首页推荐

怎样保持MySQL Performance Schema的性能开销在可控范围内？--深度解析PFS对数据库性能的影响

原创 听见风的声音 2025-07-09 290

Performance Schema (PFS)是MySQL 5.5引进的新特性，在MySQL5.6对这个特性的结构进行了大幅度的调整，从MySQL5.7至8.0，这个特性一直在持续改善。这是 MySQL 内置的诊断引擎，通过轻量级代码插桩（Instrumentation）实时采集数据库内部操作数据，为性能优化提供原子级观测能力。与传统慢查询日志不同，PFS 直接挂钩数据库内核，可追踪锁竞争、文件 I/O、内存分配等底层行为。

对于Performance Schema的使用，很多人持谨慎甚至是怀疑的态度，这主要是出于对其造成性能方面的消耗的疑虑。本文从第三方的测试报告、PFS的原理方面展示PFS对性能的影响，以及如何通过各种手段降低PFS对性能的消耗，使其在容忍的范围之内，使我们在不影响数据库的性能的情况下能享受PFS给我们带来的性能分析和诊断方面的收益。

1 从第三方的测试报告来看

oracle官方、Percona、AWS 都发布关于PFS对数据库性能影响的白皮书、测试报告或者生产基准，下面这个报告是在AWS上的测试数据，测试环境如下：

资源项	配置
服务器	AWS **r6g.8xlarge** (32 vCPU, 256 GB)
MySQL 版本	**MySQL 8.0.34** (InnoDB引擎)
数据集	**TPS 基准：** 120,000 (Sysbench OLTP)
测试工具	Sysbench 1.0.20 + 自定义监控脚本
压力模型	混合读写 (70% SELECT, 30% UPDATE)

性能测试数据（256并发线程）见下面的表格：

PFS 配置模式	QPS	TPS	Avg Latency	CPU 使用率	内存增量
PFS 完全关闭 (Baseline)	142,800	28,560	8.9 ms	78%	0 MB
仅基础监控 (waits/io + statements)	**140,200 (-1.8%)**	28,040 (-1.8%)	9.2 ms (+3.4%)	81% (+3%)	210 MB
全事件监控 (所有 instruments)	**129,500 (-9.3%)**	25,900 (-9.3%)	10.7 ms (+20.2%)	89% (+14%)	480 MB
全事件+历史记录 (含 events_statements_history_long)	122,100 (-14.5%)	24,420 (-14.5%)	12.3 ms (+38.2%)	94% (+20%)	1.2 GB
内存监控 (启用 memory/)	105,600 (-26%)	21,120 (-26%)	16.1 ms (+81%)	97% (+24%)	1.8 GB

上面的测试数据在基准情况下CPU利用率达到了78%，这是一个比较高的数值，大部分生产数据库CPU利用率经常在60%左右，平均延迟也不低，可以说是一个接近性能瓶颈的情况。在这种情况下基础监控的性能消耗在3%左右是一个可以接受的结果，开启全监控的性能消耗还是比较大的，达到14%，这个大概是无法接受的了。至于内存监控后系统消耗达到24%，DeepSeek从而得出了内存监控是高位项，是绝对不能开的。这应该是个靠不住的结论，如调整一下顺序，在基础监控后测试内存监控，可能会是另一个结果了。

2 默认情况下的性能消耗（MySQL 8）

PFS在MySQL 8 默认是打开的，事件监控则不是全开，打开的事件监控数量如下：

```
select
        SUBSTRING_INDEX(name, '/', 1) ,
        count(*),
        sum(case ENABLED when 'YES' then 1 else 0 end) enabled_count
from
        performance_schema.setup_instruments
group by
        SUBSTRING_INDEX(name, '/', 1);
+----------------------------+--------+-------------+
| SUBSTRING_INDEX(name, '/', 1) | count(*) | enabled_count |
+----------------------------+--------+-------------+
| wait                       |    382 |          53 |
| idle                       |      1 |           1 |
| stage                      |    124 |          16 |
| statement                  |    212 |         212 |
| transaction                |      1 |           1 |
| memory                     |    498 |         498 |
| error                      |      1 |           1 |
+----------------------------+--------+-------------+
```

这个配置下，PFS造成的性能损耗大概在2.5%到5%左右，所以有些小伙伴安装MySQL 8采用的默认配置也不用紧张，这个性能损耗比起PFS的收益来看，完全是值得的，Oracle的awr有没有性能损耗？理论上和实际上都有，没必要仅仅对MySQL PFS的性能损耗锱铢必较。

听见风的声音
关注
83 文章 38 粉丝 80K+ 浏览量
获得了 179 次点赞
内容获得 48 次评论
获得了 297 次收藏

热门文章
理解model高级语句 2023-03-01 8437浏览
Oracle会话超时设置1：在sqlnet.ora和listener.ora中设置 2023-02-15 8305浏览
Postgresql 15的安装及简单使用 2023-03-06 4736浏览
oracle 数据库中的行锁和死锁 2023-01-12 4211浏览
Oracle --Oracle 11.2.0.4静默安装 2023-01-17 2176浏览

在线实训环境入口
MySQL在线实训环境 查看详情 »

最新文章
MySQL MCP Server--更轻松的连接大模型和MySQL数据库 5天前 135浏览
执行效率提高数十倍，这几个Oracle SQL的优化技巧你一定要掌握 6天前 240浏览
索引条件下推和分区--一条SQL语句执行计划的分析 2025-07-23 196浏览
null和子查询--not in和not exists怎么选择？ 2025-07-21 182浏览
巧用json工具解析MySQL优化器追踪文件 2025-07-16 99浏览

目录
- 1 从第三方的测试报告来看
- 2 默认情况下的性能消耗（MySQL 8）
- 3 PFS性能开销分析
  - 3.1 从原理分析
  - 3.2 记录事件时间的开销
- 4 使用事件过滤避免不必要的性能开销
  - 4.1 instrument过滤--只激活关注的ins...

3 PFS性能开销分析

3.1 从原理分析

PFS的原理是在 MySQL 服务器代码的关键执行路径上插入轻量级的“检测点”（Instruments），通过这些检测点收集执行过程中的各种性能数据，并将这些数据存储在内存表中供用户查询分析。PFS对数据库的影响主要体现在一下三个方面

- 指令级开销 (CPU Cycles)：在每个被插桩的代码路径上，P_S 添加了额外的函数调用（检测点代码）。这些代码要检查该检测点是否启用 (if (instrument_enabled) {...})，如果启用，执行记录操作：获取高精度时间戳 (clock_gettime())，更新计数器，将事件数据写入预分配的缓冲区。这些操作本身是轻量级的，但累积起来就是主要的 CPU 开销来源。
- 内存开销： PFS使用内存表存储数据。主要内存消耗在事件缓冲区内，也就是events_waits_current, events_*history, events*_history_long 等消费者表（消费者表实质上是预分配内存的环形缓冲区或链表。history_long 表通常占用最多内存）*summary* 等聚合汇总表（在内存中维护各种维度的聚合统计（哈希表结构），内部数据结构（维护检测点状态、线程状态、对象信息等）需要的少量内存。
- 并发与锁开销：线程本地存储 (TLS)（ P_S 大量使用 TLS 来存储每个线程的当前事件状态 (events_*_current)，这避免了线程间竞争，是高效的关键设计），缓冲区写入（当事件结束时，需要将记录从线程本地状态转移到历史缓冲区 或更新汇总表 。这些操作通常需互斥锁保护共享数据结构），汇总表更新（ 对 *summary* 表的更新也需要锁保护）。

以上三种开销，比较负载和可以量化的CPU开销，用一个具体的例子（MUTEX）详细说明一下,MUTEX插桩（instrument）由宏定义的，在文件mysql/psi/mysql_thread.h中（基于MySQL 8.0源码），代码如下：

```
#ifdef HAVE_PSI_MUTEX_INTERFACE
  #define mysql_mutex_lock(M, F, L) \
    inline_mysql_mutex_lock(M, F, L)
#else
  #define mysql_mutex_lock(M, F, L) \
    ( (M)->lock() )
#endif
```

inline_mysql_mutex_lock函数的实现如下

```
static inline void
inline_mysql_mutex_lock(mysql_mutex_t *mutex, const char *src_file, uint src_line)
{
  PSI_mutex_locker *locker = nullptr;
  PSI_mutex_locker_state state;
  if (PSI_MUTEX_CALL(start_mutex_wait)(&state, mutex->m_psi, PSI_MUTEX_LOCK, src_file, src_
    locker = &state;
  }
  mutex->lock(); // 实际调用互斥锁的lock方法
  if (locker != nullptr) {
    PSI_MUTEX_CALL(end_mutex_wait)(locker, 0);
  }
}
```

上面这段代码是插桩的核心：

- 首先，它声明了一个 `PSI_mutex_locker` 指针和一个状态变量 `state` 。
- 然后，它调用 `PSI_MUTEX_CALL(start_mutex_wait)` 开始记录等待事件。这个宏会展开为Performance Schema的调用，如果当前启用了对互斥锁的监控，则返回一个非空的 `locker` （指向状态变量的指针），否则返回0（此时 `locker` 保持为 `nullptr` ）。
- 接着，执行实际的锁获取操作： `mutex->lock()` 。
- 最后，如果 `locker` 非空（表示正在记录事件），则调用 `PSI_MUTEX_CALL(end_mutex_wait)` 结束事件记录。

如果没有插桩，每次锁操作只执行锁本身的指令（这本身也有开销，比如系统调用futex或原子操作）。如果有插桩，每次锁操作额外增加：

- 一个条件判断（检查是否启用插桩）。
- 两个函数调用（start和end）。
- 在start和end函数内部：两次时间戳获取（约20-100个周期，取决于系统），事件结构的初始化（几个内存写入操作）。
  保守估计假设每次插桩额外消耗100个CPU周期，那么1000次锁操作将增加100,000个周期。在2GHz的CPU上，一个周期是0.5纳秒，所以总增加时间为50微秒。对于一个原本需要1毫秒的查询，这增加了5%的开销（50微秒/1000微秒）。如果锁操作更多（比如10,000次），开销就会达到50%。

3.2 记录事件时间的开销

插桩的开销大部分是记录事件的时间的开销，选择不同的时间单位，开销也有很大差异，不同的版本也有所不同

```
mysql> select version();--数据库版本
+-----------+
| version() |
+-----------+
| 8.4.5     |
+-----------+
1 row in set (0.00 sec)
---支持的timer
mysql> SELECT * FROM performance_schema.performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      1796616915 |                1 |             13 |
| NANOSECOND  |      1000000000 |                1 |             27 |
| MICROSECOND |         1000000 |                1 |             25 |
| MILLISECOND |            1036 |                1 |             25 |
| THREAD_CPU  |       358391564 |                1 |           1510 |
+-------------+-----------------+------------------+----------------+
```

其中TIMER_OVERHEAD是该timer下获取一次时间最少的CPU周期（cycle），TIMER_FREQUENCY是每秒的时间单元，名为cycle的timer的时间单元和cpu的主频有关，一个事件需要在开始和结束时共获取两次时间，因此，对每个事件，这个开销要乘2。

4 使用事件过滤避免不必要的性能开销

PFS的事件过滤有两种：Pre-filering和Post-filering。Pre-filering通过调整PFS的配置只收集需要的事件，只更新需要的消费者表。Post-filering是指用where语句查询消费者表。Post-filering影响的是查询的内容，不影响PFS本身的性能开销。使用Pre-filering却对PFS的性能开销造成显著的影响。

4.1 instrument过滤–只激活关注的instrument

更新setup_instrument表可以激活或者关闭特定的instrument，更新立即生效，比如要关闭所有的wait类事件

```
mysql> update setup_instruments set enabled='NO' where name like 'wait%';
Query OK, 54 rows affected (0.00 sec)
Rows matched: 402  Changed: 54  Warnings: 0
```

4.2 按账号过滤

更新setup_actors表可以按照用户来选择instrument和consumer，这个表缺省情况下最大支持100行，如果超过这个值，需要设置performance_schema_setup_actors_size，更改这个参数的值需要重启数据库。

```
mysql> select * from setup_actors;
+------+------+------+---------+---------+
| HOST | USER | ROLE | ENABLED | HISTORY |
+------+------+------+---------+---------+
| %    | %    | %    | YES     | YES     |
+------+------+------+---------+---------+
1 row in set (0.00 sec)
```

默认情况下，PFS不对用户进行过滤，如果只需要对某些用户的进行监控，可以删除默认的条目，在这个表内插入相关条目。此表更新后会在用户下一次登录后生效。

4.3 按照对象过滤

setup_objects表控制是否对特定数据库对象启用监控，对象包括事件、函数、过程、表和触发器。

```
select * from setup_objects;
+-----------+-----------------+-----------+-------+-----+
|OBJECT_TYPE|OBJECT_SCHEMA    |OBJECT_NAME|ENABLED|TIMED|
+-----------+-----------------+-----------+-------+-----+
|EVENT      |mysql            |%          |NO     |NO   |
|EVENT      |performance_schema|%          |NO     |NO   |
|EVENT      |information_schema|%          |NO     |NO   |
|EVENT      |%                |%          |YES    |YES  |
|FUNCTION   |mysql            |%          |NO     |NO   |
|FUNCTION   |performance_schema|%          |NO     |NO   |
|FUNCTION   |information_schema|%          |NO     |NO   |
|FUNCTION   |%                |%          |YES    |YES  |
|PROCEDURE  |mysql            |%          |NO     |NO   |
|PROCEDURE  |performance_schema|%          |NO     |NO   |
|PROCEDURE  |information_schema|%          |NO     |NO   |
|PROCEDURE  |%                |%          |YES    |YES  |
|TABLE      |mysql            |%          |NO     |NO   |
|TABLE      |performance_schema|%          |NO     |NO   |
|TABLE      |information_schema|%          |NO     |NO   |
|TABLE      |%                |%          |YES    |YES  |
|TRIGGER    |mysql            |%          |NO     |NO   |
|TRIGGER    |performance_schema|%          |NO     |NO   |
|TRIGGER    |information_schema|%          |NO     |NO   |
|TRIGGER    |%                |%          |YES    |YES  |
+-----------+-----------------+-----------+-------+-----+
```

从这个表的默认内容来看，对MySQL系统自带的数据库，监控是关闭的，对其他数据库则是打开的，更新这个表的内容可以只对特定的数据库打开监控，删除user和host值为%的条目，加入需要监控的数据库（模式）的相关条目。

4.4 按线程过滤

PFS也支持按线程来启动监控，通过setup_threads表控制

```
select * from setup_threads ;
-------------------------------------------------+-------+-------+----------+----------+-
NAME                                             |ENABLED|HISTORY|PROPERTIES|VOLATILITY|D
-------------------------------------------------+-------+-------+----------+----------+-
thread/performance_schema/setup                  |YES    |YES    |singleton |         0|
thread/sql/con_named_pipes                       |YES    |YES
------省略多行
thread/sql/con_sockets                           |YES    |YES    |singleton |         0|
  |singleton |         0|             |
thread/sql/replica_monitor                       |YES    |YES    |singleton |         0|
-------------------------------------------------+-------+-------+----------+----------+-
```

如不想监控某些线程，更新此表相关条目即可。

5 PFS过滤示例—只监控表的DDL操作

5.1 简介

有些数据库，需要关注的是对表的ddl操作，比如表drop了，或者是列的定义变了。这种需求，在Oracle和PG可以用系统触发器来实现，对于MySQL这种没有触发器的，使用PFS是比较好的实现方式，下面就是实现表的DDL跟踪的步骤。

5.2 调整setup_instruments，只激活表的ddl相关事件的插桩

有些PFS的插桩之间有依赖关系，statement/sql/类事件依赖statement/abstract事件就是一个例子，如果不激活相关statement/abstract事件，只激活statement/sql/事件，不会收集相关sql的信息。

1）关闭所有以statement开头的插桩

```
mysql> update  performance_schema.setup_instruments set enabled='NO' where name like 'state
Query OK, 51 rows affected (0.01 sec)
Rows matched: 214  Changed: 51  Warnings: 0
```

2）激活statement/sql依赖的事件的插桩

```
mysql> update  performance_schema.setup_instruments set enabled='yes' where name like 'stat
Query OK, 4 rows affected (0.00 sec)
Rows matched: 4  Changed: 4  Warnings: 0
```

3）激活和表的ddl语句有关的事件的插桩

```
mysql>  update  performance_schema.setup_instruments set enabled='yes' where name like 'stc
Query OK, 5 rows affected (0.00 sec)
Rows matched: 5  Changed: 5  Warnings
```

4）检查statement类事件插桩设置

```
mysql> select name,enabled from setup_instruments where name like 'statement/%' and enablec
+--------------------------------+---------+
| name                           | enabled |
+--------------------------------+---------+
| statement/sql/create_table     | YES     |
| statement/sql/alter_table      | YES     |
| statement/sql/drop_table       | YES     |
| statement/sql/show_create_table | YES     |
| statement/sql/rename_table     | YES     |
| statement/abstract/clone       | YES     |
| statement/abstract/Query       | YES     |
| statement/abstract/new_packet  | YES     |
| statement/abstract/relay_log   | YES     |
+--------------------------------+---------+
9 rows in set (0.00 sec)
```

5.3 效果验证

1）运行一些语句

```
mysql> use test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| test           |
| test_bak       |
+----------------+
2 rows in set (0.00 sec)
mysql> select count(*) from test;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)
mysql> insert into test_bak values (5,'jack');
Query OK, 1 row affected (0.02 sec)
mysql> drop table test_bak;
Query OK, 0 rows affected (0.07 sec)
```

2）查询PFS记录的语句

```
mysql> use performance_schema;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> select TIMER_START,sql_text,current_schema from performance_schema.events_statements
+------------------+--------------------+----------------+
| TIMER_START      | sql_text           | current_schema |
+------------------+--------------------+----------------+
| 7343067004258000 | drop table test_bak | test           |
+------------------+--------------------+----------------+
1 row in set (0.00 sec)
```

可以看到，在执行的SQL语句中，只有drop table被记录了下来，这个示例扩展一下，可以限制记录的schema，达到都某个业务的DDL进行监控的目的。

6 小结

MySQL PFS的性能开销完全在可控范围内，8.0 以上版本采用默认配置其性能开销也在可以容忍的范围之内。要注意的是，PFS的开销与数据库的负载基本成正比。在要求严格生产环境内可以按照需求定义监控项，因为大部分监控项的启停都是立即生效的，完全可以只启用基本的监控项（如某个数据库的sql或者事务监控），在需要时在启用其他监控项。

墨力计划 mysql 性能优化

「喜欢这篇文章，您的关注和赞赏是给作者最好的鼓励」

关注作者 赞赏

评论

分享你的看法，一起交流吧～

相关阅读

ACDU周度精选 | 本周数据库圈热点 + 技术干货分享（2025/7/25期）
墨天轮小助手 469次阅读 2025-07-25 15:54:18

ACDU周度精选 | 本周数据库圈热点 + 技术干货分享（2025/7/17期）
墨天轮小助手 436次阅读 2025-07-17 15:31:18

墨天轮「实操看我的」数据库主题征文活动启动
墨天轮编辑部 379次阅读 2025-07-22 16:11:27

【GaussDB】深入剖析Insert Select慢的定位全过程
DarkAthena 305次阅读 2025-07-27 01:28:24

深度解析MySQL的半连接转换
听见风的声音 204次阅读 2025-07-14 10:23:00

MySQL 9.4.0 正式发布，支持 RHEL 10 和 Oracle Linux 10
严少安 199次阅读 2025-07-23 01:21:32

索引条件下推和分区-一条SQL语句执行计划的分析
听见风的声音 196次阅读 2025-07-23 09:22:58

null和子查询--not in和not exists怎么选择？
听见风的声音 182次阅读 2025-07-21 08:54:19

MySQL数据库SQL优化案例(走错索引)
陈举超 164次阅读 2025-07-17 21:24:40

使用 MySQL Clone 插件为MGR集群添加节点
黄山谷 162次阅读 2025-07-23 22:04:19