# 发展过程

线程池是非常有实际价值的数据库组件，可以充分解决数据库并发和连接数越来越高的情况，让数据库在高压力下具备一定的自保能力，防止雪崩情况的发生。这里讲一下为什么说 sharedserver 是第 4 代线程池技术，对线程池技术的思考和研发是从 2019 年开始的，如下所述：

- 第一代线程池，指 Percona 分支中的开源版本，其单一队列的设计使之仅适合于单一负载场景，能不能适用需要看场景。当时在阿里内部是开启线程池以支持最高十万多的连接的，但公共云上无法默认开启。
- 第二代线程池，指为阿里云 RDS 实现的版本，采用多队列的设计，并优化不同队列的线程池退化机制（偏向急速退化），使之能适用于公共云上的任何外部业务场景，不会出现线程饿死的情况。
- 第三代线程池，指为目前任职公司业务场景重新设计和编码的版本，优化了队列设计以及不同队列的线程池退化机制，采用了渐进退化而非急速退化的策略，使之具备更好的高并发自保能力。
- 第四代线程池，指实验室中的完美版本，集成了一个极其紧凑有效的 SQL Digest 生成模块，在运行过程中收集了 SQL 的执行成本信息（执行时长、CPU 资源消耗、等锁时长）。采用了基于 SQL 执行成本的算法进行线程池调度，增加了慢 SQL、慢 DML、大事务队列，能够更好地平衡线程池的总线程数，并集成了较强的可观测能力。

目前还没有规划出第五代线程池的改进点，正在考虑集成一套自适应限流机制，增强数据库高并发下的自治能力。

## 综合优势

现在的业务都实现了服务化，并且应用采用虚拟化实例部署，需要连接同一个数据库的应用实例数量会非常大，因此一个数据库具有数千连接数或数万连接数都十分正常，一些头部的互联网公司的单个数据库实例，可能会有超 10 万个连接，会给稳定性带来极大的挑战。线程池有以下技术优势：

- 动态开启

  可以对新连接动态开启或关闭线程池，不需要重起 MySQL 实例。社区版本需要重起 MySQL 实例才能让线程池生效，不利于线上灰度。

- 节约 CPU

  数据库中会用很多的锁来保护一些关键的全局数据，比如锁列表、事务列表等，不同的会话会用忙等方式消耗 CPU 去抢占这些锁，在高并发的情况下，会出现著名的“ut_delay”函数消耗大量 CPU 的现象，线程池可以很好地消除这部份的无用消耗，MySQL 57 在数百并发更新下竞争就会比较严重，MySQL 8.0 虽然情况有所好转，在上千并发更新下竞争依然很严重。线程池是一种对业务透明的排队技术，可以让数据库一直处于高效运行的状态。

- 节约内存

  当不使用线程池时，每个连接都会创建一个 OS 线程，这是一笔不小的开销。每个线程还会有一定的私有内存空间（大约 150KB），当有上万个连接时，这部份的内存节约也可以达到 1GB，很可能让 MySQL 处于 Swap 或 OOM 的边缘。此外处理 SQL 的过程中，还可能申请线程级别的缓存，这是第三部份可以优化的内存消耗。

- 场景通用

通过创新的多队列技术（类似于高速公路上的分道行驶），可以让线程池胜任混合场景负载，不用担心相互干扰引起的卡顿现象（已在多个公司验证）。而社区的线程池则只能在同等负载（每个请求执行代码相差不大）场景上开启，否则便会相互影响，让短平快的查询（比如主键查询）变慢许多，绝大多数应用都无法接受这种情况。

- 更短 RT 幅度

  在较大规格的机器上用 1024 个并发测试 sysbench 的 Read Write 用例，可以发现不开启线程池时，RT 的 95 线在 100ms 左右，超出平均线 38ms 许多。而开启线程池的情况下，RT 的 95 线在 36ms 左右，平均线为 29ms，仅比平均线多一点点。除高并发下的性能提升外，较小的 RT 抖动幅度非常有利于业务应用的稳定运行。

- 提高连接数支持

  操作系统在 1-2 万线程时调度上已经很吃力了，在使用线程池后，通过 Listener / Worker / Session 机制，如同 Nginx 那样支持极高的数十万并发连接，让应用的连接数配置不再是难题，不再担心连数据库连接数高导致的弹性失败，可以省去很多事情。

- 提高短连接能力

  操作系统里创建线程的代价极高，大约一秒钟能够创建 1000 个已经很吃力了，这里就会限制你大批量应用发布或重起的速度，而采用线程池后则可以每秒钟支持 6000 个连接创建，并且在创建新连接过程中不会引起主机层面的性能抖动。

几家云大厂的 RDS 都已经开启线程池功能。

# 插件安装

线程池基于插件机制，可以在现有的 MySQL 片本上加载，比如在社区上下载的 MySQL RPM 包，可以成功加载，不需要采用外部编译的二进制包。如下图所示：

```
[root@RH6SRV1 ~]# /usr/local/onesql8031/bin/mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 8.0.31 OneSQL distribution

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show variables like 'sharedserver%';
+-----------------------------+---------+
| Variable_name               | Value   |
+-----------------------------+---------+
| sharedserver_admin_users    |         |
| sharedserver_big_dml        | 100     |
| sharedserver_big_query      | 50      |
| sharedserver_cpu_groups     |         |
| sharedserver_cpu_sample     | 1000    |
| sharedserver_enabled        | ON      |
```

插件需要根据目标 MySQL 的版本信息（MySQL 社区版本号、OS 版本号），如果是自己编译的则还需要提供 GCC 编译器版本信息，才能够编译出能够被加载的二进制插件。在得到二提制插件后，可以查询目标 MySQL 的插件安装位置，如下所示：

```
mysql> show variables like 'plugin_dir';
+---------------+-----------------------------------+
| Variable_name | Value                             |
+---------------+-----------------------------------+
| plugin_dir    | /usr/local/onesql8031/lib/plugin/ |
+---------------+-----------------------------------+
1 row in set (0.00 sec)
```

然后将二进制插件文件拷到指定目录，再在 my.cnf 中进行加载，比如文件名为“sharedserver_m8031.so”，加入配置“plugin-load = sharedserver_m8031.so”，然后重新起动 MySQL 实例即可。安装成功后，用”show plugins”命令应当可以看到如下信息：

```
| SHAREDSERVER                | ACTIVE   | DAEMON              | sharedserver_m8031.so |
| SHAREDSERVER_INFO           | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_RUNNING        | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SQLTEXT        | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SQLSTATS_ALL   | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SQLSTATS_60    | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SQLSTATS_300   | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SQLSTATS_900   | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_PERFSTAT       | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_PERFSTAT_HIS   | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SESSIONS_CUR   | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SESSIONS_ALL   | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SESSIONS_60    | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SESSIONS_300   | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
| SHAREDSERVER_SESSIONS_900   | ACTIVE   | INFORMATION SCHEMA  | sharedserver_m8031.so |
+-----------------------------+----------+---------------------+-----------------------+
60 rows in set (0.00 sec)
```

安装成功后，可以顺便了解一下线程池提供的可观测性能力增强。

## 参数配置

带插件起动后，可以查看线程池相关的新增参数（以"sharedserver"开头），绝大多数参数不需要显式配置。如下图所示：

```
mysql> show variables like 'sharedserver%';
+-----------------------------+---------+
| Variable_name               | Value   |
+-----------------------------+---------+
| sharedserver_admin_users    |         |
| sharedserver_big_dml        | 100     |
| sharedserver_big_query      | 50      |
| sharedserver_cpu_groups     |         |
| sharedserver_cpu_sample     | 1000    |
| sharedserver_enabled        | ON      |
| sharedserver_idle_timeout   | 10      |
| sharedserver_max_threads    | 10000   |
| sharedserver_normal_weights | 25      |
| sharedserver_oversubscribe  | 32      |
| sharedserver_route_mode     | SESSION |
| sharedserver_size           | 1       |
| sharedserver_stall_limit    | 100     |
| sharedserver_trans_weights  | 25      |
| sharedserver_wait_bigscale  | 100     |
| sharedserver_wait_delay     | 10      |
| sharedserver_wait_scale     | 100     |
+-----------------------------+---------+
17 rows in set (0.00 sec)
```

下面来介绍一下可能会需要调整的几个参数：

- Sharedserver_admin_users

  有些情况下，可能对 RT 要求非极期敏感，不希望请求被排队，比如 HA 的探针，可能稍微慢一点就会被认为服务不可用，这时可以将 HA 探针所用的

用户设置为管理类用户，这时每个连接还会使用独立 Worker 线程的模式，确保下发的请求会被立马执行，可以用逗号分隔指定多个用户名。

- Sharedserver_big_dml

  线程池会记录每个 SQL 的性能信息，可以自动根据后端执行时长进行不同队列分发操作。这个参数可指定慢 DML 的时长标准，默认值为 50ms，指后端执行时间超过 50ms，就会进入慢 DML 队列，以免阻挡小 DML 执行。

- Sharedserver_big_lockwait

  数据库中行锁等待是比较耗费资源的操作，如果大量的会话都处于行锁等待状态，会导致严重的数据库性能问题，比如热点商品抢拍中的并发扣减同一个库存的行为，本质上是数据库中更新同一行。线程池可以根据行锁等待时长决定是否将更新操作归入到慢 DML 队列。

- Sharedserver_big_query

  用于指定大查询的时长标准，默认值为 20ms，即执行时间超过 20ms 的 SQL 就会进入大查询队列，以免阻挡简单查询的执行。

- sharedserver_enabled

  设置是否启用线程池连接模式，仅对新建连接生效，原有连接需要断开重连才能改变连接模式。

- sharedserver_oversubscribe

  单个线程池 Group 的最佳 Worker 数量，默认值为 32，比较适合 MySQL 57 的默认值，如果是 MySQL 8.0 则可以配置为 64。一般情况下不需要调整，调整幅度不大的话，调整的影响也非常可控。

- sharedserver_route_mode

线程池 Group 可能有多个，可以根据一定的规则将会话分派到不同的 Group，由于单个 Group 一般不会用完所有资源，因此也可以起到一定的资源限制与隔离的作用。默认是 SESSION 模式，按会话轮流模式进行均衡打散，也可以选择用户名（USER）、客户端主机地址（HOST）、登录默认数据库（DATABASE）进行路由，比如在一个实例下有很多不同的用户或不同的数据库，象 SaaS 应用，可以选择其他路由模式，以确保一个用户不会严重影响其他用户。

- sharedserver_size

  指定线程池 Group 的个数，已经根据 CPU 核数进行默认设置，不需要调整。

- sharedserver_cpu_group

  可按 Thread Group 为单位进行工作线程的 CPU 绑定，进行一定程度的资源限制和隔离。指定格式为“Group_ID:cpu_list [;Group_ID:cpu_list]”，也可以用”*”来代表所有的 Thread Group。在“route_mode”提到的场景中，可以确保 SaaS 的单个租户不会用完所有的 CPU 资源。

- sharedserver_cpu_sample

  对部份执行时间较长（达到 big_query 的一半）的 SQL 进行指定次数的 CPU 耗用统计（默认为 1000 次），可用以评估 SQL 的 CPU 资源耗用情况。

- sharedserver_normal_weights

  指定事务队列之外的单队列并发度比例，和“oversubscribe”相乘即是单个队列允许的工作线程并发度，默认值为 25%，既单个队列可以占据四分之一的工作线程以防止饿死情况。

- sharedserver_trans_weights

  指定事务队列的并发度比例，和“oversubscribe”相乘即是单个队列允许的工

作线程并发度，默认值为 25%，既单个队列可以占据四分之一的工作线程以防止饿死情况。

- sharedserver_relese_rolocks

  当 transaction_isolation 设置为 READ-COMMITTED 时，只读事务内的 MDL 锁不需要一直保留，可以在每个 SQL 语句执行完后及时释放。比如对于从库，应用也可能会开启只读事务，同时也会有备份任务存在，会产生 MDL 锁冲突，导致备份一直无法启动，这时可以开启此开关。

## 可观测性

由于所有的 SQL 请求和会话都经过线程池，使得线程池天生具有极强的可观测性能力，通过输出视图的方式可以将这些能力开放出来。在线程池中我们集成了以下几个主要的视图，可以优化非定制 MySQL 内核在可测性上的不足。

1. SHAREDSERVER_INFO

   用于揭示线程池自身的运行状态。

2. SHAREDSERVER_RUNNING

   活跃会话信息，可以揭示正在执行的 SQL 请求的情况，大家常用的的“SHOW PROCESSLIST”执行时间太慢了，并且输出了所有的会话，在成千上万并发时会形成很大的干扰，而这个视图不仅包含了“SHOW PROCESS”的所有列，还有更多的会话级性能指标，并且可以高频执行，形成会话的动态快照。

3. SHAREDSERVER_SQLTEXT & SHAREDSERVER_SQLSTATS_*

   通过线程池可以记录每个执行的 SQL 的诸多信息，并且可以根据 SQL 的特征串归类汇总，达到与商业数据库同样的性能分析能力。

4. SHAREDSERVER_PERFSTAT & SHAREDSERVER_PERFSTAT_HIS

实例维度的性能指标，实现每秒精准输出，并可在线查看 5 分钟的历史。

5. SHAREDSERVER_SESSIONS_*

按会话维度汇总的性能指标，并且允许查看不同会话的部份特定选项设置。

下面我们来一个一个地看一下：

- 视图：SHAREDSERVER_INFO

作用：观察线程运行情况，不包含任何实例相关性能信息，此表信息仅用于排查线程池的运行情况，比如在竞争冲突很严重的场景，前端可能会有卡的感觉，这时可以查询这个图进行诊断。

| 列名 | 说明 |
| --- | --- |
| GROUP_ID | Thread Group 编号 |
| DB_CONN | 当前 Thread Group 服务的用户连接数 |
| THD_CNT | 当前 Thread Group 起动的 Worker 总数 |
| THD_CACH | 当前 Thread Group 缓存的 Worker 数量以减少线程创建 |
| THD_IDLE | 当前 Thread Group 闲置的 Worker 数量 |
| THD_DED | 当前 Thread Group 独占模式运行的 Worker 数量 |
| THD_ACT | 当前 Thread Group 被唤醒的 Worker 数量 |
| THD_BUSY | 当前 Thread Group 正在处理请求的 Worker 数量 |
| THD_WAIT | 当前 Thread Group 处于等待状态的 Worker 数量 |
| AVG_PICK | 过去几个点 Listener 收到（唤醒）的平均请求数量 |
| AVG_BUSY | 过去几个点被唤醒并正在执行任务的平均 Worker 数量 |
| AVG_IDLE | 过去几个点被唤醒但还未获得任务的平均 Worker 数量 |

| | |
|---|---|
| AVG_RATIO | 过去几个点被唤醒并执行任务的平均 Worker 数量占比 |
| CON_ACT | 正在处理连接登录的 Worker 数量 |
| CON_WAIT | 处于登录等待状态的 Worker 数量 |
| CON_LEN | 登录请求队列中积压未处理的请求数量 |
| CON_GET | 从登录请求队列中获取任务的次数 |
| CON_WTM | 在登录请求队列中累计的等待时间（毫秒） |
| REQ_ACT | 正在处理 SQL 读取任务的 Worker 数量 |
| REQ_WAIT | 处于 SQL 读取任务等待状态的 Worker 数量 |
| REQ_LEN | SQL 读取任务队列中积压未处理的请求数量 |
| REQ_GET | 从 SQL 读取任务队列中获取任务的次数 |
| REQ_WTM | 在 SQL 读取任务队列中累计的等待时间（毫秒） |
| DEF_ACT | 正在处理默认（管理命令）请求任务的 Worker 数量 |
| DEF_WAIT | 处于默认（管理命令）请求任务等待状态的 Worker 数量 |
| DEF_LEN | 默认（管理命令）请求队列中积压未处理的请求数量 |
| DEF_GET | 从默认（管理命令）请求队列中获取任务的次数 |
| DEF_WTM | 在默认（管理命令）请求队列中累计的等待时间（毫秒） |
| DML2_ACT | 正在处理慢 DML 请求任务的 Worker 数量 |
| DML2_WAIT | 处于慢 DML 请求任务等待状态的 Worker 数量 |
| DML2_LEN | 慢 DML 请求队列中积压未处理的请求数量 |
| DML2_GET | 从慢 DML 请求队列中获取任务的次数 |
| DML2_WTM | 在慢 DML 请求队列中累计的等待时间（毫秒） |
| QRY2_ACT | 正在处理大查询请求任务的 Worker 数量 |

| | |
|---|---|
| QRY2_WAIT | 处于大查询请求任务等待状态的 Worker 数量 |
| QRY2_LEN | 大查询请求队列中积压未处理的请求数量 |
| QRY2_GET | 从大查询请求队列中获取任务的次数 |
| QRY2_WTM | 在大查询请求队列中累计的等待时间（毫秒） |
| TRX_ACT | 正在处理显式事务请求任务的 Worker 数量 |
| TRX_WAIT | 处于显式事务请求任务等待状态的 Worker 数量 |
| TRX_LEN | 显式事务请求队列中积压未处理的请求数量 |
| TRX_GET | 显式事务请求队列中获取任务的次数 |
| TRX_WTM | 在显式事务请求队列中累计的等待时间（毫秒） |
| DML_ACT | 正在处理慢 DML 请求任务的 Worker 数量 |
| DML_WAIT | 处于快 DML 请求任务等待状态的 Worker 数量 |
| DML_LEN | 快 DML 请求队列中积压未处理的请求数量 |
| DML_GET | 从快 DML 请求队列中获取任务的次数 |
| DML_WTM | 在快 DML 请求队列中累计的等待时间（毫秒） |
| QRY_ACT | 正在处理小查询请求任务的 Worker 数量 |
| QRY_WAIT | 处于小查询请求任务等待状态的 Worker 数量 |
| QRY_LEN | 小查询请求队列中积压未处理的请求数量 |
| QRY_GET | 从小查询请求队列中获取任务的次数 |
| QRY_WTM | 在小查询请求队列中累计的等待时间（毫秒） |

- 视图：SHAREDSERVER_RUNNING

作用：观察活跃会话信息（注意：仅显示通过线程池连接进来的会话，不包含后

端任务），比 SHOW PROCESSLIST 更快，并且包含更多会话级性能数据。

| 列名 | 说明 |
| --- | --- |
| HOST_NAME | 服务端主机名 |
| HOST_PORT | 服务端 TCP 端口 |
| HOST_UUID | 服务端 UUID 标识 |
| SESS_TPID | 线程池生成的会话 ID， |
| SESS_TIME | 会话登录时间（精确到毫秒） |
| SESS_THID | MySQL 后端的 Thread ID |
| SESS_USER | 会话登录的用户名 |
| SESS_HOST | 会话客户端机器名 |
| SESS_DBNM | 会话连接的默认 DB 名字 |
| SESS_CMMD | 会话正在执行的命令类型 |
| SESS_TIMZ | 会话的时区设置 |
| SESS_CSID | 会话的字符集设置 |
| PREV_SQLH | 上一个 SQL 的 Hash 值 |
| CURR_SQLH | 当前 SQL 的 Hash 值 |
| SESS_BLCK | 是事持有其他会话需要的 MDL 锁，即是否阻塞其他会话 |
| SESS_SPCW | 是否处于等待空间扩展关态（Space Wait） |
| SESS_TXTM | 显式事务开始时间（如果处于事务中） |
| SESS_SQLTM | 当前 SQL 请求开始时间 |
| TLCK_TIME | Table 锁等待开始时间 |
| TLCK_WAIT | Table 锁等待累计时间 |

| | |
|---|---|
| RLCK_TIME | Row 锁等待开始时间 |
| RLCK_WAIT | Row 锁等待累计时间 |
| MLCK_TIME | MDL 锁等待开始时间 |
| MLCK_WAIT | MDL 锁等待累计时间 |
| DISK_TIME | 磁盘读取等待开始时间 |
| DISK_WAIT | 磁盘读取等待累计时间 |
| TRAN_SQLS | 显式事务内执行的 SQL 总数 |
| DISK_READ | 当前会话累计磁盘访问等待次数 |
| ROWS_SENT | 当前会话发送给客户端的记录数 |
| ROWS_EXAM | 当前会话访问 InnoDB 的记录数 |
| TMPT_DISK | 当前会话累计生成磁盘临时表的次数 |
| TMPT_CREA | 当前会话累计生成临时表的次数 |
| HAWR_INST | Handler 层累计的 Insert 次数（Handler_write） |
| HAWR_UPDT | Handler 层累计的 Update 次数（Handler_update） |
| HAWR_DELT | Handler 层累计的 Delete 次数（Handler_delete） |
| HARD_FIST | Handler_read_first |
| HARD_LAST | Handler_read_last |
| HARD_PKEY | Handler_read_key |
| HARD_NEXT | Handler_read_next |
| HARD_PREV | Handler_read_prev |
| HARD_RAND | Handler_read_rnd |
| HARD_RNXT | Handler_read_rnd_next |

| | |
|---|---|
| SORT_PASS | Sort_merge_passes |
| SORT_ROWS | Sort_rows |
| BYTE_RECV | Bytes_received |
| BYTE_SENT | Bytes_sent |
| ROWS_FIND | 对于 DML 是 FOUND_ROWS，其他操作为 0 |
| LAST_INST | 当前会话的 Last Insert ID |
| SESS_MEMR | 当前会话 mem_root 的大小 |
| SESS_STAT | 当前会话的状态（同 show processlist） |
| EXEC_FUJN | Select_full_join |
| EXEC_FRJN | Select_full_range_join |
| EXEC_RANG | Select_range |
| EXEC_RNCK | Select_range_check |
| EXEC_SCAN | Select_scan |
| EXEC_NOID | 未使用索引的次数 |
| EXEC_NGID | 未使用有效索引的次数 |
| SESS_SQLH | SQL Hash 值（需要开启 Performance Schema） |
| SESS_SQLT | 正在执行的 SQL 文本串 |

- 视图：SHAREDSERVER_SQLTEXT

作用：通过线程池连接执行的 DML 和 SELECT 语句特征（去掉具体值）列表。

例如：

```
mysql> SELECT SQL_HASH, SQL_TEXT FROM
    ->   INFORMATION_SCHEMA.SHAREDSERVER_SQLTEXT
    ->   LIMIT 5;
+------------+-----------------------------------------------------------------------------------------+
| SQL_HASH   | SQL_TEXT                                                                                |
+------------+-----------------------------------------------------------------------------------------+
| 3187428896 | SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORDER BY c                                |
| 2560518885 | SELECT c FROM sbtest1 WHERE id=?                                                        |
| 2837905032 | SELECT @@version_comment LIMIT ?                                                        |
| 2312534216 | SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ?                                          |
| 3830222792 | SELECT tlck_time, tlck_wait, tran_sqls FROM information_schema.SHAREDSERVER_RUNNING |
+------------+-----------------------------------------------------------------------------------------+
5 rows in set (0.00 sec)
```

| 列名 | 说明 |
| --- | --- |
| SQL_HOST | 服务端主机名 |
| SQL_PORT | 服务端 TCP 端口 |
| SQL_UUID | 服务端 UUID 标识 |
| SQL_HASH | SQL 特征串的 Hash 值 |
| SQL_MEMR | SQL 需要的额外 mem_root 内存大小 |
| SQL_TEXT | SQL 特征串 |

- 视图：SHAREDSERVER_SQLSTATS_{ALL | 60 | 300 | 900}

作用：通过线程池连接执行的 DML 和 SELECT 语句的部份性能指标，按特征 Hash 值进行归并。

例如：

```
mysql> SELECT SQL_HASH, SQL_EXEC, SQL_ELA1, SQL_ELA2, SQL_EROW, SQL_SROW, SQL_FROW
    ->   FROM INFORMATION_SCHEMA.SHAREDSERVER_SQLSTATS_ALL
    ->   LIMIT 5;
+------------+----------+----------+----------+----------+----------+----------+
| SQL_HASH   | SQL_EXEC | SQL_ELA1 | SQL_ELA2 | SQL_EROW | SQL_SROW | SQL_FROW |
+------------+----------+----------+----------+----------+----------+----------+
| 3187428896 |   272724 |     4190 |     4190 | 54544800 | 27272400 | 27272400 |
| 3503037124 |        1 |        0 |        0 |        5 |        5 |        5 |
| 2560518885 |  2727239 |    19745 |    19745 |  2727240 |  2727240 |  2727240 |
| 2837905032 |        2 |        0 |        0 |        2 |        2 |        2 |
| 2312534216 |   272724 |     4645 |     4645 | 27272400 | 27272400 | 27272400 |
+------------+----------+----------+----------+----------+----------+----------+
5 rows in set (0.00 sec)
```

| 列名 | 说明 |
| --- | --- |
| SQL_HOST | 服务端主机名 |

| | |
|---|---|
| SQL_PORT | 服务端 TCP 端口 |
| SQL_UUID | 服务端 UUID 标识 |
| SQL_HASH | SQL 特征串的 Hash 值 |
| SQL_LAST | 最后一次执行时间 |
| SQL_EXEC | 累计执行次数 |
| SQL_ELA1 | 前端执行累计时长（含线程池队列等待时间） |
| SQL_ELA2 | 后端执行累计时长（不含线程池队列等待时间） |
| SQL_SAMP | CPU 执行时间采样次数（仅对执行时间较长的 SQL 采样） |
| SQL_CPUT | CPU 采样累计执行时间 |
| SQL_TLCK | Table 锁累计等待时间 |
| SQL_RLCK | Row 锁累计等待时间 |
| SQL_MLCK | MDL 锁累计等待时间 |
| SQL_IOWT | 磁盘 IO 累计等待时间 |
| SQL_IOCT | 磁盘 IO 累计等待次数 |
| SQL_EROW | SQL 累计访问 InnoDB 的记录数 |
| SQL_SROW | SQL 累计发送给客户端的记录数 |
| SQL_FROW | SQL（DML）累计更新记录数（FOUND_ROWS） |
| SQL_SORT | SQL 累计排序记录数 |
| SQL_SMPS | SQL 累计 Sort_merge_passes |
| SQL_RECV | SQL 累计接受客户端字节数 |
| SQL_SENT | SQL 累计发送客户端字节数 |
| SQL_SCAN | SQL 累计 Select_scan |

| | |
|---|---|
| SQL_RANG | SQL 累计 Select_range |
| SQL_RCHK | SQL 累计 Select_range_check |
| SQL_FFJN | SQL 累计 Select_full_join |
| SQL_FRJN | SQL 累计 Select_full_range_join |
| SQL_NOID | SQL 累计无索引执行次数 |
| SQL_NGID | SQL 累计无有效索引执行次数 |
| SQL_TMPD | SQL 累计磁盘临时表使用次数 |
| SQL_TMPT | SQL 累计临时表创建次数 |
| SQL_HWRT | Handler 层 SQL 累计的 Insert 次数（Handler_write） |
| SQL_HUPD | Handler 层 SQL 累计的 Update 次数（Handler_update） |
| SQL_HDEL | Handler 层 SQL 累计的 Delete 次数（Handler_delete） |
| SQL_HRFT | SQL 累计 Handler_read_first |
| SQL_HRLT | SQL 累计 Handler_read_last |
| SQL_HRKY | SQL 累计 Handler_read_key |
| SQL_HRXT | SQL 累计 Handler_read_next |
| SQL_HPRV | SQL 累计 Handler_read_prev |
| SQL_HRND | SQL 累计 Handler_read_rnd |
| SQL_HRRT | SQL 累计 Handler_read_rnd_next |
| SQL_LONG | SQL 累计计入慢查询次数 |
| SQL_MEMR | SQL 需要的额外 mem_root 内存大小 |

其中：

“ALL”表示取出所有记录，“60”表示仅显示 60 秒内执行过的 SQL 信息，“300”

和“900”同理

- 视图：SHAREDSERVER_PERFSTAT & SHAREDSERVER_PERFSTAT_HIS

作用：实例秒级性能指标信息，HIS 表可显示最近 5 分钟的历史信息。

例如：

```
mysql> SELECT DATA_TIME, HOST_LOAD01, HOST_LOAD05, HOST_LOAD15
    ->   FROM information_schema.SHAREDSERVER_PERFSTAT_HIS
    ->   LIMIT 5;
+---------------+-------------+-------------+-------------+
| DATA_TIME     | HOST_LOAD01 | HOST_LOAD05 | HOST_LOAD15 |
+---------------+-------------+-------------+-------------+
| 1686896227288 |           0 |           0 |           0 |
| 1686896228294 |           0 |           0 |           0 |
| 1686896229300 |           0 |           0 |           0 |
| 1686896230307 |           0 |           0 |           0 |
| 1686896231314 |           0 |           0 |           0 |
+---------------+-------------+-------------+-------------+
5 rows in set (0.01 sec)
```

| 列名 | 说明 |
|---|---|
| DATA_TIME | 数据取样时间 |
| DATA_VERS | 数据取样序号 |
| DATA_INTVAL | 数据取样间隔时间 |
| HOST_NAME | 服务端主机名 |
| HOST_PORT | 服务端 TCP 端口 |
| HOST_UUID | 服务端 UUID 标识 |
| HOST_LOAD01 | Load Average 1 Min |
| HOST_LOAD05 | Load Average 5 Min |
| HOST_LOAD15 | Load Average 15 Min |
| HOST_MEMALL | 实例所在机器总内存大小（MB） |
| HOST_MEMFRE | 实例所在机器可用内存大小（MB） |
| HOST_MEMBUF | 实例所在机器 Buffers 内存大小（MB） |
| HOST_MEMCAH | 实例所在机器 Cached 内存大小（MB） |

| | |
|---|---|
| HOST_PSWPIN | 实例所在机器 Swap In 块数 |
| HOST_PSWPOU | 实例所在机器 Swap Out 块数 |
| HOST_CPSALL | 实例所在机器总 CPU 量 |
| HOST_CPUUSR | 实例所在机器用户 CPU 使用量 |
| HOST_CPUSYS | 实例所在机器 SYS CPU 使用量 |
| HOST_IBYTES | 实例所在机器网络进向字节 |
| HOST_IERROR | 实例所在机器网络进向错误数 |
| HOST_IDROPS | 实例所在机器网络进向丢包数 |
| HOST_OBYTES | 实例所在机器网络出向字节 |
| HOST_OERROR | 实例所在机器网络出向错误数 |
| HOST_ODROPS | 实例所在机器网络出向丢包数 |
| HOST_NROPEN | 实例所在机器文件句柄使用比例 |
| PROC_CPUUSR | MySQL 进程用户态 CPU 使用量 |
| PROC_CPUSYS | MySQL 进程内核态 CPU 使用量 |
| PROC_MEMALL | MySQL 进程 VM 内存大小 |
| PROC_MEMRSS | MySQL 进程常驻内存大小 |
| PROC_THDNUM | MySQL 进程下的线程个数 |
| PROC_INBLKS | MySQL 从块设备读取的块数 |
| PROC_OUBLKS | MySQL 向块设备写出的块数 |
| MYDB_MAXCON | MySQL 实例配置最大连接数 |
| MYDB_SESCNT | MySQL 实例当前连接数 |
| MYDB_SESRUN | MySQL 实例活跃连接数 |

| | |
|---|---|
| MYDB_DATARD | MySQL 实例数据文件所在盘的读请求数 |
| MYDB_DATART | MySQL 实例数据文件所在盘的读请求时长 |
| MYDB_DATAWR | MySQL 实例数据文件所在盘的写请求数 |
| MYDB_DATAWT | MySQL 实例数据文件所在盘的写请求时长 |
| MYDB_BLOGRD | MySQL 实例 Binlog 文件所在盘的读请求数 |
| MYDB_BLOGRT | MySQL 实例 Binlog 文件所在盘的读请求时长 |
| MYDB_BLOGWR | MySQL 实例 Binlog 文件所在盘的写请求数 |
| MYDB_BLOGWT | MySQL 实例 Binlog 文件所在盘的写请求时长 |
| MYDB_TEMPRD | MySQL 实例临时文件所在盘的读请求数 |
| MYDB_TEMPRT | MySQL 实例临时文件所在盘的读请求时长 |
| MYDB_TEMPWR | MySQL 实例临时文件所在盘的写请求数 |
| MYDB_TEMPWT | MySQL 实例临时文件所在盘的写请求时长 |
| MYDB_DATAFR | MySQL 实例数据盘可用空间 |
| MYDB_BLOGFR | MySQL 实例 Binlog 盘可用空间 |
| MYDB_TEMPFR | MySQL 实例临时文件盘可用空间 |
| MYDB_EXECNT | MySQL 实例 SQL 执行次数 |
| MYDB_EXETIM | MySQL 实例 SQL 执行时长 |
| MYDB_TRXCNT | MySQL 实例事务执行次数 |
| MYDB_TRXSQL | MySQL 实例事务 SQL 执行次数 |
| MYDB_TRXTIM | MySQL 实例事务执行时长 |
| MYDB_DSKCNT | MySQL 数据块磁盘读等待次数 |
| MYDB_DSKTIM | MySQL 数据库磁盘读等待时长 |

| | |
|---|---|
| MYDB_ROWEXM | MySQL 实例访问 InnoDB 的记录数 |
| MYDB_ROWSNT | MySQL 实例发送给客户端的记录数 |
| MYDB_LCKTAB | MySQL 实例 Table 锁时长 |
| MYDB_LCKROW | MySQL 实例 Row 锁时长 |
| MYDB_LCKMDL | MySQL 实例 MDL 锁时长 |
| MYDB_EXEINS | MySQL 实例 Insert 次数（Handler_write） |
| MYDB_EXEUPD | MySQL 实例 Update 次数（Handler_update） |
| MYDB_EXEDEL | MySQL 实例 Delete 次数（Handler_delete） |
| MYDB_EXERNG | MySQL 实例 Select_range 次数 |
| MYDB_EXESCN | MySQL 实例 Select_scan 次数 |
| MYDB_EXESLW | MySQL 实例慢查询次数 |
| MYDB_TMPDSK | MySQL 实例磁盘临时文件使用数 |
| MYDB_TMPCRE | MySQL 实例临时表使用数 |
| MYDB_NETRCV | MySQL 实例接收字节数 |
| MYDB_NETSNT | MySQL 实例发送字节数 |
| MYDB_SRTROW | MySQL 实例排序记录数 |
| MYDB_SRTPAS | MySQL 实例 Sort_merge_passes |
| MYDB_ERRSIZ | MySQL 实例 Error 文件大小 |
| MYDB_ERRINC | MySQL 实例 Error 文件增量 |
| MYDB_SLWSIZ | MySQL 实例 Slow Log 文件大小 |
| MYDB_GENSIZ | MySQL 实例 General Log 文件大小 |
| MYDB_IBDSIZ | MySQL 实例 ibdata1 文件大小 |

| | |
|---|---|
| MYDB_UD1SIZ | MySQL 实例 undo001/undo_001 文件大小 |
| MYDB_UD2SIZ | MySQL 实例 undo002/undo_002 文件大小 |
| MYDB_IBTSIZ | MySQL 实例 ibtmp1 文件大小 |

- 视图：SHAREDSERVER_SESSIONS_{CUR | ALL | 60 | 300 | 900}

作用：通过线程池连接的会话的部份性能指标及变量设置值。“CUR”表示仅列出当前会话，“ALL”表示列出所有会话，“60”表示仅显示 60 秒内执行过的会话信息，“300”和“900”同理。

例如：

| 列名 | 说明 |
|---|---|
| HOST_NAME | 服务端主机名 |
| HOST_PORT | 服务端 TCP 端口 |
| HOST_UUID | 服务端 UUID 标识 |
| SESS_TPID | 线程池生成的会话 ID， |
| SESS_THID | 会话登录时间（精确到毫秒） |
| SESS_USER | MySQL 后端的 Thread ID |
| SESS_HOST | 会话登录的用户名 |
| SESS_TIMZ | 会话的时区设置 |
| SESS_CSID | 会话的字符集设置 |
| PREV_SQLH | 上一个 SQL 的 Hash 值 |
| CURR_SQLH | 当前 SQL 的 Hash 值 |
| SESS_RWTX | 是否在写事务中 |
| SESS_MDLK | 是否在等待 MDL 锁中 |

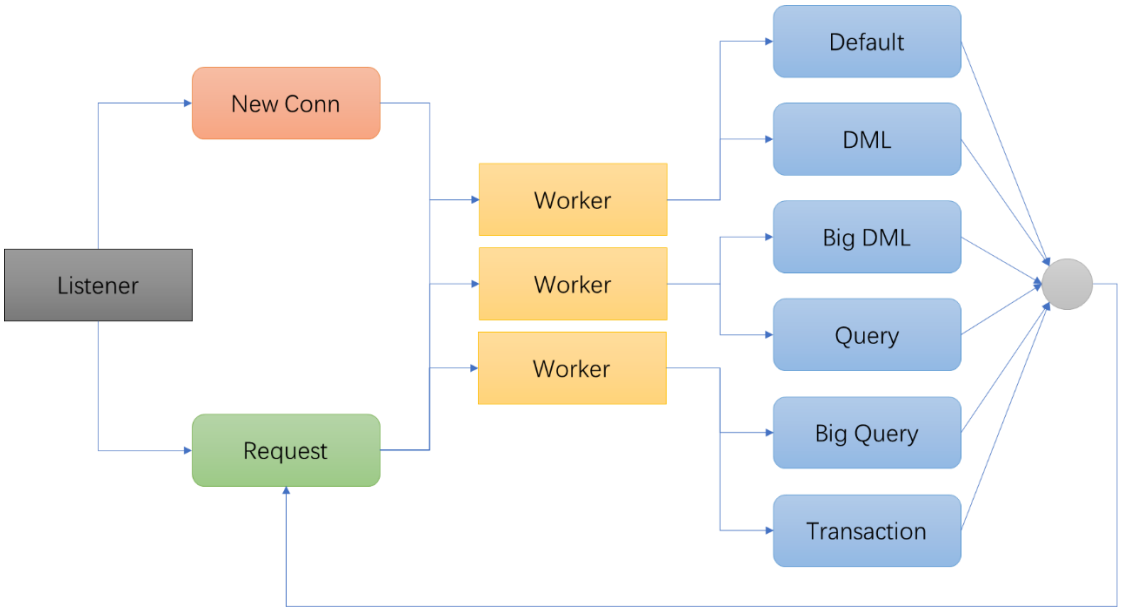
| | |
|---|---|
| SESS_TLCK | 是否开启 TABLE_LOCK 选项（OPTION_LOCK_TABLE） |
| SESS_LCKT | 是否开启表锁模式（locked_tables_mode） |
| SESS_TXTM | 事务开始时间 |
| SESS_SQTM | SQL 开始时间 |
| TLCK_TIME | Table 锁开始时间 |
| RLCK_TIME | Row 锁开始时间 |
| MLCK_TIME | MDL 锁开始时间 |
| DISK_TIME | 磁盘 IO 开始时间 |
| EXEC_COUT | SQL 执行次数 |
| EXEC_TIME | SQL 执行时长 |
| EXEC_TLTM | Table 锁累计时长 |
| EXEC_RLTM | Row 锁累计时长 |
| EXEC_MLTM | MDL 锁累计时长 |
| EXEC_IOWC | 磁盘 IO 等待累计次数 |
| EXEC_IOWT | 磁盘 IO 等待累计时长 |
| EXEC_NOID | SQL 无索引次数 |
| EXEC_NGID | SQL 无有效索引次数 |
| TMPT_DISK | 当前会话累计生成磁盘临时表的次数 |
| TMPT_CREA | 当前会话累计生成临时表的次数 |
| HAWR_INST | 会话累计的 Insert 次数（Handler_write） |
| HAWR_UPDT | Handler 层累计的 Update 次数（Handler_update） |
| HAWR_DELT | Handler 层累计的 Delete 次数（Handler_delete） |

| | |
|---|---|
| HARD_FIST | Handler_read_first |
| HARD_LAST | Handler_read_last |
| HARD_PKEY | Handler_read_key |
| HARD_NEXT | Handler_read_next |
| HARD_PREV | Handler_read_prev |
| HARD_RAND | Handler_read_rnd |
| HARD_RNXT | Handler_read_rnd_next |
| BYTE_RECV | 从客户端接收的字节数 |
| BYTE_SENT | 向客户端发送的字节数 |
| SORT_PASS | 会话累计 Sort_merge_passes |
| SORT_ROWS | 会话累计排序记录数 |
| EXEC_FUJN | Select_full_join |
| EXEC_FRJN | Select_full_range_join |
| EXEC_RANG | Select_range |
| EXEC_RNCK | Select_range_check |
| EXEC_SCAN | Select_scan |
| EXEC_SLOW | 会话慢查询次数 |
| ROWS_EXAM | 会话访问 InnoDB 的记录数 |
| ROWS_SENT | 会话发送给客户端的记录数 |
| LAST_INST | 当前会话的 Last Insert ID |
| SESS_MEMR | 当前会话 mem_root 的大小 |
| SESS_OPSZ | Output Packet Length |

| | |
|---|---|
| SESS_OPAZ | Output Packet Allocated Size |
| SESS_UVAR | 用户变量个数 |
| V_SELIMIT | 会话级 Select_limit 设置 |
| V_LCKWTMO | 会话级 Lock_wait_timeout 设置 |
| V_READTMO | 会话级 net_read_timeout 设置 |
| V_WAITTMO | 会话级 net_wait_timeout 设置 |
| V_WRITTMO | 会话级 net_write_timeout 设置 |
| V_INTATMO | 会话级 net_interactive_timeout 设置 |
| V_EXECTMO | 会话级 max_execution_time 设置 |
| V_NETBUFF | 会话级 net_buffer_length 设置 |
| V_READBUF | 会话级 read_buff_size 设置 |
| V_SORTBUF | 会话级 sortbuff_size 设置 |
| V_JOINBUF | 会话级 join_buff_size 设置 |
| V_MAXPKCT | 会话级 max_allowed_packet 设置 |
| V_LONGQRY | 会话级 long_query_time 设置 |
| V_TMPSIZE | 会话级 tmp_table_size 设置 |
| V_OPTBITS | 会话级 option_bits 设置 |
| V_OPTSWCH | 会话级 optimizer_switch 设置 |
| V_SQLMODE | 会话级 sql_mode 设置 |
| V_AINCVAL | 会话级 auto_increment_increment 设置 |
| V_AINCOFF | 会话级 auto_increment_offset 设置 |
| V_TXISOLT | 会话级 transaction_isolation 设置 |

## 内部设计

下图简要地介绍了工作流程，其中直角长方型为线程类型，只分为监听线程和工作线程，而圆角的长方型则为内部的不同队列，可以看到各种不同的负载请求可以在各自的队列中按预定设定好的并发度去执行，不会出现相互影响的现象，从而可以胜任通用业务场景。




## 场景测试

纯内存点查测试，可有效验证线程池插件会不会影响性能，可以看到带线程池也可以将点查压到 170 多万的 QPS，所以不用担心性能问题：

```
Threads started!

[ 10s ] thds: 384 tps: 1774996.94 qps: 1774996.74 (r/w/o: 1774996.74/0.00/0.00)
[ 20s ] thds: 384 tps: 1767055.83 qps: 1767055.83 (r/w/o: 1767055.83/0.00/0.00)
[ 30s ] thds: 384 tps: 1765855.58 qps: 1765855.68 (r/w/o: 1765855.68/0.00/0.00)
[ 40s ] thds: 384 tps: 1763760.13 qps: 1763760.23 (r/w/o: 1763760.23/0.00/0.00)
[ 50s ] thds: 384 tps: 1754940.17 qps: 1754940.17 (r/w/o: 1754940.17/0.00/0.00)
[ 60s ] thds: 384 tps: 1770161.83 qps: 1770161.73 (r/w/o: 1770161.73/0.00/0.00)
[ 70s ] thds: 384 tps: 1772117.21 qps: 1772117.21 (r/w/o: 1772117.21/0.00/0.00)
[ 80s ] thds: 384 tps: 1765203.85 qps: 1765203.75 (r/w/o: 1765203.75/0.00/0.00)
[ 90s ] thds: 384 tps: 1765660.06 qps: 1765660.06 (r/w/o: 1765660.06/0.00/0.00)
[ 100s ] thds: 384 tps: 1767989.81 qps: 1767989.81 (r/w/o: 1767989.81/0.00/0.00)
[ 110s ] thds: 384 tps: 1767644.23 qps: 1767644.43 (r/w/o: 1767644.43/0.00/0.00)
```

接下来测试秒杀场景，用 1024 个客户端去更新同一条记录，测试的 SQL 语句为

“update t_binlog set col2 = col2 - 1 where id = 1”，在开启线程池的情况下，可以达到 2W 多的 TPS，如下图所示：

```
2023-07-02 18:22:48  MYDBTEST: MySQL Database Test Utility , Release 2.0.1
2023-07-02 18:22:48  (@) Copyright Lou Fangxin (AnySQL.net) 2012 - 2022, all rights reserved.
2023-07-02 18:22:48  ======================================== CASE SUMMARY ========================================
2023-07-02 18:22:48  SQL01 tm=62.3s exe=1311308=21059/s f=0 row=1311308=1.00/e ela=63230.71s avg=48219us
2023-07-02 18:22:48  SQL01     10 ms  exec=    245726, ela=     1517066 ms, avg=    6173 us, pct= 18.74%,  18.74%
2023-07-02 18:22:48  SQL01     20 ms  exec=      2957, ela=       37550 ms, avg=   12698 us, pct=  0.23%,  18.96%
2023-07-02 18:22:48  SQL01     30 ms  exec=      2809, ela=       70375 ms, avg=   25053 us, pct=  0.21%,  19.18%
2023-07-02 18:22:48  SQL01     40 ms  exec=      3581, ela=      126394 ms, avg=   35295 us, pct=  0.27%,  19.45%
2023-07-02 18:22:48  SQL01     50 ms  exec=      7258, ela=      332174 ms, avg=   45766 us, pct=  0.55%,  20.01%
2023-07-02 18:22:48  SQL01     60 ms  exec=    680421, ela=    38603240 ms, avg=   56734 us, pct= 51.89%,  71.89%
2023-07-02 18:22:48  SQL01     70 ms  exec=    368555, ela=    22543844 ms, avg=   61168 us, pct= 28.11%, 100.00%
2023-07-02 18:22:48  SQL01     80 ms  exec=         1, ela=          70 ms, avg=   70146 us, pct=  0.00%, 100.00%
2023-07-02 18:22:48  Total tm=62.3s tran=1311308=21059/s, qtps=1311308=21059/s, ela=63231.00s, avg=48219us
```

在关闭线程池的情况下，更新同一行记录的性能惨不忍睹，如下图所示：

```
2023-07-02 18:34:11  MYDBTEST: MySQL Database Test Utility , Release 2.0.1
2023-07-02 18:34:11  (@) Copyright Lou Fangxin (AnySQL.net) 2012 - 2022, all rights reserved.
2023-07-02 18:34:11  ======================================== CASE SUMMARY ========================================
2023-07-02 18:34:11  SQL01 tm=279.8s exe=51200=182/s f=0 row=51200=1.00/e ela=286355.91s avg=5592888us
2023-07-02 18:34:11  SQL01     10 ms  exec=         2, ela=           7 ms, avg=    3525 us, pct=  0.00%,   0.00%
2023-07-02 18:34:11  SQL01    100 ms  exec=         1, ela=          93 ms, avg=   93913 us, pct=  0.00%,   0.01%
......
2023-07-02 18:34:11  SQL01   3390 ms  exec=         2, ela=        6778 ms, avg= 3389068 us, pct=  0.00%,   1.73%
2023-07-02 18:34:11  SQL01   3400 ms  exec=         3, ela=       10174 ms, avg= 3391464 us, pct=  0.01%,   1.74%
......
2023-07-02 18:34:11  SQL01   5110 ms  exec=         4, ela=       20414 ms, avg= 5103638 us, pct=  0.01%,   2.97%
2023-07-02 18:34:11  SQL01   5120 ms  exec=     49681, ela=   281880017 ms, avg= 5673799 us, pct= 97.03%, 100.00%
2023-07-02 18:34:11  Total tm=279.8s tran=51200=182/s, qtps=51200=182/s, ela=286355.94s, avg=5592889us
```

接下来测试大查询场景，在这里我们用“select count(*) from sbtest1”语句进行测试，测试表有 200 万条记录，在开启线程池的情况下，QPS 稍低一些，CPU 资源使用一半核的样子（说明复杂的 SQL 被线程池归入慢查询队列，最大并发被有效限制），如下图所示：

```
2023-07-02 18:50:29  MYDBTEST: MySQL Database Test Utility , Release 2.0.1
2023-07-02 18:50:29  (@) Copyright Lou Fangxin (AnySQL.net) 2012 - 2022, all rights reserved.
2023-07-02 18:50:29  ======================================== CASE SUMMARY ========================================
2023-07-02 18:50:29  SQL01 tm=169.9s exe=12800=75/s f=0 row=12800=1.00/e ela=41562.83s avg=3247096us
2023-07-02 18:50:29  SQL01    410 ms  exec=         2, ela=         818 ms, avg=  409262 us, pct=  0.02%,   0.02%
2023-07-02 18:50:29  SQL01    420 ms  exec=         3, ela=        1252 ms, avg=  417542 us, pct=  0.02%,   0.04%
......
2023-07-02 18:50:29  SQL01   3230 ms  exec=       175, ela=      564412 ms, avg= 3225215 us, pct=  1.37%,  43.06%
2023-07-02 18:50:29  SQL01   3240 ms  exec=       204, ela=      659949 ms, avg= 3235047 us, pct=  1.59%,  44.66%
......
2023-07-02 18:50:29  SQL01   4200 ms  exec=         1, ela=        4193 ms, avg= 4193518 us, pct=  0.01%,  99.99%
2023-07-02 18:50:29  SQL01   4280 ms  exec=         1, ela=        4274 ms, avg= 4274056 us, pct=  0.01%, 100.00%
2023-07-02 18:50:29  Total tm=169.9s tran=12800=75/s, qtps=12800=75/s, ela=41562.85s, avg=3247097us
```

在不开启线程池时，QPS 稍高，但 CPU 被打满了，如下图所示：

```
2023-07-02 19:03:39  MYDBTEST: MySQL Database Test Utility , Release 2.0.1
2023-07-02 19:03:39  (@) Copyright Lou Fangxin (AnySQL.net) 2012 - 2022, all rights reserved.
2023-07-02 19:03:39  ======================================== CASE SUMMARY ========================================
2023-07-02 19:03:39  SQL01 tm=134.4s exe=12800=95/s f=0 row=12800=1.00/e ela=33441.13s avg=2612588us
2023-07-02 19:03:39  SQL01    530 ms  exec=         1, ela=         524 ms, avg=  524565 us, pct=  0.01%,   0.01%
2023-07-02 19:03:39  SQL01    550 ms  exec=         1, ela=         549 ms, avg=  549343 us, pct=  0.01%,   0.02%
......
2023-07-02 19:03:39  SQL01   2590 ms  exec=       270, ela=      697906 ms, avg= 2584839 us, pct=  2.11%,  38.98%
2023-07-02 19:03:39  SQL01   2600 ms  exec=       291, ela=      755206 ms, avg= 2595212 us, pct=  2.27%,  41.26%
......
2023-07-02 19:03:39  SQL01   3260 ms  exec=         2, ela=        6507 ms, avg= 3253858 us, pct=  0.02%,  99.99%
2023-07-02 19:03:39  SQL01   3280 ms  exec=         1, ela=        3279 ms, avg= 3279583 us, pct=  0.01%, 100.00%
2023-07-02 19:03:39  Total tm=134.4s tran=12800=95/s, qtps=12800=95/s, ela=33441.13s, avg=2612588us
```