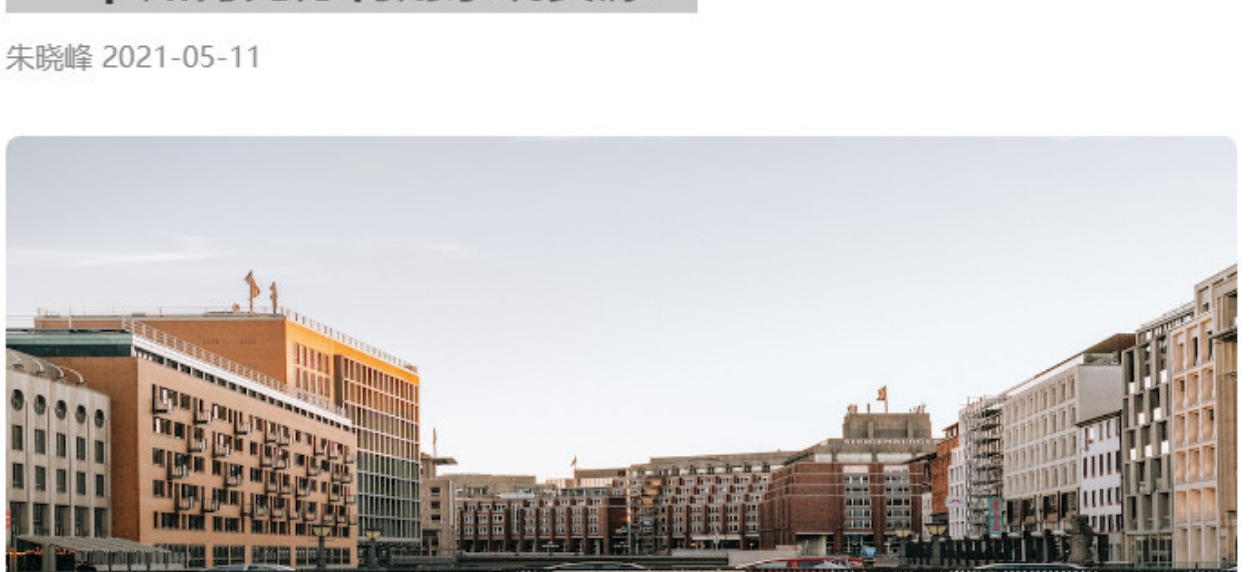
26 | 如何充分利用系统资源？

朱晓峰 2021-05-11



你好，我是朱晓峰。

内存和 CPU 都是有限的资源，因此，把它们的作用发挥到极致，对提高应用的承载能力来说至关重要。

磁盘读写需要计算位置、发出读写指令等，这些都要消耗 CPU 资源，很容易成为提升系统效能的瓶颈。

如果采取“先把数据放在内存，然后集中写入磁盘”的办法，可以节省 CPU 资源和磁盘读取的时间，但是也会面临系统故障时会丢失数据的风险；相反，如果每次都写入磁盘，数据最安全，但是频繁的磁盘读写，会导致系统效率低下。这就需要我们提升优化资源配置的能力。

今天，我就给你介绍一下优化系统配置的方法，同时还会讲解系统自带的监控工具，从而帮助你合理配置系统资源，精准发现系统资源的瓶颈，进一步提升你处理大并发、大数据的能力。

优化系统资源配置

对 CPU 资源的掌控，关系到系统整体的成败。因为 CPU 资源是系统最核心的资源，无可替代，而且获取成本高。如果应用无法控制 CPU 的使用率，就有可能是失败的，不管你的界面多么人性化，功能多么强大。

因此，我们需要管理好系统配置，把资源效率提升到极致。**系统参数控制着资源的配置，调整系统参数的值，可以帮助我们提升资源的利用效率。**

我来借助一个小例子，给你介绍下怎么通过对系统变量进行调优，来提升系统的整体效率。

我曾参与过一个点餐系统应用的开发，其实就是一个为客户提供点餐服务的应用，类似于美团。商家购买服务，入住平台，开通之后，商家可以在系统中录入自己能够提供的各类餐食品种，客户通过手机 App、微信小程序等点餐，商家接到订单以后进行制作，并根据客户需求提供堂食或者送餐服务。

系统中包括订单表（orderlist）、客户信息表（clientlist）和客户储值表（clientdeposit）。

订单表：

orderid（订单编号）	clientid（客户编号）	Itemnumber（商品编号）	quantity（数量）	price（价格）	remarks（备注）	transdate（交易时间）
1	1	1	10	35	少油	2020-10-18 12:00:00

客户信息表：

clientid（客户编号）	clientname（客户名称）	phone（电话）	address（地址）
1	张三	18512345678	北京市海淀区

客户储值表：

clientid（客户编号）	depositvalue（储值金额）	balance（余额）	recordingdate（储值日期）
1	100	50	2020-10-18

刚刚上线的时候，系统运行状态良好。但是，随着入住的商家不断增多，使用系统的用户量越来越多，每天的订单数据达到了 2 万条以上。这个时候，系统开始出现问题，CPU 使用率不断飙升。终于，有一天午餐高峰的时候，CPU 使用率达到 99%，这实际上就意味着，系统的计算资源已经耗尽，再也无法处理任何新的订单了。换句话说，系统已经崩溃了。

这个时候，我们想到了对系统参数进行调整，因为参数的值决定了资源配置的方式和投放的程度。为了解决这个问题，我们一共调整了 3 个系统参数，分别是 **InnoDB_flush_log_at_trx_commit、InnoDB_buffer_pool_size、InnoDB_buffer_pool_instances。**

接下来，我就给你讲一讲怎么对这三个参数进行调整。

1. 调整系统参数 InnoDB_flush_log_at_trx_commit

这个参数适用于 InnoDB 存储引擎。因为刚刚的表用的存储引擎都是 InnoDB，因此，这个参数对我们系统的效能就有影响。需要注意的是，如果你用的存储引擎不是 InnoDB，调整这个参数对系统性能的提升就没有什么用了。

这个参数存储在 MySQL 的配置文件 my.ini 里面，默认的值是 1，意思是每次提交事务的时候，都把数据写入日志，并把日志写入磁盘。**这样做的好处是数据安全性最佳，不足之处在于每次提交事务，都要进行磁盘写入的操作。**在大并发的场景下，过于频繁的磁盘读写会导致 CPU 资源浪费，系统效率变低。

这个参数的值还有 2 个可能的选项，分别是 0 和 2。其中，0 表示每隔 1 秒将数据写入日志，并将日志写入磁盘；2 表示，每次提交事务的时候都将数据写入日志，但是日志每间隔 1 秒写入磁盘。

最后，我们把这个参数的值改成了 2。这样一来，就不用每次提交事务的时候都启动磁盘读写了，在大并发的场景下，可以改善系统效率，降低 CPU 使用率。即便出现故障，损失的数据也比较小。0 虽然效率更高一些，但是数据安全性方面不如 2。

2. 调整系统参数 InnoDB_buffer_pool_size

这个参数的意思是，InnoDB 存储引擎使用缓存来存储索引和数据。这个值越大，可以加载到缓存区的索引和数据量就越多，需要的磁盘读写就越少。

因为我们的 MySQL 服务器是数据库专属服务器，只用来运行 MySQL 数据库服务，没有其他应用了，而我们的计算机是 64 位机，内存也有 128G。于是我们把这个参数的值调整为 64G。这样一来，磁盘读写次数可以大幅降低，我们就可以充分利用内存，释放出一些 CPU 的资源。

3. 调整系统参数 InnoDB_buffer_pool_instances

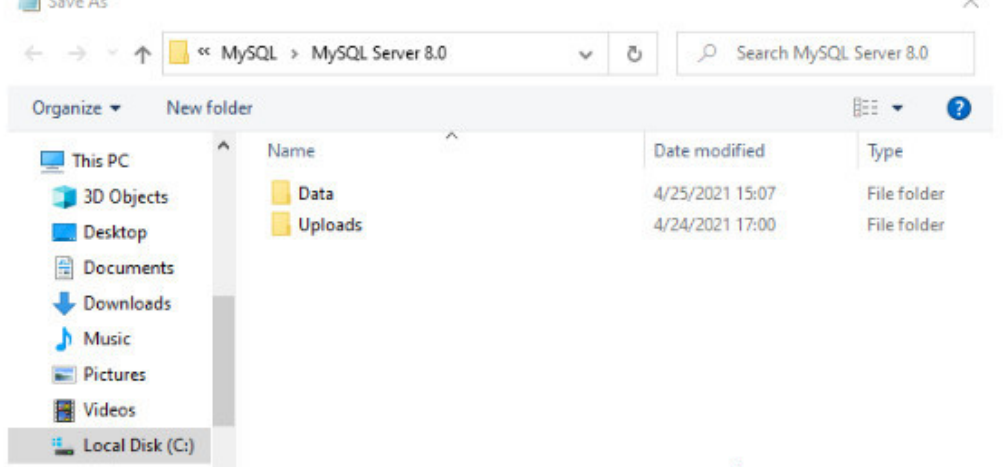
这个参数的意思是，将 InnoDB 的缓存区分成几个部分，这样一来，就可以提高系统的并行处理能力，因为可以允许多个进程同时处理不同部分的缓存区。这就好比买电影票，如果大家都挤在一个窗口、一个接一个地进行交易，效率肯定是很慢的。如果一次开很多售票窗口，多笔交易同时进行，那速度就快得多了。

我们把 InnoDB_buffer_pool_instances 的值修改为 64，意思就是把 InnoDB 的缓存区分成 64 个分区，这样就可以同时有多个进程进行数据操作，CPU 的效率就高多了。

修改好了系统参数的值，我们需要重新保存 MySQL 的配置文件 my.ini，并且重启 MySQL 数据库服务器。

这里有个坑你要注意：由于 my.ini 文件是文本格式文件，你完全可以用记事本对文件进行修改操作。但是，如果你只是简单地进行保存，就会发现，MySQL 服务器停止之后，再次启动时没有响应，服务器起不来了。其实，这就是文件的码制出了问题。

记事本保存文件默认的码制是 UTF-8，但配置文件的码制必须是 ANSI 才行。所以，当**你修改完 MySQL 的配置文件 my.ini 之后，保存的时候，记得用 ANSI 的格式。**如下图所示：



经过我们对系统参数的调整，重启 MySQL 服务器之后，系统效率提高了，CPU 资源的使用率下来了，系统得以正常运行。

咱们来小结下。CPU 资源是系统的核心资源，获取成本非常高。CPU 的特点就是阻塞，只要 CPU 一开始计算，就意味着等待。遇到 CPU 资源不足的问题，可以从 2 个思路去解决：

1. 疏通拥堵路段，消除瓶颈，让等待的时间更短；
2. 开拓新的通道，增加并行处理能力。

刚刚的调优思路，其实就是围绕着这 2 个点展开的。如果遇到 CPU 资源不足的问题，我建议你也从这 2 个角度出发去思考解决办法。

如何利用系统资源来诊断问题？

在刚刚的例子中，我提到了解决 CPU 资源不足需要消除瓶颈。而消除瓶颈的第一步就是要发现它。如何发现呢？幸运的是，MySQL 提供了很好的工具：**Performance Schema**。

这是一种专门用来监控服务器执行情况的存储引擎，它会把监控服务器执行情况的数据记录在系统自带的数据库 performance_schema 中。我们可以利用监控的数据，对服务器中执行查询的问题进行诊断。

我还是以刚刚的那个点餐系统为例，来解释一下。

当我们调整完系统参数之后，系统恢复了运行。可是随着数据量的不断增大，单日订单量超过 20 万，我们再次遇到了问题：CPU 飙升到 99%，系统无法工作了。这个时候，我们就可以利用 performance_schema 记录的监控数据来发现问题。

我先讲一讲怎么让 Performance Schema 监控查询执行事件，并且把我们需要的监控数据记录下来。

如何启用系统监控？

系统数据库 performance_schema 中的表 setup_instruments 和 setup_consumers 中的数据，是启用监控的关键。

setup_instruments 保存的数据，表示哪些对象发生的事件可以被系统捕获（在 MySQL 中，把这些事件称作信息生产者）。

我们可以通过下面的代码，来查看一下当前 MySQL 会监控哪些事件的信息：

```
mysql> SELECT NAME,ENABLED,TIMED
-> FROM performance_schema.setup_instruments
-> LIMIT 1,10;
+---------------------------------------------------+---------+-------+
| NAME | ENABLED | TIMED |
+---------------------------------------------------+---------+-------+
| wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_tc | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit_queue | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_done | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_flush_queue | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_index | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_binlog_end_pos | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_sync | YES | YES |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_sync_queue | YES | YES |
+---------------------------------------------------+---------+-------+
10 rows in set (0.00 sec)
```

表的内容有很多，为了方便演示，这里我只查询了前 10 条数据。在这个表中，你要注意 3 个字段。

- NAME：表示事件的名称；
- ENABLED：表示是否启用了对这个事件的监控；
- TIMED：表示是否收集事件的时间信息。

这里我们启用所有事件的监控，以便诊断问题：

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES';
```

setup_instruments 表中的数据指定了是否对某一事件进行监控，而 setup_consumers 这个表保存的数据，则指定了是否保存监控事件发生的信息。在 MySQL 中，Performance Schema 把监控到的事件信息存入 performance_schema 中的数据表中，这些数据表保存了事件的监控信息，扮演了事件监控信息消费者的角色，被称为消费者。

下面我们来修改设置，使系统保存所有事件信息，并了解一下 setup_consumers 这个表里都能够控制保存哪些事件的信息。我在数据旁边加了注释，来说明保存的事件信息的种类。

```
UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'

mysql> SELECT *
-> FROM performance_schema.setup_consumers
-> ;
+----------------------------------+---------+
| NAME | ENABLED |
+----------------------------------+---------+
| events_stages_current | YES |            -- 当前阶段
| events_stages_history | YES |            -- 阶段历史
| events_stages_history_long | YES |       -- 阶段长历史
| events_statements_current | YES |        -- 当前语句
| events_statements_history | YES |        -- 历史语句
| events_statements_history_long | YES |   -- 长历史语句
| events_transactions_current | YES |      -- 当前事务
| events_transactions_history | YES |      -- 历史事务
| events_transactions_history_long | YES | -- 长历史事务
| events_waits_current | YES |             -- 当前等待
| events_waits_history | YES |             -- 等待历史
| events_waits_history_long | YES |        -- 等待长历史
| global_instrumentation | YES |
| thread_instrumentation | YES |
| statements_digest | YES |
+----------------------------------+---------+
15 rows in set (0.01 sec)
```

更新之后，所有种类事件信息的保存都已经启用。这样一来，所有查询事件都能够被监控，事件的相关信息也都能够被记录下来了。如果查询中出现问题，那么，异常信息就会被记录下来。接下来，我们就利用系统监控到的信息来诊断一下问题。

利用监控信息诊断问题

为了利用保存下来的监控事件信息来诊断系统问题，我先介绍一下几个保存监控信息数据的系统数据表。

第 1 个表是 performance_schema.events_statements_current。

这个表中记录的是当前系统中的查询事件。表中的每一行对应一个进程，一个进程只有一行数据，显示的是每个进程中被监控到的查询事件。

第 2 个表是 performance_schema.events_statements_history。

这个表中记录了系统中所有进程中最近发生的查询事件。这个表中包含的查询事件都是已经完成了的。另外，表中可以为每个进程保存的最大记录数由系统变量决定。

下面的代码可以查询当前系统中可以为每个进程保存的最大记录数的值：

```
mysql> show variables like '%performance_schema_events_statements_history_size
-> ;
+---------------------------------------------------+-------+
| Variable_name | Value |
+---------------------------------------------------+-------+
| performance_schema_events_statements_history_size | 10 |  -- 最多10条
+---------------------------------------------------+-------+
1 row in set, 1 warning (0.01 sec)
```

结果显示，当前的设定是：这个表中为每个进程保留最多 10 条记录。

我要说的第 3 个表，是 performance_schema.events_statements_history_long。

这个表中记录了系统中所有进程中最近发生的查询事件，表中包含的查询事件都是已经完成了的。同时，这个表中可以保存的记录数由系统变量决定。下面的代码用来查询当前系统设置的这个表可以保存的最大记录数。

```
mysql> show variables like '%performance_schema_events_statements_history_long
-> ;
+--------------------------------------------------------+-------+
| Variable_name | Value |
+--------------------------------------------------------+-------+
| performance_schema_events_statements_history_long_size | 10000 |
+--------------------------------------------------------+-------+
1 row in set, 1 warning (0.00 sec)
```

结果显示，当前系统的设定是，这个表中最多保留 10000 条记录。

知道了这几个保存事件信息的系统数据表，我们重新回到项目中来，看看如何利用这几个系统数据表中的信息，来发现问题。

我们先用下面的语句，来看一下哪些查询消耗的时间多。这个表的数据量很大，为了方便你查看，我只让它显示最初的 2 行数据。

```
mysql> SELECT
-> TRUNCATE(TIMER_WAIT / 1000000000000, 6) AS duration, -- 计算查询的时长，单位是
-> sql_text,
-> EVENT_ID
-> FROM
-> performance_schema.events_statements_history_long
-> WHERE
-> TRUNCATE(TIMER_WAIT / 1000000000000, 6) <> 0
-> AND sql_text IS NOT NULL
-> ORDER BY TRUNCATE(TIMER_WAIT / 1000000000000, 6) DESC
-> LIMIT 1,2;
+----------+-----------------------------------+----------+
| duration | sql_text | EVENT_ID |
+----------+-----------------------------------+----------+
| 137.2529 | select count(*) from demo.trans | 17 |
| 137.2420 | select count(*) from demo.trans | 907 |
+----------+-----------------------------------+----------+
2 rows in set (0.00 sec)
```

结果显示，持续时间最长的 2 个事件分别是事件编号为 17 和 907 的事件。这样一来，我们就可以发现到底是哪些查询消耗了最多的 CPU 资源，就可以有针对性地进行优化了。

下面我具体给你解释一下这个查询。

1. 字段 TIMER_WAIT：表示这个查询消耗了多少时间，单位是微微秒，也就是万亿分之一秒。
2. TRUNCATE(X,D) 函数：表示给 X 保留 D 位小数，注意这个函数是直接截取，没有四舍五入。
3. 字段 sql_text：表示执行的 SQL 语句的内容。
4. EVENT_ID：表示事件编号。

这个查询的意思是，按照查询花费时间多少排序，查询出花费时间最多的 2 个查询的事件编号、执行时长和查询的内容。

通过查询这个表的内容，我们发现，有大量的查询出现在执行时间较长的列表中，这些查询都涉及数据表 demo.clientdeposit。

为了支持顾客储值消费，我们新加了这个表，但是这个表的使用频率很高，数据量较大，我们却忘记了创建索引，这就导致了 CPU 资源不足。确定了问题，解决起来就很容易了，创建了索引之后，CPU 资源消耗直接下降到了 20% 左右。

总之，Performance Schema 是一种很好的工具，可以监控到 MySQL 服务器内部执行的信息。如果你遇到查询中出现难以解决的问题，就可以调取数据库 performance_shema 中的监控数据，分析服务器中的执行情况，从而定位和解决问题。

总结

这节课，我们学习了通过系统参数来配置资源、提高查询效率的方法。

- 系统参数 InnoDB_flush_log_at_trx_commit 适用于 InnoDB 存储引擎。默认的值是 1，意思是每次提交事务的时候，都把数据写入日志，并把日志写入磁盘。0 表示每隔 1 秒将数据写入日志，并将日志写入磁盘。2 表示每次事务提交的时候，将数据写入日志，但是日志每隔 1 秒写入磁盘。
- 系统参数 InnoDB_buffer_pool_size 表示 InnoDB 存储引擎使用多少缓存来存储索引和数据。这个值越大，可以加载到缓存区的索引和数据量就越多，需要的磁盘读写就越少。
- 系统参数 InnoDB_buffer_pool_instances 的意思是，将 InnoDB 的缓存区分成几个部分，可以提高系统的并行处理能力。

而且，我还给你介绍了怎么通过使用 Performance Schema 来监控服务器运行中的事件。系统数据库 performance_schema 中的表 setup_instruments 和 setup_consumers 中的数据，是启用监控的关键。

- setup_instruments 保存的数据，表示哪些对象发生的事件可以被系统捕获。
- setup_consumers 保存的数据用来控制保存哪些事件的信息。

我们可以通过修改这两个表的内容来开启系统监控，并且把监控数据保存到相应的数据表中，方便我们对查询的执行情况进行诊断。

最后，我还想额外给你一个建议：如果你遇到了因为无法控制 CPU 的使用率而导致系统崩溃的情况，首先应该想到的**是应用本身有缺陷**，然后找到自身的问题，并加以解决，而不是增加系统资源的投入，采购功能强大的 CPU 和扩大内存等。

原因有 2 个：资源永远是有限的，如何在有限的资源前提下提高系统的承载能力，是我们应该首先考虑的；资源的投入都是经过谨慎评估的，除非你有充足的理由确信，同等条件下，你开发的应用承载能力已经超过了业界最高水平，否则就说明应用本身还有提升的空间。

所以，课下你一定要把今天的内容多学几遍，一定要掌握通过 MySQL 自身的事件监控机制来诊断问题的方法，它可以帮助你找到真正的瓶颈。

思考题

前面提到，我把 InnoDB_flush_log_at_trx_commit 的值改成了 2，因为 0 虽然效率更高一些，但是在数据安全性方面不如 2。你知道为什么 0 的效率更高一些，但是数据安全性却不如 2 吗?

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你分享给你的朋友或同事，我们下节课见。