

## 22 | MySQL：数据库级监控及常用计数器解析（上）

2020-02-10 高楼

性能测试实战30讲

[进入课程 >](#)



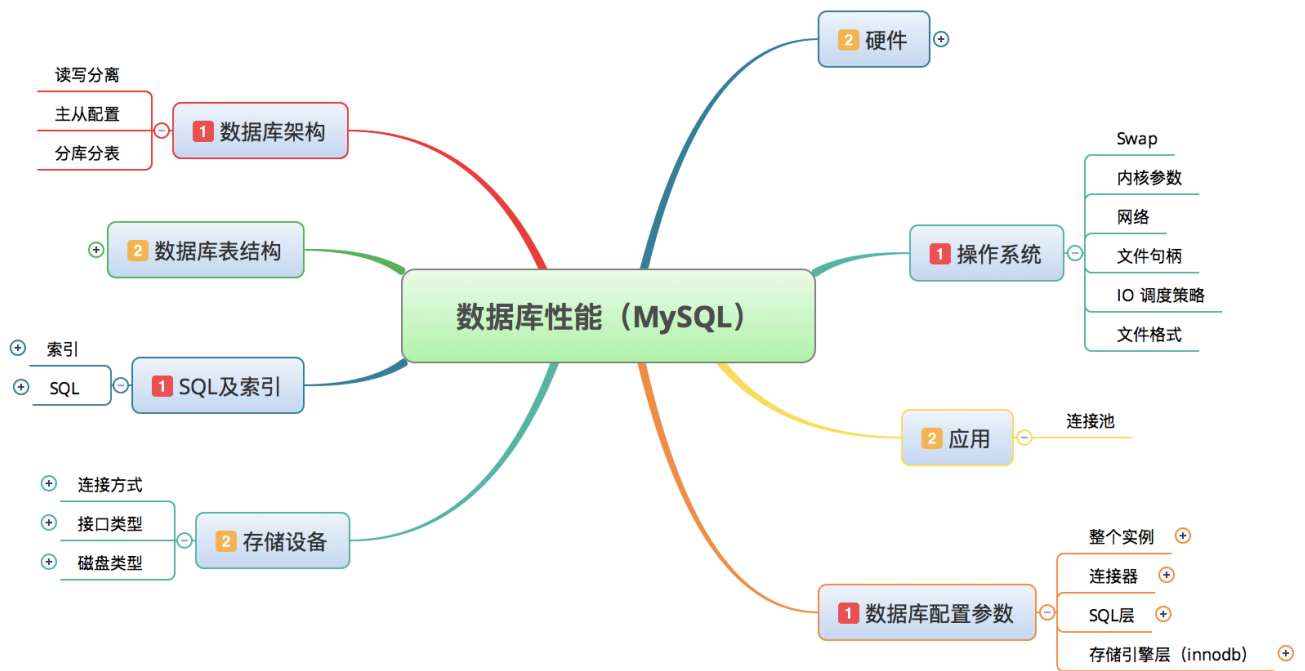
**讲述：高楼**

时长 29:21 大小 26.89M



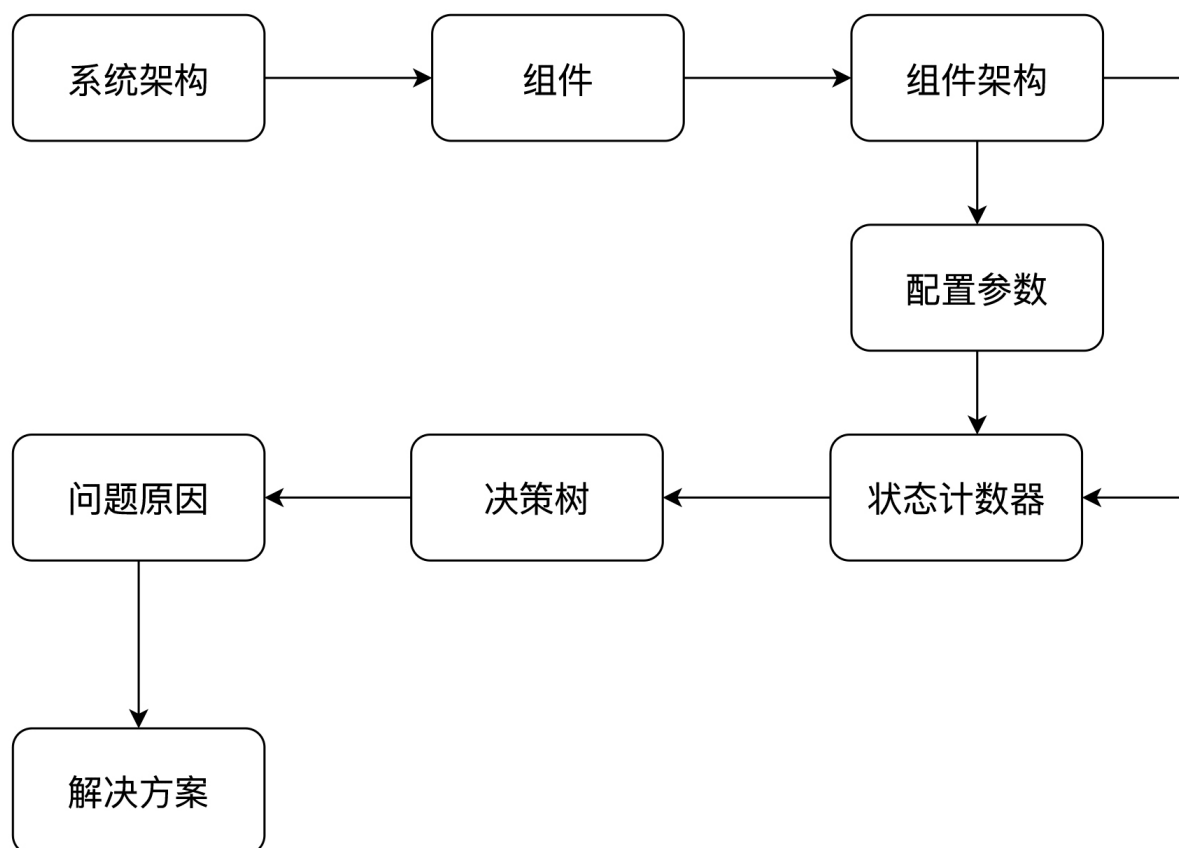
数据库是一个非常大的话题，我们在很多地方，都会看到对数据库的性能分析会包括以下部分。





但其实呢，以上这些内容都是我们应该具备的基础知识，所以我今天要讲的就是，具备了这些基础知识之后我们应该干什么事情。

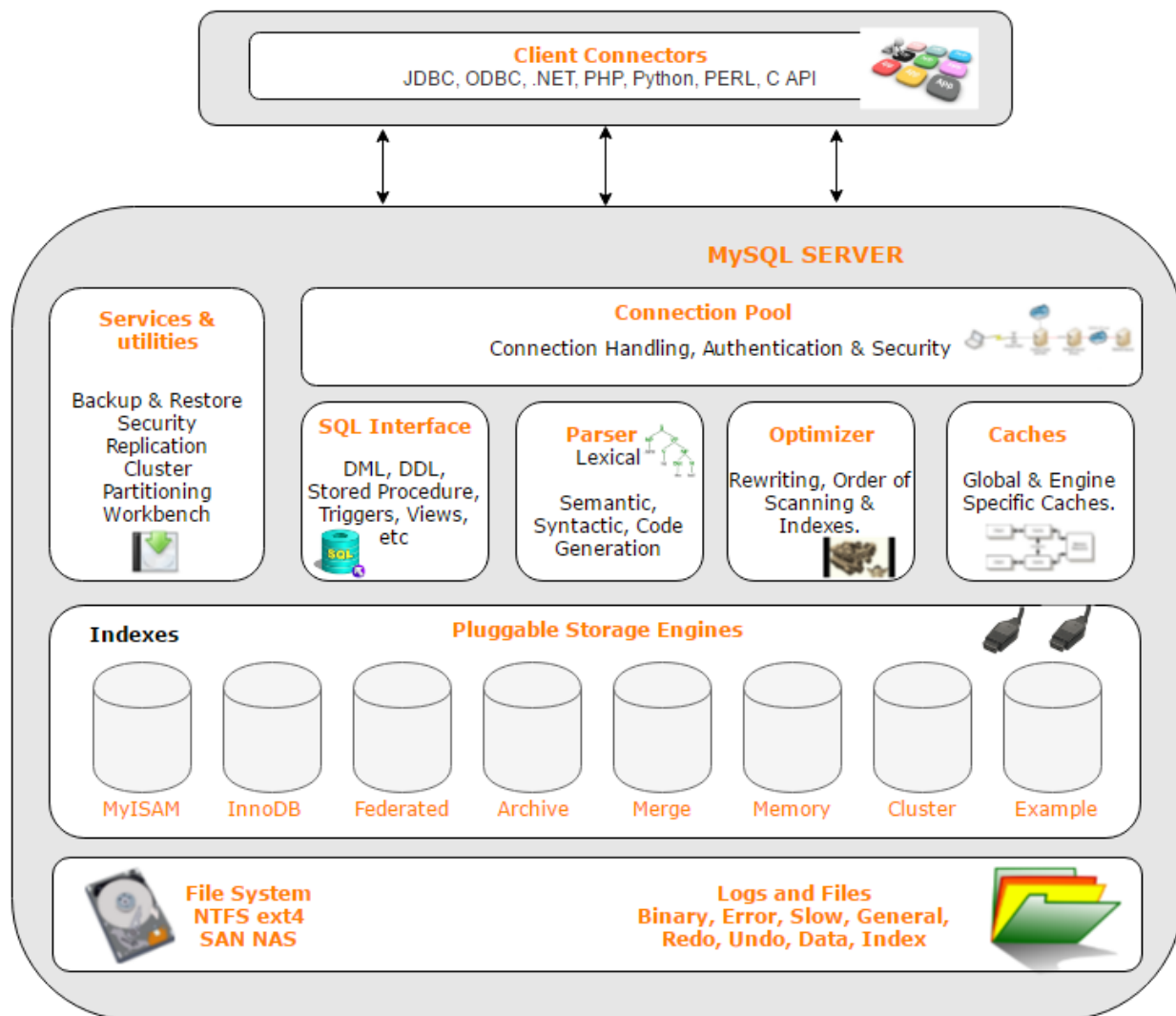
也就是说，从性能瓶颈判断分析的角度入手，才是性能从业人员该有的逻辑。每次我分析一个性能问题时，逻辑总是这样的：



1. 先画出整个系统的架构图。
2. 列出整个系统中用到了哪些组件。这一步要确定用哪些监控工具来收集数据，具体的内容你可以看下之前讲到的监控设计相关的内容。
3. 掌握每个组件的架构图。在这一步中需要列出它们的关键性能配置参数。
4. 在压力场景执行的过程中收集状态计数器。
5. 通过分析思路画出性能瓶颈的分析决策树。
6. 找到问题的根本原因。
7. 提出解决方案并评估每个方案的优缺点和成本。

这是我一直强调的分析决策树的创建逻辑。有了这些步骤之后，即使不熟悉一个系统，你也可以进行性能分析。

对于 MySQL 数据库来说，我们想对它进行分析，同样也需要看它的架构图。如下图所示（这是 MySQL5 版本的架构示意图）：



这里就有一个问题了：看架构图是看什么？这个图够细吗？

首先，看架构图，一开始肯定是看大而全的架构。比如说上图，我们知道了，MySQL 中有 Connection Pool、SQL Interface、Parser 等这些大的模块。

其次，我们得知道这些模块的功能及运行逻辑。比如说，我们看到了这些模块之后，需要知道，当一个 SQL 通过 Connection Pool 进到系统之后，需要先进入 SQL Interface 模块判断这个语句，知道它是一个什么样的 SQL，涉及到了什么内容；然后通过 Parser 模块进行语法语义检查，并生成相应的执行计划；接着到 Optimizer 模块进行优化，判断走什么索引，执行顺序之类的；然后就到 Caches 中找数据，如果在 Caches 中找不到数据的话，就得通过文件系统到磁盘中找。

这就是一个大体的逻辑。但是知道了这个逻辑还不够。还记得前面我们说的对一个组件进行“全局一定向”的监控思路吧。

这里我们还得找工具实现对 MySQL 的监控，还好 MySQL 的监控工具非常多。

在讲 MySQL 的监控工具之前，我们先来了解下 MySQL 中的两个 Schema，分别是 `information_schema` 和 `performance_schema`。

为什么呢？

`information_schema` 保存了数据库中的所有表、列、索引、权限、配置参数、状态参数等信息。像我们常执行的 `show processlist;` 就来自于这个 schema 中的 `processlist` 表。

`performance_schema` 提供了数据库运行时的资源消耗情况，它以较低的代价收集信息，可以提供不少性能数据。

所以这两个 Schema 对我们来说就非常重要了。

你没事的时候，也可以查一下它们相关的各个表，一个个看着玩。监控工具中的很多数据来自于它们。

还有两个命令是你在分析 MySQL 时一定要学会的：`SHOW GLOBAL VARIABLES;` 和 `SHOW GLOBAL status;`。前一个用来查看配置的参数值，后一个用来查询状态值。当你没有其他工具可用的时候，就可以用这两个命令的输出结果来分析。对于全局监控来说，这两个命令绝对够用。

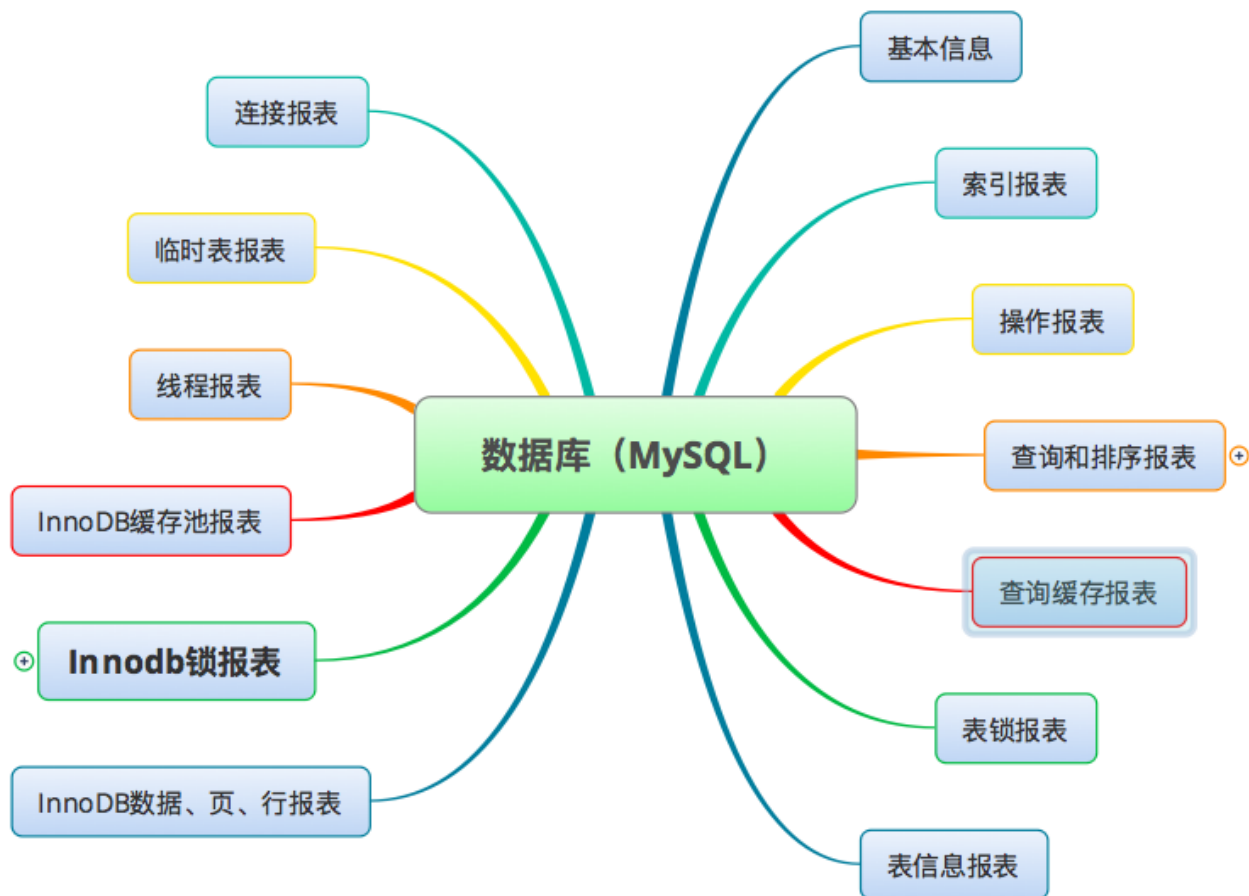
对于 MySQL 的监控工具有很多，但我主要讲的是以下几个工具：

`mysqlreport`、`pt-query-digest`、`mysql_exportor`+`Prometheus`+`Grafana`。

今天我们先来说一下 `mysqlreport`。

## 全局分析：mysqlreport

这个工具执行之后会生成一个文本文件，在这个文本文件中包括了如下这些内容。



我觉得这个工具是属于既不浪费资源，又能全局监控 MySQL 的很好的工具。

在我们执行性能场景时，如果想让 mysqlreport 抓取到的数据更为准确，可以先重启一下数据库。如果你觉得重启数据库这个动作实在是有点大，可以先把状态计数器、打开表、查询缓存等数据给刷新一下。

我认为 mysqlreport 有一些重要的知识点需要你知道，在这里我找一个例子给你解释一下。

## 索引报表

 复制代码

```
1  _ Key -----
2  Buffer used    5.00k of 8.00M %Used:    0.06
3    Current     1.46M          %Usage:  18.24
```

请注意，这里所指的 Key Buffer 是指 MyISAM 引擎使用的 Shared Key Buffer，InnoDB 所使用的 Key Buffer 不在此处统计。

从上面的数据来看，MySQL 每次分配的Key Buffer最大是 5K，占 8M 的 0.06%，还是很小的。下一行中的数据可以看到的是当前只用了 1.46M，占 8M 的 18.24%。

显然这个 Key Buffer 是够用的，如果这个使用率高，你就得增加key\_buffer\_size的值了。

### 操作报表

复制代码

1	__ Questions -----									
2	Total	126.82M	32.5/s							
3	+Unknown	72.29M	18.5/s	%Total:	57.00					
4	Com_	27.63M	7.1/s		21.79					
5	DMS	26.81M	6.9/s		21.14					
6	COM_QUIT	45.30k	0.0/s		0.04					
7	QC Hits	38.18k	0.0/s		0.03					
8	Slow 2 s	6.21M	1.6/s		4.90	%DMS:	23.17	Log:		
9	DMS	26.81M	6.9/s		21.14					
10	SELECT	20.73M	5.3/s		16.34		77.30			
11	INSERT	3.68M	0.9/s		2.90		13.71			
12	UPDATE	1.43M	0.4/s		1.13		5.33			
13	DELETE	983.11k	0.3/s		0.78		3.67			
14	REPLACE	0	0/s		0.00		0.00			
15	Com_	27.63M	7.1/s		21.79					
16	admin_comma	11.86M	3.0/s		9.35					
17	set_option	10.40M	2.7/s		8.20					
18	commit	5.15M	1.3/s		4.06					

从这个数据可以看到的信息量就有点大了，它可以反应出来这个数据库现在忙不忙。

从 32.5 每秒的操作量上来说，还是有点忙的。你还可以看到下面有操作数的细分，其实我不太愿意看下面的这些细分，描述上除了QC Hits和DMS的意思比较清晰之外，其他的几个值理解起来比较费劲。我也不建议你看下面那几个，因为它们对性能分析来说没起到什么正向的作用。

而 Slow 那这一行就很重要了，从这行可以看出slow log的时间是设置为 2 秒的，并且每秒还出现 1.6 个的慢日志，可见这个系统的 SQL 的慢日志实在是有点多。

DMS部分可以告诉我们这个数据库中各种 SQL 所占的比例。其实它是具有指向性的，像我们的这个例子中，显然是SELECT多，那如果要做 SQL 优化的话，肯定优先考虑SELECT的

语句，才会起到立竿见影的效果。

## 查询和排序报表

复制代码

1	__ SELECT and Sort -----			
2	Scan	7.88M	2.0/s	%SELECT: 38.04
3	Range	237.84k	0.1/s	1.15
4	Full join	5.97M	1.5/s	28.81
5	Range check	913.25k	0.2/s	4.41
6	Full rng join	18.47k	0.0/s	0.09
7	Sort scan	737.86k	0.2/s	
8	Sort range	56.13k	0.0/s	
9	Sort mrg pass	282.65k	0.1/s	

这个报表具有着绝对的问题指向性。这里的Scan（全表扫描）和Full join（联合全表扫描）在场景执行过程中实在是太多了，这显然是 SQL 写得有问题。

Range 范围查询很正常，本来就应该多。

## 查询缓存报表

复制代码

1	__ Query Cache -----			
2	Memory usage	646.11k of	1.00M	%Used: 63.10
3	Block Fragmnt	14.95%		
4	Hits	38.18k	0.0/s	
5	Inserts	1.53k	0.0/s	
6	Insrt:Prune	2.25:1	0.0/s	
7	Hit:Insert	24.94:1		

在这部分中，我们看的关键点是，Query Cache没用！因为各种query都没有缓存下来。同时这里我们还要看一个关键值，那就是Block Fragment，它是表明Query Cache碎片的，值越高，则说明问题越大。

如果你看到下面这样的数据，就明显没有任何问题。

复制代码

1	__ Query Cache -----			
---	----------------------	--	--	--



```

2 Memory usage    38.05M of 256.00M %Used: 14.86
3 Block Fragmnt   4.29%
4 Hits           12.74k    33.3/s
5 Inserts        58.21k    152.4/s
6 Insrt:Prune    58.21k:1    152.4/s
7 Hit:Insert      0.22:1

```

这个数据明显看到缓存了挺多的数据。Hits 这一行指的是每秒有多少个 SELECT 语句从 Query Cache中取到了数据，这个值是越大越好。

而通过Insrt:Prune的比值数据，我们可以看到 Insert 远远大于 Prune（每秒删除的 Query Cache碎片），这个比值越大就说明Query Cache越稳定。如果这个值接近 1: 1 那才有问题，这个时候就要加大Query Cache或修改你的 SQL 了。


而通过下面的Hit:Insert的值，我们可以看出命中要少于插入数，说明插入的比查询的还要多，这时就要去看这个性能场景中是不是全是插入了。如果我们查看了，发现 SELECT 语句还是很多的，而这个比值又是 Hit 少，那么我们的场景中使用的数据应该并不是插入的数据。其实在性能场景的执行过程中经常这样。所以在性能分析的过程中，我们只要知道这个值就可以了，并不能说明Query Cache就是无效的了。

## 表信息报表

```

1 __ Table Locks -----
2 Waited          0          0/s %Total: 0.00
3 Immediate      996        0.0/s
4
5
6 __ Tables -----
7 Open           2000 of 2000 %Cache: 100.00
8 Opened        15.99M      4.1/s
9

```


 复制代码

这个很明显了，表锁倒是不存在。但是你看现在table\_open\_cache已经达到上限了，设置为 2000，而现在已经达到了 2000，同时每秒打开表 4.1 个。

这些数据说明了什么呢？首先打开的表肯定是挺多的了，因为达到上限了嘛。这时候你会自然而然地想到去调table\_open\_cache参数。但是我建议你调之前先分析下其他的部分，

如果在这个性能场景中，MySQL 的整体负载就会比较高，同时也并没有报错，那么我不建议你调这个值。如果负载不高，那再去调它。


## 连接报表和临时表

 复制代码

```
1  __ Connections -----
2  Max used          521 of 2000      %Max:  26.05
3  Total            45.30k      0.0/s
4
5
6  __ Created Temp -----
7  Disk table       399.77k      0.1/s
8  Table            5.81M      1.5/s      Size:  16.0M
9  File             2.13k      0.0/s
```

这个数据连接还完全够用，但是从临时表创建在磁盘（Disk table）和临时文件（File）上的量级来说，还是有点偏大了，所以，可以增大tmp\_table\_size。

## 线程报表

 复制代码

```
1  __ Threads -----
2  Running           45 of 79
3  Cached            9 of 28      %Hit:  72.35
4  Created          12.53k      0.0/s
5  Slow              0      0/s
6
7
8  __ Aborted -----
9  Clients           0      0/s
10 Connects          7      0.0/s
11
12
13 __ Bytes -----
14 Sent              143.98G    36.9k/s
15 Received           21.03G    5.4k/
```

当 Running 的线程数超过配置值时，就需要增加thread\_cache\_size。但是从这里来看，并没有超过，当前配置了 79，只用到了 45。而这里 Cached 的命中%Hit是越大越好，我们通常都希望在 99% 以上。

# InnoDB 缓存池报表

复制代码

1	__ InnoDB Buffer Pool -----			
2	Usage	1.87G of 4.00G	%Used:	46.76
3	Read hit	100.00%		
4	Pages			
5	Free	139.55k	%Total:	53.24
6	Data	122.16k	46.60 %Drty:	0.00
7	Misc	403	0.15	
8	Latched		0.00	
9	Reads	179.59G 46.0k/s		
10	From file	21.11k 0.0/s	0.00	
11	Ahead Rnd	0 0/s		
12	Ahead Sql	0/s		
13	Writes	54.00M 13.8/s		
14	Flushes	3.16M 0.8/s		
15	Wait Free	0 0/s		

这个部分对 MySQL 来说是很重要的，innodb\_buffer\_pool\_size为 4G，它会存储表数据、索引数据等。通常在网上或书籍里，你能看到有人建议将这个值设置为物理内存的 50%，当然这个值没有绝对的，还要在具体的应用场景中测试才能知道。

这里的Read hit达到 100%，这很好。

下面还有些其他的读写数据，这部分的数据将和我们在操作系统上看到的 I/O 有很大关系。有些时候，由于写入的过多，导致操作系统的I/O wait很高的时候，我们不得不设置innodb\_flush\_log\_at\_trx\_commit参数（0：延迟写，实时刷；1：实时写，实时刷；2：实时写，延迟刷）和sync\_binlog 参数（0：写入系统缓存，而不刷到磁盘；1：同步写入磁盘；N：写 N 次系统缓存后执行一次刷新操作）来降低写入磁盘的频率，但是这样做的风险就是当系统崩溃时会有数据的丢失。

这其实是我们做测试时，存储性能不高的时候常用的一种手段，为了让 TPS 更高一些。但是，你一定要知道生产环境中的存储是什么样的能力，以确定在生产环境中应该如何配置这个参数。

# InnoDB 锁报表

复制代码


1	__ InnoDB Lock -----			
---	----------------------	--	--	--

```
2 Waits                227829      0.1/s
3 Current              1
4 Time acquiring
5 Total               171855224 ms
6 Average              754 ms
7 Max                 6143 ms
```

这个信息就有意思了。显然在这个例子中，锁的次数太多了，并且锁的时间都还不短，平均时间都能达到 754ms，这显然是不能接受的。

那就会有人问了，锁次数和锁的平均时间多少才是正常呢？在我的经验中，锁平均时间最好接近零。锁次数可以有，这个值是累加的，所以数据库启动时间长，用得多了，锁次数就会增加。

## InnoDB 其他信息

 复制代码

```
1 __ InnoDB Data, Pages, Rows -----
2 Data
3 Reads          35.74k      0.0/s
4 Writes         6.35M      1.6/s
5 fsync          4.05M      1.0/s
6 Pending
7 Reads          0
8 Writes         0
9 fsync          0
10
11
12 Pages
13 Created       87.55k      0.0/s
14 Read          34.61k      0.0/s
15 Written       3.19M      0.8/s
16
17
18 Rows
19 Deleted       707.46k     0.2/s
20 Inserted      257.12M     65.9/s
21 Read          137.86G    35.3k/s
22 Updated       1.13M      0.3/
```

这里的数据可以明确告诉你的一点是，在这个性能场景中，插入占有着绝对的量级。

## 总结

好了，我们拿一个 mysqlreport 报表从上到下看了一遍之后，你是不是觉得对 MySQL 有点感觉了？这里我给一个结论性的描述吧：

1. 在这个性能场景中，慢日志太多了，需要定向监控看慢 SQL，找到慢 SQL 的执行计划。
2. 在这个插入多的场景中，锁等待太多，并且等待的时候又太长，解决慢 SQL 之后，这里可能会解决，但还是要分析具体的原因的，所以这里也是指向了 SQL。

这里为什么要描述得这么细致呢？主要是因为当你看其他一些工具的监控数据时，分析思路是可以共用的。

但是有人说这里还有一个问题：SQL 怎么看？

其实对于我们分析的逻辑来说，在数据库中看 SQL 就是在做定向的分析了。请你不要相信一些人所吹嘘的那样，一开始就把所有的 SQL 执行时间统计出来，这真的是完全没有必要的做法。因为成本太高了。

在下一篇文章里，我们换个工具来看看 SQL 的执行时间到底应该怎么分析。

## 思考题

最后给你留两道思考题吧，MySQL 中全局监控工具可以给我们提供哪些信息？以及，如何判断 MySQL 状态值和配置值之间的关系呢？

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。

点击查看 

# 打卡学习，成为真正的性能测试高手



PC端用户扫码参与



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 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 21 | Tomcat：中间件监控及常用计数器解析

## 精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。