# TXSQL 并行查询用户手册

2023-08-06 017:45

# 目录

# 发布链接

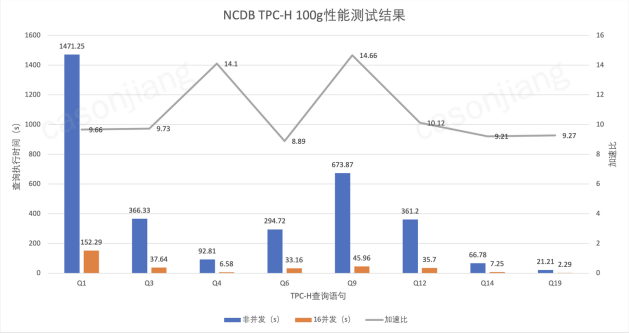
https://cloud.tencent.com/document/product/1003/81863

# 概要

腾讯云数据库 (TencentDB for MySQL) 和云原生数据库 (TDSQL-C MySQL) 同步发布并行查询特性。打开并行查询功能开关，就可以自动识别大查询，利用并行查询能力，调动多核计算资源，大幅缩短大查询响应时间。

并行查询加速的原理，是将数据划分为互不重叠的数据分区，将查询分拆成可以并行的子任务，分发到若干并行线程上，每个并行线程分摊一些计算任务，而这些分区计算结果会经过汇总运算，再返回给用户。显然，参与并行计算的核心越多，可并行子任务耗时就越短。按照 Amdahl 定律，加速效果会受限于并行子任务在总计算任务中的比重。

对于 TDSQL-C MySQL 环境下的 TPCH 100GB 性能测试，在并行度 16 时，有以下查询获得了明显提速。




# 背景

在 MySQL 生态里，各开源发行版只支持传统的单线程查询处理模式，即单条 SQL 处理涉及到的解析、优化和执行等阶段，都是在一个线程（称为用户线程）中完成的。这种处理模式可以有效地支持 TP 负载。因为 TP 负载的特点，是查询数量非常多，但其中的每个查询都只需要处理非常少的数据量，所以，提高查询并发数量（查询间并行），就可以有效提高系统的吞吐量，而对于查询耗时本身，主要是通过线程切换、缓存复用和访问路径优化等手段进行优化。

虽然长期以来 MySQL 主要是用于支持 TP 负载，但业务上还是会有一些 AP 查询，例如报表统计或者其他分析查询。这些查询虽然不多，但通常要处理比较大的数据量。对于 MySQL 单线程模式，即使当下有空闲硬件资源，也无法调动它们，只能白白浪费，并容忍这些超长的查询耗时。变通办法有，业务层面想办法将大查询拆成多个小查询，挖掘查询间并发能力，然后在业务层面进行合并，或者，采用 TP + AP 双系统，将这些 AP 查询转到专用的 AP 系统里。但无论是哪一种办法，都有相应的显著的成本（业务复杂度或者部署复杂度）。

对于 AP 查询，在同一份数据上，调动多核服务于同一个查询（查询内并行），无疑是查询加速和降本增效的重要措施。这就是并行查询( Parallel Query )。在并行查询中，用户线程会进行数据划分和任务分拆，同时充当协调者角色，协调多个工作线程并行地执行子任务，从而有效地利用空闲硬件来加速 AP 查询。事实上，各大商业数据库早已支持该功能，各大云厂商近年也纷纷提供了各自的并行查询能力。

# 快速使用

并行查询功能参数比较多，但大多都设置了合理的默认值，只需一个全局参数即可开启并行查询之旅。具体地说，只要规划了可用并行线程资源，大查询就可以自动并行起来。

我们将以 TPC-H q1 的简化形式，来演示如何启用并行查询。对 q1 进行简化是为了减少篇幅但又不影响表达。这个查询的特点，是单表扫描分组聚合，扫描行数很多，分组很少，是典型的报表运算。

```
SELECT l_returnflag, l_linestatus, sum(l_quantity) as sum_qty
FROM lineitem
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

从演示截图可以看到，并行查询使用了 4 个工作线程，有明显的提速效果。新的并行执行计划和原生串行执行计划相比，有三个变化：(1) 聚合运算拆分成了上下两段，用户线程和并行线程分别执行；(2) 引入了新的算子支持线程间的数据交换；(3) 表数据访问采用了并行扫描算子。


> 注1：虽然上图使用了两个参数，但并行度参数 (txsql_parallel_degree ) 只是为了强化对比，实际上它是默认值。参数设置可以查阅参数手册。

> 注2：演示中采用了树状执行计划打印 (EXPLAIN format=tree query)，这比传统的表状执行计划更加友好。

# 并行原理

下面以一个简单的例子介绍并行查询的基本原理。注：MySQL/InnoDB 物理存储为段页式，分区单位为页，这里用行演示分区的概念。

给定一个表 t 和如下分组聚合查询，

```
select x, count(*) from t group by x;
```

哈希聚合算法（迭代求值）过程如下表所示。算法迭代每一行，更新分组聚合状态。当所有数据行迭代结束时，就得到了分组聚合结果。假设聚合状态更新操作是恒定的，那么算法时间复杂度是 O(n) 的。

| x | y | Aggregate by hash |
|---|---|---|
| a | 1 | {a:1} |
| b | 2 | {a:1; b:1} |
| a | 3 | {a:2; b:1} |
| b | 4 | {a:2; b:2} |
| b | 5 | {a:2; b:3} |
| a | 6 | **{a:3; b:3}** |

行迭代

串行执行 (count)

如果用 k 个线程来加速这个查询（这里 k = 2），那么最好是先将数据表划分成 k*p 个分区（这里 p = 1）。这样，每个线程可以处理 p 个分区，产生一个局部结果。显然，这些局部结果并不是最终结果，需要进行合并处理。合并操作需要放在同一个线程（这里是用户线程）里执行，才能获得正确的最终结果。

数据分区

| x | y | Aggregate by hash |
|---|---|---|
| a | 1 | {a:1} |
| b | 2 | {a:1; b:1} |
| a | 3 | **{a:2; b:1}** |
| b | 4 | {b:1} |
| b | 5 | {b:2} |
| a | 6 | **{a:1; b:2}** |

分区1

分区2

分区并行 (count)

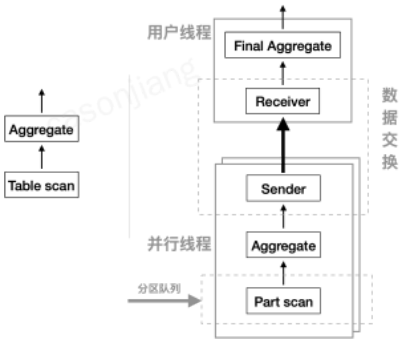
| x | count | Final Aggregate |
|---|---|---|
| a | 2 | {a:2} |
| b | 1 | {a:2; b:1} |
| a | 1 | {a:3; b:1} |
| b | 2 | **{a:3; b:3}** |

结果合并 (sum)

假设每个分区大小均衡，分区行数为 `n / kp`，合并行数 `m`，那么，算法时间复杂度为 `O(n/k + m)`。

显然并行加速比跟线程数 `k` 以及合并行数 `m` 相关。如果 `m` 远小于 `n` (`m << n`)，即局部聚集后数量剧减，那么加速比趋近于 `k`，但这是最理想的加速比。如果 `m` 近似于 `n` (`m == n`)，即数据大致表现出唯一性，那么，加速比可能趋近于 `k / (k + 1)`，该算法下并行可能会更慢一点，这时候需要切换到其他算法。

这两个模式的运算可以用代数算子图表示：



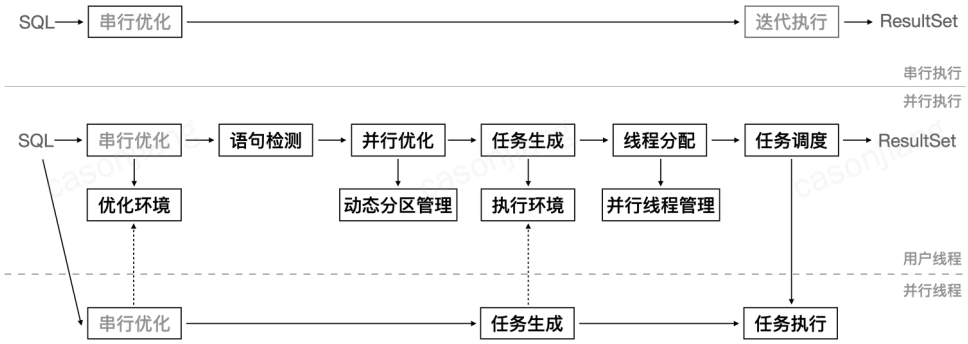

那么，可以总结并行查询的几个核心要素：

1. 数据分区。将原始表数据划分成不重叠的分区，并支持按分区读数据。
2. 任务拆分。将原始计划中的特殊运算拆分成“局部-整体”两段运算，除此之外，还要插入数据交换算子，支持跨线程数据传递。
3. 数据交换。支持在不同线程间传递数据。

# 并行设计

## 并行流程

基于上述核心要素，就可以将串行处理流程扩展成并行处理流程。




## 线程模型

并行查询中，用户线程负责数据分区和任务分拆，同时充当协调者角色（也称为协调线程），协调多个工作线程并行地执行子任务。用户线程还负责合并最终结果，返回给用户。工作线程执行并行子任务，并通过数据通道交换中间结果。

## 任务调度

协调线程按任务依赖顺序执行任务。每个任务是一个迭代子树，其执行即迭代执行，只是其输出都通过数据交换算子发送，而非直接返给用户。

> 注：火山迭代器模型 (Goetz Graefe 1994, Volcano— An Extensible and Parallel Query Evaluation System) 是关系数据库行式执行的经典模型。迭代器算子遵循相同的 open-next-close 接口，每次 next 返回一行数据。优化器选择合适的算子，组装成符合给定 SQL 语义的树状计算结构。 MySQL 8.0 引入火山迭代器模型 (WL#12074)，彻底重构计算层实现了统一的迭代逻辑。

考虑到任务之间的数据依赖关系，只需要同时调起上下游两个任务，更下游的任务即使调起也会处于空闲状态。按照任务依赖关系交替调起任务即可完成整个查询。

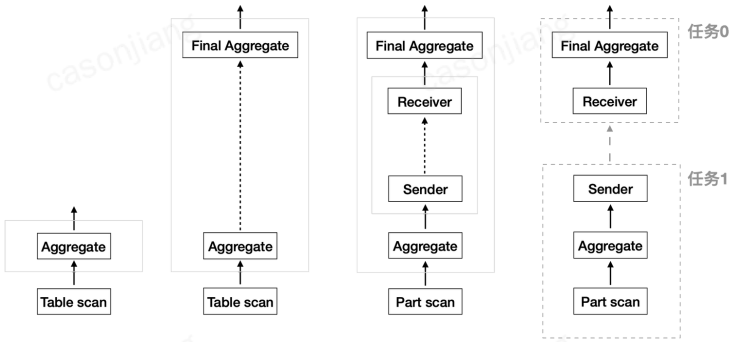
## 语句检测

并行查询自动检测语句环境、数据和执行计划，决定是否需要并行。

语句层面检测必须是动态查询，数据隔离级别是 RC 或 RR，数据处理量足够多（代价），执行计划层面检测迭代算子和函数是否可以并行。详情可查“兼容性手册”。

## 任务拆分

原始执行计划是一颗迭代算子树，任务拆分在其上构造一颗粗粒度的任务依赖树。大致上分为特殊算子拆分和插入数据交换算子两步，例如，




迭代器算子虽然很多，但是分为两大类：**(1) 流水线算子**和 **(2) 流水线断点算子**。流水线算子不需要暂存数据。例如条件过滤 (`Filter`)，虽然每次调用可能读入多行，但是只返回符合条件的那一行。流水线算子可以串联起来，基于一个行数据缓冲区依次进行运算。流水线断点算子则需要先暂存（又称物化）所有数据，进行特定运算，重新产生新的迭代序列。例如聚合 (`Aggregate`) 要先读入所有行更新内部状态，然后返回聚合结果。

在将串行执行计划拆分为串并行混合计划时，只需考虑在流水线断点处拆分，因为在流水线上拆分反而会引入数据行传输的开销。显然，一个复杂的计划可以有非常多的拆分点。目前的版本只支持最底层的拆分点，这样完整计划会分成上下两层。两层之间是一个或多个拆分点。其中上层总是由用户线程执行并向客户端返回数据，只有下层是并行执行的，所以也称为“**一阶段并行**”。后续版本会支持多层拆分，以及“**两阶段并行**”调度。

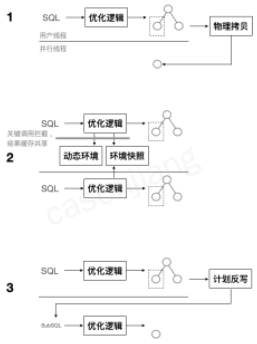
### 特殊运算拆分

| 特殊运算 | 并行计算方法 |
| --- | --- |
| **排序** | 局部外部排序，整体合并排序 |
| **聚合统计 – count** | 局部计数，整体求和 |
| **聚合统计 – sum** | 局部求和，整体求和 |
| **聚合统计 – average** | 如上分两步算出 sum 和 count，那么 average = sum / count |
| **聚合统计 – min/max** | 局部最值，整体最值 |
| **连接 – Nested Loop Join** | 驱动表并行扫描 |

| 特殊运算 | 并行计算方法 |
| --- | --- |
| **连接 – Hash Join(in mem)** | 各自构建完整 hash 表，probe 端并行扫描 |
| **聚合统计 – variance** | 如上算出 sum(x), count(x) 和 sum(xi^2), 再根据公式求值 |
| **聚合统计 – std** | variance 结果开平方 |
| **聚合统计 – var_samp** | 如上算出 sum(x), count(x) 和 sum(xi^2), 再根据公式求值 |
| **聚合统计 – stddev_samp** | var_samp 结果开平方 |

## 任务副本

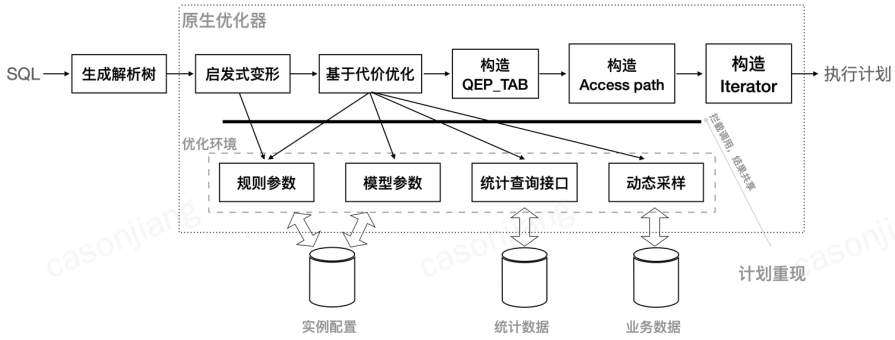
并行任务需要从用户线程分发到工作线程执行。在 SQL 语境下，任务有三种表达形式：子计划物理结构，完整计划的局部视图，或者子计划的 SQL 表示。对应地，有三种构造任务副本的方案：




并行查询实现采用了方案二，本质上复用了优化器的基础能力（复现给定语句的优化过程）。那么，工作线程会有额外的优化开销：对于大查询而言，优化开销远小于执行开销，可以忽略不计；对于小查询而言，记录优化环境会有微小的开销，因此不建议在高并发小查询中启用并行查询。但其优势也是显而易见的，**天然支持几乎所有普通函数和连接操作**，除非其定义上不支持并行或者需要底层适配。

### 优化环境

SQL 优化器可以简单理解为一个高度抽象的确定性计算模块，其输入是 SQL 和优化环境，输出是执行计划。对于相同的输入，就会有相同的优化路径，产生相同的输出。这个特性并非并行查询专用，例如，还可以用于导出生产实例上的优化环境，然后导入到离线环境进行问题分析。商业数据库基本上都已经支持优化环境导入导出。




## 数据分区

MySQL / InnoDB 表和索引都是 B+ 树。这是一颗平衡树。平衡树分区的做法，是基于某一个层次的所有节点进行划分。如果分区数量不是工作线程的整数倍，那就需要对余数分区进行更深层次划分。总分区数量太多和太少都可能有问题，太多会产生较大的分区开销，太少又可能会导致工作线程不均衡。

这些动态分区信息形成一个分区队列。SQL 执行计划的相应数据访问算子就改成从动态分区队列获取分区信息，并按分区读取数据。

事实上，MySQL 官方提供了并行动态分区功能 (WL#11720 – InnoDB: Parallel read of index)，但仅支持主索引正向扫描，而且代码结构比较模糊。所以，并行查询实现时进行了借鉴和重新设计，使动态分区逻辑可以支持各种计算需求。

## 数据通信

任务间传输数据，通过数据交换算子 (Exchange) 实现。数据交换算子由两个迭代算子组成。参与数据交换的线程，分为数据生产端和数据消费端两组，每一对生产端线程和消费端线程之间，会有一个无锁消息队列。相对于 MySQL 宽松行格式，数据消息格式为紧凑行格式。所以，对于一阶段并行，即消费端和生产端是 1:N 的关系，就需要 N 个消息队列。

# 常见问题

问题1：大规格情况下，若用到更多并行度，耗时是不是也会线性减少?

回答：正如“并行原理”一节分析，整体查询耗时的减少幅度取决于可并行部分占整体计算量的比重和并行线程工作量的均衡性。一般来说，投入线程越多，提速效果会更好，逼近理论极限值。优化没有止境，关键看业务需求。

问题2：开了并行查询，资源消耗会增加多少?

回答：并行查询相当于把单线程数据扫描变成多线程并行扫描，扫描速度倍增系数为线程个数。优化环境、数据交换队列和计划副本里的行缓冲区也会消耗额外的内存。优化环境内存开销与语句有关，数据交换队列开销与语句和并行度有关，它们都有相应的控制参数。

需要注意的是，优化环境开销对于最终不能通过并行检测的查询也存在，对于短查询也会有微小的影响。换个角度，如果有并行查询运行，导致资源消耗倍增，也会在一定程度上拖慢系统。所以，如果业务负载是高并发短查询，建议将并行查询可用线程资源设为 0 ，即可彻底关闭并行查询。线程池等短查询高并发场景下的优化手段，和并行查询这种大查询特性通常也不能很好地配合。

问题3：什么场景下的 SQL 能用到该特性?

回答：在可用线程资源足够多、查询足够大、没有锁等特殊语义时，SQL 可以自动并行执行。详情参考兼容性手册。

问题4：如何确定是否开启了并行查询?

回答：查看并行查询状态计数增长可以在实例级观察是否有运行过的并行查询。`SHOW PARALLEL PROCESSLIST` 可以看当前正在运行的并行查询。`EXPLAIN` 可以看一条语句是否会由并行查询支持运行。

问题5：并行查询结果和串行查询结果不同?

串并行查询结果集差异有两种情况，未定义行为和数值误差。由于 SQL 集合是“无序包”，除非在语法中显示指定 ORDER BY ，否则任意结果集顺序都是正确的。需要注意的是，排序字段上如果有 NULL ，那么结果集顺序也是未定义的，但这属于算子自身的行为。浮点运算存在一定误差，可能会导致尾部数位跳变。此外，由于参与线程变多，可能都会遇到导致 warning 的记录，所以，并行查询的 warnings 也可能会变多。

# 参考手册

## 兼容性手册

我们已经实现了具备如下特征的SQL语句的并行查询处理，并在逐渐完善更多的功能场景。

- 对于单表扫描：支持全表扫描、索引扫描、索引范围扫描、索引REF查询等扫描类型的正序、逆序扫描
- 对于多表连接：支持 Nested Loop Join 算法，以及 Semi Join、Anti Join、Outer Join 等连接类型
- 对于子查询：支持 derived table 的并行；**支持子查询并行(ncdb-3.1.11, cdb-20230330)**
- 对于数据类型：支持带多种数据类型的查询，包括：整型数据、字符型数据、浮点型数据、时间类型数据、以及（有运行时大小限制的）溢出类型数据。
- 普通运算符和函数原则上不限
- 聚合函数支持 COUNT/SUM/AVG/MIN/MAX, **STDDEV/VARIANCE (ncdb-3.1.10, cdb-20230330)**
- 支持UNION/UNION ALL查询
- 支持 traditional (默认格式) 、 json 和 tree 三种 EXPLAIN 格式
- **支持 EXPLAIN ANALYZE (ncdb-3.1.10, cdb-20230330)**

当前并行查询不支持的场景如下：

语句兼容性限制：

- 非查询语句不支持并行查询，包括 INSERT ... SELECT 和 REPLACE ... SELECT 。
- stored program 中的查询语句无法并行
- ~~**prepared statement 中的查询语句无法并行**~~ **(ncdb-3.1.10, cdb-20230330)**
- 串行化隔离级别事务内的查询语句无法并行
- 加锁读语句无法并行，比如 select for update/share lock
- ~~**CTE 无法并行**~~ **(ncdb-3.1.10, cdb-20230330)** Recursive CTE 无法并行

表/索引兼容性限制

- 查询表为系统表/临时表/非Innodb表无法并行
- 空间索引无法并行
- 全文索引无法并行
- 分区表无法并行
- 扫描方式为 index_merge 的表无法并行

表达式/Field兼容性限制

- 包含 Generated Column 、~~**BLOB、TEXT、JSON**~~**(ncdb-3.1.11， cdb-20230330)** 、BIT 和 GEOMETRY 字段的表无法并行
- BIT_AND, BIT_OR and BIT_XOR 类型的聚合函数无法并行
- aggregation(distinct)，如 SUM(DISTINCT)、COUNT(DISTINCT) 等聚合函数无法并行
- GIS 相关函数（如SP_WITHIN_FUNC、ST_DISTANCE 等）无法并行
- 用户自定义函数无法并行
- json相关的函数无法并行（如json_length，json_type，JSON_ARRAYAGG等）
- XML相关函数无法并行（xml_str）
- 用户锁相关的函数无法并行（is_free_lock，is_used_lock，release_lock， release_all_locks，get_lock）
- sleep 函数、random 函数、GROUP_CONCAT函数、set_user_var 函数、weight_string 函数无法并行
- ~~**部分统计相关函数（STD/STDDEV/STDDEV_POP，VARIANCE/VAR_POP/VAR_SAMP）无法并行**~~ **(ncdb-3.1.10, cdb-20230330)**

- ~~子查询无法并行~~ (ncdb-3.1.11， cdb-20230330)
- 窗口函数无法并行
- ~~rollup无法并行~~(ncdb-3.1.10， cdb-20230330)

## Hint 语法手册

并行查询 Hint 基于 MySQL Optimizer Hint (WL#3996: Add more hints) 进行扩展，支持查询级指定并行表和并行度。

> 注1: MySQL 的 SET_VAR() hint 也可以支持查询级指定系统变量。
>
> 注2: 低版本的 MySQL 命令行客户端（mysql) 会将 Hint /*+ */ 视为注释而直接丢弃，那么，对于 select /*+ parallel(n) */ MySQL 服务端收到的其实是 select ... (不含 hint 内容)，这会导致 hint 无效。建议用版本自带客户端，或者，对于系统预装客户端，带上 mysql -c 参数保留注释。可查 general log 确认此问题。

开启并行查询：

```
SELECT /*+PARALLEL(x)*/ ... FROM ...;   -- x > 0, x 表示语句并发度
```

关闭并行查询：

```
SELECT /*+PARALLEL(0)*/ ... FROM ...;
SELECT /*+NO_PARALLEL(t)*/ ... FROM ...;
```

指定并行表：

```
SELECT /*+PARALLEL(t)*/ ... FROM ...;
```

并行查询可以在hint里面同时指定并行表以及语句并发度

```
SELECT /*+PARALLEL(t 8)*/ * ... FROM ...;
```

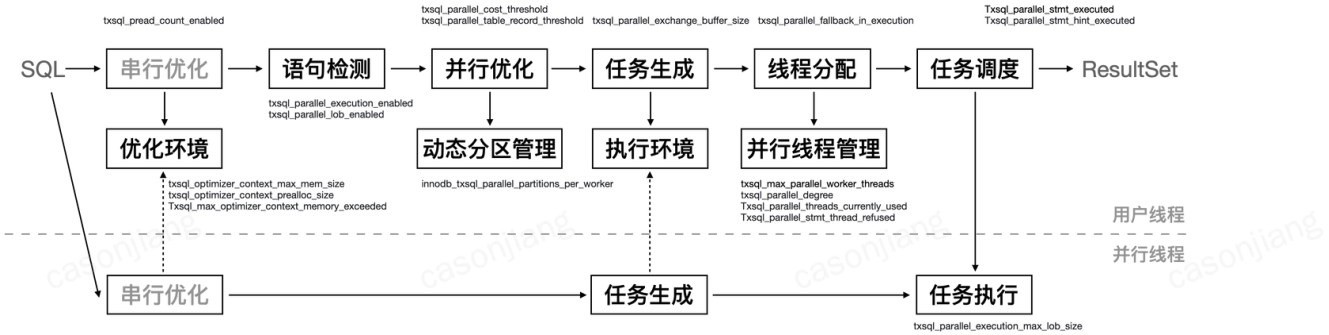
## 运维手册

### 线程列表

`SHOW PROCESSLIST` 显示所有用户线程，包含并行查询的用户线程。

`SHOW PARALLEL PROCESSLIST` 显示所有并行查询的用户线程和工作线程。

### 参数/监控作用图




### 参数表

| 参数 | 版本 | 变量类型 | 作用域 | 默认值 | 取值范围 | 含义 |
| --- | --- | --- | --- | --- | --- | --- |
| txsql_max_parallel_worker_threads | 3.1.8 | Integer | Global | 0 | 0~1024 | 实例节点可用于并行查询 |
| txsql_parallel_degree | 3.1.8 | Integer | Global Session | 4 | 0~1024 | 单条语句并行组的最大线 |
| txsql_parallel_cost_threshold | 3.1.8 | Numeric | Global Session | 50000 | 0 ~ DBL_MAX (maximum double value) | 并行执行代价阈值。只有 |
| txsql_parallel_table_record_threshold | 3.1.8 | Integer | Global Session | 5000 | 0 ~ 18446744073709551615 | 并行表行数阈值。只有行 |
| txsql_parallel_exchange_buffer_size | 3.1.8 | Integer | Global Session | 1048576 | 128 ~ 18446744073709551615 | 数据交换缓冲区大小（每 |
| txsql_optimizer_context_max_mem_size | 3.1.8 | Integer | Global Session | 8388608 | 0 – 18446744073709551615 | 单条语句可申请的并行查 |
| innodb_txsql_parallel_partitions_per_worker | 3.1.8 | Integer | Global | 13 | 0 – 4294967295 | 并行扫描切分数据时采用 |

| 参数 | 版本 | 变量类型 | 作用域 | 默认值 | 取值范围 | 含义 |
| --- | --- | --- | --- | --- | --- | --- |
| txsql_parallel_fallback_in_execution | 3.1.8 | Boolean | Global | OFF | ON OFF | 内部参数，回归测试专用， |
| txsql_parallel_execution_enabled | 3.1.8 | Boolean | Global | ON | ON OFF | 内部参数，支持云平小实 |
| txsql_parallel_execution_max_lob_size | 3.1.8 | Integer | Global Session | 65536 | 128 ～ 18446744073709551615 | 单个溢出字段的内存限制， |
| txsql_parallel_lob_enabled | 3.1.8 | Boolean | Global Session | OFF | ON OFF | 启用溢出字段并行扫描能 |
| txsql_pread_count_enabled | 3.1.8 | Boolean | Global Session | ON | ON OFF | 使用社区存储层并行读支 |
| txsql_optimizer_context_prealloc_size | 3.1.8 | Integer | Global | 4096 | 512~18446744073709551615 | 内部参数，性能测试用， |
| txsql_parallel_partition_verbose | 3.1.10 20230330 | Boolean | Global Session | OFF | ON OFF | 启用时 EXPLAIN ANALY |
| txsql_parallel_limit_enabled | 3.1.10 20230330 | Boolean | Global Session | ON | ON OFF | 启用时支持带 LIMIT 限定 |
| txsql_parallel_rollup_pushdown_threshold | 3.1.10 20230330 | Integer | Global Session | 4 | 0~100 | ROLLUP 下推算法的启用 |
| txsql_parallel_force_scan_enabled | 3.1.10 20230330 | Boolean | Global Session | OFF | ON OFF | 全表扫描或全索引扫描查 |
| txsql_parallel_force_range_enabled | 3.1.10 20230330 | Boolean | Global Session | ON | OFF | 索引范围扫描 (RANGE/R |
| txsql_parallel_subselect_cost_threshold | 3.1.11 20230330 | Numeric | Global Session | 50000 | 0 ~ DBL_MAX | 并行执行代价阈值。只有 |
| txsql_parallel_partition_threshold | 3.1.11 20230330 | Integer | Global Session | 10000 | 0~ULONG_MAX | Nested Loop Join 并行内 表。 |
| txsql_parallel_force_probe_enabled | 3.1.11 20230330 | Boolean | Global Session | OFF | ON OFF | Hash Join 的 probe 表并 |
| txsql_parallel_derived_enabled | 3.1.11 20230330 | Boolean | Global Session | ON | ON OFF | 启用时支持 derived table |

注1: 都是动态参数。

注2: 带 Session 作用域参数的都是针对单条语句的，其 Global 设置仅提供初始值。仅 Global 作用域的参数，是针对功能模块的设置。例如，txsql_max_parallel_worker_threads 这个是所有并行查询共用的工作线程数量上限。

注3: txsql_max_parallel_worker_threads 和 txsql_parallel_execution_enabled 都可以强制关闭并行。后者内部参数，云平运营专用，后者关闭时忽略前者是否开启。

注4: 并行查询模块先进行语句结构分析，按需申请 optimizer context 。确定可并行后，分配固定大小的 exchange buffer 。

## 监控表

| 全局监控项 | 版本 | 含义 |
| --- | --- | --- |
| Txsql_parallel_stmt_error | 3.1.8 | 并行查询过程中报错的语句数量 |
| Txsql_parallel_stmt_executed | 3.1.8 | 已完成执行的并行查询语句数量 |
| Txsql_parallel_stmt_hint_executed | 3.1.8 | 并行执行且带 hint 的语句数量 |
| Txsql_parallel_stmt_fallback | 3.1.8 | 可并行查询实际回退到串行执行的语句数量 |

| 全局监控项 | 版本 | 含义 |
| --- | --- | --- |
| Txsql_parallel_threads_currently_used | 3.1.8 | 并行执行功能当前使用的线程总数 |
| Txsql_parallel_stmt_thread_refused | 3.1.8 | 因线程超限而无法并行的语句数量 |
| Txsql_max_optimizer_context_memory_exceeded | 3.1.8 | 因内存限制而无法使用优化环境的语句数量 |
| Txsql_parallel_request_not_chosen_small_cost | 3.1.10 20230330 | 因代价小于阈值没有并行 |
| Txsql_parallel_request_not_chosen_hint_invalid_dop | 3.1.10 20230330 | 因 hint 给定 DOP 大于最大并行线程数没有并行 |
| Txsql_parallel_request_not_chosen_invalid_dop | 3.1.10 20230330 | 因设置 DOP 大于最大并行线程数没有并行 |
| Txsql_parallel_request_not_chosen_tx_isolation | 3.1.10 20230330 | 因隔离级别为 Serializable 没有并行 |
| Txsql_parallel_request_not_chosen_attachable_transaction | 3.1.10 20230330 | 因存在 attachable 事务没有并行 |
| Txsql_parallel_request_not_chosen_locking_clause | 3.1.10 20230330 | 因语句存在 locking 没有并行 |
| Txsql_parallel_request_not_chosen_not_select_stmt | 3.1.10 20230330 | 因语句不是 SELECT 语句没有并行 |
| Txsql_parallel_request_not_chosen_prepared_statement | 3.1.10 20230330 | 因语句是 prepared statement 没有并行 |
| Txsql_parallel_request_not_chosen_stored_program | 3.1.10 20230330 | 因语句是存储过程没有并行 |
| Txsql_parallel_request_not_chosen_match_func | 3.1.10 20230330 | 因语句使用 MATCH() AGAINST 语法没有并行 |
| Txsql_parallel_request_not_chosen_udf | 3.1.10 20230330 | 因语句使用 UDF 没有并行 |
| Txsql_parallel_request_not_chosen_ull | 3.1.10 20230330 | 因连接使用 user-level lock 没有并行 |
| Txsql_parallel_request_not_chosen_hint_no_parallel | 3.1.10 20230330 | 因 hint 禁用并行表没有并行 |
| Txsql_parallel_request_not_chosen_tmp_table | 3.1.10 20230330 | 因并行表是临时表没有并行 |
| Txsql_parallel_request_not_chosen_not_innodb_table | 3.1.10 20230330 | 因并行表非 innodb 表没有并行 |
| Txsql_parallel_request_not_chosen_partition_table | 3.1.10 20230330 | 因并行表是分区表没有并行 |
| Txsql_parallel_request_not_chosen_user_table | 3.1.10 20230330 | 因并行表是用户表没有并行 |
| Txsql_parallel_request_not_chosen_locking_read | 3.1.10 20230330 | 因并行表上锁没有并行 |
| Txsql_parallel_request_not_chosen_full_text_index | 3.1.10 20230330 | 因并行表使用全文索引没有并行 |
| Txsql_parallel_request_not_chosen_view_or_derived | 3.1.10 20230330 | 因并行表是视图表或 derived 表没有并行 |
| Txsql_parallel_request_not_chosen_small_table | 3.1.10 20230330 | 因并行表行数\分区数小于阈值没有并行 |
| Txsql_parallel_request_not_chosen_unsupported_scan_access | 3.1.10 20230330 | 因并行表扫描方式不支持没有并行 |
| Txsql_parallel_request_not_chosen_unsupported_range_access | 3.1.10 20230330 | 因并行表 Range 扫描方式不支持没有并行 |
| Txsql_parallel_request_not_chosen_generated_column | 3.1.10 20230330 | 因并行表含有生成列没有并行 |
| Txsql_parallel_request_not_chosen_geometry_column | 3.1.10 20230330 | 因并行表含有 Geometry 字段没有并行 |
| Txsql_parallel_request_not_chosen_bit_column | 3.1.10 20230330 | 因并行表含有 Bit 字段没有并行 |
| Txsql_parallel_request_not_chosen_lob_column | 3.1.10 20230330 | 因并行表含有 Blob 字段没有并行 |
| Txsql_parallel_request_not_chosen_ref_or_null | 3.1.10 20230330 | 因并行表扫描方式为 ref_or_null 没有并行 |
| Txsql_parallel_request_not_chosen_eq_ref | 3.1.10 20230330 | 因并行表扫描方式为 eq_ref 没有并行 |
| Txsql_parallel_request_not_chosen_const_table | 3.1.10 20230330 | 因并行表为 Const 表没有并行 |
| Txsql_parallel_request_not_chosen_stream_limit | 3.1.10 20230330 | 因执行计划为流式计划且附带 limit 没有并行 |