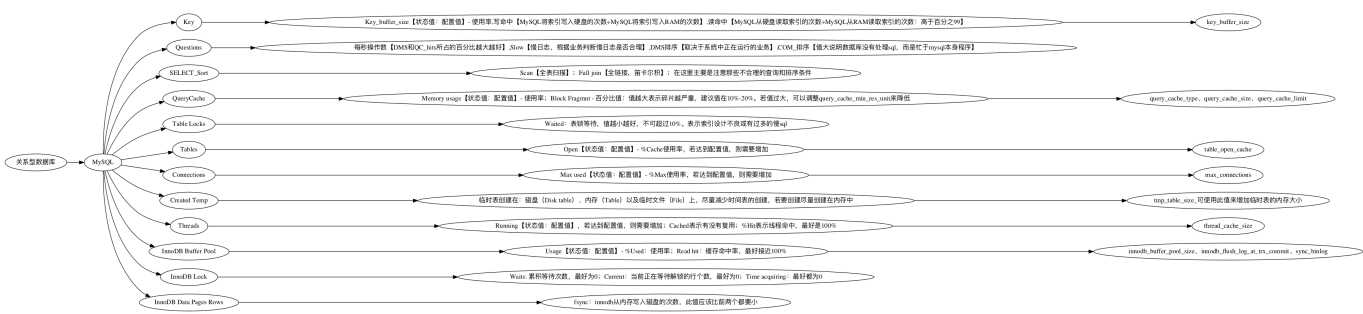
```

从上面最后的 profile 信息你可以看出来，步骤一点没少，但是速度快了很多，这才是正确的优化思路。

在上一篇文章中，我描述了在一个数据库中，如何从全局监控的角度查看数据，今天讲的是如何找到具体慢的 SQL，以及如何定位这个 SQL 的问题。

当然不是所有的情况下，都是 SQL 的问题，也有可能是配置的问题，也有可能是硬件的问题。不管什么样的问题，其分析思路都是这样的，也就是我总是在强调的：全局监控 - 定向监控。

当然，在这里我也应该给出 MySQL 分析决策树的思路。从mysqlreport的划分上，给出几个具体的分析决策树的树枝。



这是常见的问题，如果你有兴趣，可以自己完善这棵完整的树，因为你可能会有不一样的划分计数器的工具或思路，所以这个树是可以灵活变化的。

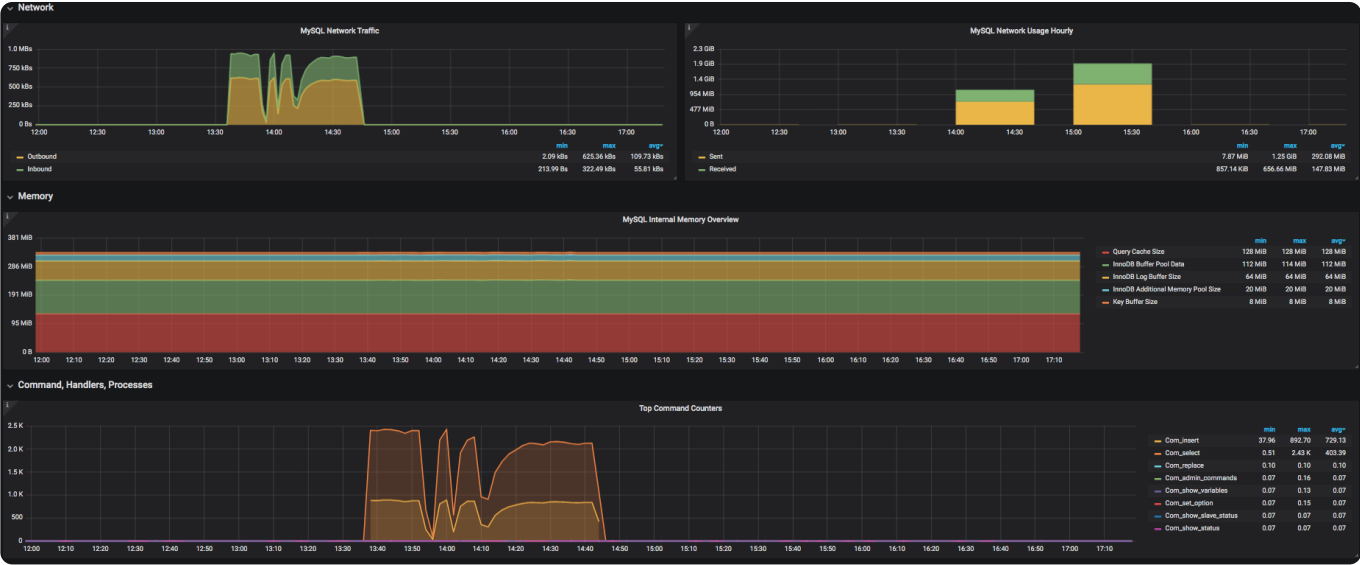
你一定要记得，别人给你的东西，永远变不成自己的东西，它们只能引导你。如果你自己动手去做一遍，哪怕只画出一个分枝来，都会是很大的进步。

如果你想用其他的全局监控工具，也可以考虑如下的组合，也就是mysql_exportor+Prometheus+Grafana。

mysql_exportor+Prometheus+Grafana

我在前面也屡次提到过这类组合，不同的 exporters 结合 Prometheus+Grafana，可以实现实时监控及数据的保存。

在这里我们看一下mysql_exportor可以给我们提供什么样的监控数据。这里截几个图，给你大概看一下这个套装工具能看什么内容，有兴趣的话，你也可以自己搭建一下。



总结

有关数据库的知识实在是太多了，在这两篇文章中，我重点想告诉你的，就是分析数据库应该具有的思路。至于其他的知识点，我想应该是你打开文章之前就应该储备的东西。

我们再来总结一下，在数据库的分析中，最有可能在三个方面出现问题：

1. 硬件配置
2. 数据库配置
3. SQL 语句

对于硬件配置来说，我们只能在解决了 2 和 3 的问题之后，再来评估到底多少硬件够用的。而面对数据库配置问题，这个实在没什么好招，只能去了解数据库架构等一系列的知识之后，再学着解决。而 SQL 的问题呢，应该说是我们在性能测试和分析中最常见的了。SQL 性能问题的分析思路也比较清晰，那就是判断出具体的 SQL 瓶颈点，进而做相应的优化，切记不要蒙！

现在的数据库类别比之前多太多了，每种数据库都有自己的架构和使用场景，我们要在充分了解了之后，才能下手去调。

思考题

我在这里照例留两个问题。你能说一下数据库分析的大体思路是什么吗？如何在数据库中迅速找到一个慢 SQL 的根本原因呢？

欢迎你在评论区写下你的思考，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。

关注极客时间服务号 每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 22 | MySQL：数据库级监控及常用计数器解析（上）

下一篇 24 | Kafka：性能监控工具之队列级监控及常用计数器解析

精选留言 (3)

写留言



Geek_f93234

2020-02-14

数据库分析的大体思路是什么吗？

全局分析--定向分析

1.全局分析：分析数据库硬件配置，数据库配置，SQL语句，采用全局监控工具如mysqlre
port工具收集到的测试数据，分析可能存在的问题；

2.定向分析：如：针慢查询导致的性能问题，采用pt-query-digest工具分析慢查询日志...

展开

作者回复：抓住重点了。



1



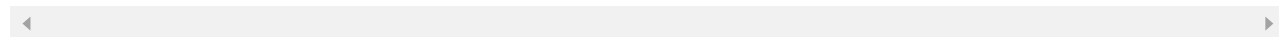
张红占

2020-02-14

干货太多了，课程价格严重低估了！

展开 ▾

作者回复: 多谢支持，后面要不我写点湿的。 😊



夜空中最亮的星（华仔...

2020-02-13

好好在家呆着学习 那也不去

展开 ▾

