TDSQL Boundless

故障处理

腾讯云

文档目录

故障处理
性能相关
CPU 利用率过高

最近更新时间：2026-07-10 10:54:30

现象描述

当 TDSQL Boundless 实例 CPU 利用率过高时，通常会出现以下现象：

- 实例监控中 **CPU 利用率**或 **CPU 最大使用率**持续处于高位，或在特定时段出现明显尖峰。
- **慢查询数**增加，部分 SQL 执行耗时明显上升。
- **SQL 平均执行时间**、**P95 SQL 执行时间**、**P99 SQL 执行时间**升高。
- **当前打开连接数**、**汇总活跃线程数**或节点维度**活跃线程数**增多。
- 在多节点实例中，可能仅个别节点 CPU 明显高于其他节点。
- 严重时，业务访问可能出现响应抖动、请求堆积或超时。

若 CPU 利用率长期过高，可能影响实例性能和业务稳定性，建议及时排查。

可能原因

TDSQL Boundless 实例 CPU 利用率过高，常见原因如下：

慢 SQL 或低效 SQL

慢 SQL 是 CPU 利用率升高的常见原因之一，通常表现为查询未命中合适索引，或 SQL 存在大范围扫描、排序、聚合、回表等高开销操作。在这类场景下，通常会同时看到**慢查询数**、**SQL 平均执行时间**、**P95 SQL 执行时间**、**P99 SQL 执行时间**上升。

业务请求量突增

当业务高峰流量突增，或批量任务、抽数任务、迁移任务在短时间内集中执行时，实例 CPU 可能快速升高。此时常伴随**总 QPS**、**读 QPS**、**写 QPS**、**每秒处理的事务数**抬升。

热点访问或负载分布不均

TDSQL Boundless 为分布式数据库。如果请求长期集中在少量热点数据、热点分片或少数节点上，可能导致个别节点负载明显偏高，出现单节点 CPU 打满的情况。此时通常会看到节点维度 **CPU 利用率**、**活跃线程数**、**总 QPS**、**数据盘 IO 时耗**等指标明显高于其他节点。

元数据访问频繁

频繁执行 `SHOW TABLES`、`SHOW TABLES LIKE` 或大量查询表结构、系统对象信息等元数据操作时，也可能带来较高 CPU 开销。尤其是在单个数据库表数量较多时，影响会更加明显。

内存或 I/O 压力放大 CPU 问题

在部分场景下，CPU 升高并不是首要原因，而是由资源瓶颈间接引起。例如**内存利用率**偏高、**读 IOPS** 或**写 IOPS** 持续高位、**数据盘 IO 最大时耗**升高等，都会导致 SQL 执行变慢，进而放大 CPU 压力。

处理步骤

步骤1： 查看实例监控

登录 TDSQL Boundless 控制台，进入 TDSQL Boundless 实例监控页面，优先确认 CPU 异常的范围和持续时间。
建议重点关注以下几类监控指标：

类型	指标
资源类指标	用于判断是否存在资源瓶颈或节点负载异常，建议重点关注： • CPU 利用率 • CPU 最大使用率 • 内存利用率 • 内存最大使用率 • 数据盘最大使用率 • 磁盘利用率 • 读 IOPS • 写 IOPS • 总 IOPS • 数据盘 IO 最大时耗
连接类指标	用于判断是否存在连接堆积或线程压力，建议重点关注： • 当前打开连接数 • 最大连接数 • 汇总活跃线程数 • 连接数利用率
访问类指标	用于判断是否存在流量突增、慢 SQL 增多或 SQL 执行效率下降，建议重点关注： • 慢查询数 • 总 QPS • 读 QPS • 写 QPS • 每秒处理的事务数

	• SQL 失败数 • SQL 平均执行时间 • P95 SQL 执行时间 • P99 SQL 执行时间

如果实例为多节点部署，建议同时查看节点维度监控，确认是否仅有个别节点异常。若仅单节点 CPU 明显偏高，通常需要重点排查热点访问或负载倾斜问题。

步骤2：通过 DBbrain 查看异常诊断

TDSQL Boundless 支持通过 DBbrain 进行异常诊断和慢 SQL 分析。

1. 登录 DBbrain 控制台 。
2. 在左侧导航中选择**诊断优化**。
3. 在页面顶部选择数据库类型为 **TDSQL Boundless**，并选择目标实例。
4. 单击**异常诊断**页签，查看近3小时内触发的诊断事件。
5. 单击具体事件，查看**事件详情**、**现场描述**、**智能分析**和**优化建议**。

   如果实例存在慢 SQL，建议同时查看**慢 SQL 分析**页面，重点确认异常时间段内是否存在高耗时、高频或扫描行数偏大的 SQL。详细操作请参见 异常诊断 。

步骤3：排查慢 SQL 和低效 SQL

若 CPU 利用率升高伴随**慢查询数**增多或 **SQL 平均执行时间**上升，建议优先排查 SQL 执行效率。
可执行以下语句查看当前会话：

```
SHOW FULL PROCESSLIST;
```

对于可疑 SQL，建议查看执行计划：

```
EXPLAIN SELECT ...;
```

重点关注以下情况：

- 查询是否命中合适索引
- 是否存在全表扫描
- 是否存在大量回表
- 是否存在高成本排序、聚合、临时表操作
- 异常 SQL 的执行时间是否与监控中的异常时间段一致

如果确认是索引设计不合理或 SQL 写法低效导致，建议根据实际查询条件进行索引和 SQL 优化。

步骤4：排查热点访问

若实例为多节点部署，且仅个别节点 CPU 明显偏高，建议进一步排查是否存在热点访问问题。
建议重点确认以下信息：

- 某类 SQL 是否长期访问同一业务 key
- 某个节点是否在高峰时段持续 CPU 偏高
- 异常节点的**总 QPS**、**活跃线程数**、**读 IOPS**、**写 IOPS**、**数据盘 IO 时耗**是否同步升高
- 异常节点的连接和线程指标是否明显高于其他节点

若确认存在热点访问，建议从以下方向处理：

- 优化相关 SQL，降低单次查询开销
- 优化业务模型，避免请求长期集中在单一热点数据
- 通过扩容分散负载

步骤5：排查业务流量和后台任务

若 CPU 异常与业务高峰、定时任务、批处理、迁移或巡检时间一致，建议进一步排查是否存在流量突增或任务集中执行的情况。
建议重点关注：

- **总 QPS**
- **读 QPS**
- **写 QPS**
- **每秒处理的事务数**
- **SQL 失败数**

如果上述指标在异常时间段明显升高，则可能是业务请求量突增、任务集中执行或应用重试放大导致。建议通过错峰执行任务、控制并发度、应用侧限流等方式缓解。

步骤6：检查是否存在内存或 I/O 压力

若 CPU 升高的同时，还伴随以下现象，则说明问题可能与资源瓶颈有关：

- **内存利用率** 持续偏高
- **读 IOPS**、**写 IOPS**、**总 IOPS** 持续高位
- **数据盘 IO 最大时耗** 或节点维度**数据盘 IO 时耗**明显升高
- **SQL 平均执行时间**、**P95 SQL 执行时间**、**P99 SQL 执行时间**整体变长
- 节点维度出现**是否发生 OOM**

此时建议综合评估当前实例规格是否满足业务负载需求，并结合实际情况考虑升配或扩容。

步骤7：必要时进行实例扩容

如果经过排查确认 CPU 高是由当前规格不足导致，建议结合实际业务情况进行扩容。

- 若实例整体 CPU 长期处于高位，可考虑升配。详细请参见 调整实例配置 。
- 若仅个别节点负载明显偏高，可优先考虑扩容以分散热点负载。

- 若同时存在内存和 I/O 压力，建议综合评估后统一扩容。

说明：

- CPU 利用率升高不一定完全由 CPU 本身引起，建议结合**慢查询数**、**总 QPS**、**SQL 平均执行时间**、**P95 SQL 执行时间**、**P99 SQL 执行时间**、**读 IOPS**、**写 IOPS**、**内存利用率**等指标综合判断。
- 对于多节点实例，建议同时查看实例维度和节点维度监控，避免遗漏单节点热点问题。
- 在未确认原因前，不建议直接进行高风险参数调整，应优先完成 SQL、流量和资源侧排查。
- 如果业务中存在大量元数据访问，建议尽量降低调用频率，并避免在业务高峰期执行。
- 若节点维度出现**是否发生 OOM**，建议优先排查内存压力、异常 SQL 和缓存命中情况。
- 若排查后仍无法定位原因，建议进一步收集异常时间段的监控数据、慢 SQL、执行计划及业务变更信息后再做深入分析。

内存使用率过高

最近更新时间：2026-07-10 10:54:30

现象描述

TDSQL Boundless 实例在运行过程中，内存使用率持续处于高位或出现突增，可能伴随以下现象：

- 实例监控面板中内存使用率持续超过80%。
- 查询响应时间变长，出现间歇性卡顿。
- 新连接建立失败或被拒绝。
- 节点因 OOM（Out Of Memory）触发重启，可能导致主备切换。

您可在 TDSQL Boundless 控制台的**实例详情 > 监控告警 > 指标监控**页面中查看各节点的内存使用率指标。

故障风险

内存使用率过高可能导致以下风险：

- 节点触发 OOM 后自动重启，正在执行的事务中断。
- 存储节点（DN）OOM 可能触发主备切换，切换期间业务出现秒级闪断。
- 计算节点（CN）OOM 会导致该节点上的所有会话断开，需重新连接。
- 业务高峰期发生 OOM 或主备切换，会严重影响业务稳定性和连续性。

可能原因

TDSQL Boundless 的内存消耗主要来源于以下几类，OOM 可能由任一类内存的异常增长引发。

会话级私有内存

每个数据库连接在执行 SQL 时会独立分配私有内存缓冲区，并发连接数越多、单条 SQL 消耗的缓冲区越大，私有内存总量就越高。

- **低效 SQL 或全表扫描**：缺少索引或执行计划不佳的 SQL 会触发大量排序和临时表操作，导致单个会话的私有内存缓冲区被反复分配，内存占用急剧上升。
- **连接数过多**：并发连接数远超业务实际需求，每个连接的私有内存累加后总量过大。可通过参数 max_connections 控制最大连接数。
- **表缓存数量过大**：table_open_cache 和 table_definition_cache 设置过大，缓存的表对象和表定义过多，占用大量内存。

全局共享内存

全局共享内存在实例启动时预分配，所有连接共享访问。

- **Block Cache 不足**：Block Cache 大小不足以缓存热点数据，大量读请求穿透到磁盘，导致 I/O 压力增大。Block Cache 大小由实例内存规格决定，不支持单独调整。

- **Write Buffer 积压**：存储引擎的 MemTable 在写入高峰期积压大量内存，通常与大批量写入操作有关。

事务内存

大事务和批量导入会在事务执行期间持有大量内存。

- **大事务**：单个事务写入数据量过大，占用过多事务内存。可通过参数 tdstore_max_txn_size 限制单个事务的内存使用上限。
- **批量导入**：`LOAD DATA`、`INSERT ... SELECT` 等批量操作在短时间内产生大量写入，事务内存集中占用。可通过参数 tdstore_bulk_load_total_merge_buffer_size 控制批量导入的合并缓冲区大小。

实例规格不匹配

- **内存规格不足**：实例配置的内存规格无法满足业务实际负载需求。
- **内存与 CPU 比例失衡**：CPU 扩容后内存未同步调整，导致内存成为瓶颈。

解决思路

TDSQL Boundless 的内存管理以实例内存规格为核心，实例启动时根据内存规格自动计算并联动设置最大连接数、Block Cache 大小、表缓存内存上限等参数。处理内存使用率过高问题的整体思路如下：

1. **查看监控**：通过控制台监控确认内存使用趋势和异常时间点。
2. **定位内存消耗来源**：通过系统视图查看各内存组件的占用分布，判断是共享内存、私有内存还是事务内存的问题。
3. **优化 SQL 和连接**：优化低效 SQL、减少无效长连接，降低会话级私有内存消耗。
4. **调整参数**：通过控制台参数设置页面调整用户可见的内存相关参数。
5. **扩容升级**：优化后仍无法满足需求时，升级实例内存规格。

处理步骤

步骤1：查看内存监控

1. 登录 TDSQL Boundless 控制台，在**实例列表**中单击目标**实例 ID**，进入**实例详情**页。
2. 选择**监控告警 > 指标监控**页签，查看各节点的内存使用率曲线。
3. 重点关注以下信息：
   - 内存使用率是否持续超过80%。
   - 是否存在突增（短时间内急剧上升）。
   - 突增时间点是否与业务高峰、批量任务执行时间吻合。
4. 如需更细粒度的内存分布信息，执行步骤2。

步骤2：查看内存使用详情

通过系统视图查看各内存组件的占用情况，判断内存消耗的主要来源。

1. 连接数据库实例，执行以下 SQL 查看 LogService 内存统计信息。

```
SELECT * FROM information_schema.LOGSERVICE_MEMORY_STAT;
```

该视图展示各内存组件的当前使用量和总量，帮助您快速定位内存消耗大户。

2. 执行以下 SQL 查看 `performance_schema` 中的内存汇总信息（适用于已开启 `performance_schema` 的实例）。

```
SELECT
    EVENT_NAME,
    CURRENT_NUMBER_OF_BYTES_USED / 1024 / 1024 AS CURRENT_MB,
    HIGH_NUMBER_OF_BYTES_USED / 1024 / 1024 AS HIGH_MB
FROM performance_schema.memory_summary_global_by_event_name
WHERE CURRENT_NUMBER_OF_BYTES_USED > 0
ORDER BY CURRENT_NUMBER_OF_BYTES_USED DESC
LIMIT 20;
```

该查询列出当前内存占用最高的20个内存事件，帮助您定位具体的内存消耗来源。

步骤3：优化慢 SQL

慢 SQL 是导致会话级私有内存异常增长的常见原因。通过 DBbrain 智能诊断或手动排查，优化低效 SQL。

1. 登录 DBbrain 控制台，在左侧导航选择**诊断优化**，选择**慢 SQL 分析**页签。选择查看是否存在慢 SQL 诊断建议。详细请参见 慢 SQL 分析 。
2. 连接数据库实例，执行以下 SQL 查看当前正在执行的高消耗 SQL。

```
SELECT
    ID,
    USER,
    HOST,
    DB,
    COMMAND,
    TIME,
    STATE,
    INFO
FROM information_schema.PROCESSLIST
WHERE COMMAND != 'Sleep'
ORDER BY TIME DESC
LIMIT 20;
```

重点关注 `TIME` 值较大且 `STATE` 包含 `Sorting result`、`Creating sort index`、`Sending data` 等状态的会话，这些会话通常正在消耗大量排序或连接缓冲区。

3. 对识别出的慢 SQL 进行优化：
   - 添加合适的索引，避免全表扫描。
   - 避免在 SQL 中对大结果集进行排序或分组。
   - 拆分复杂大查询为多个小查询。

步骤4：减少无效连接

连接数过多会导致会话级私有内存总量过大。

1. 登录 TDSQL Boundless 控制台，在**实例列表**中单击目标**实例 ID**，进入**实例详情**页。
2. 选择**参数设置** > **数据库参数**页签，查看 max_connections（最大连接数）的当前值。
3. 选择**监控告警** > **指标监控**页签，查看当前连接数指标，确认实际并发连接数是否接近 `max_connections` 的上限。
4. 连接数据库实例，执行以下 SQL 查看各来源的连接分布，定位连接数过多的来源。

```
SELECT
    SUBSTRING_INDEX(HOST, ':', 1) AS CLIENT_HOST,
    COUNT(*) AS CONNECTION_COUNT
FROM information_schema.PROCESSLIST
GROUP BY CLIENT_HOST
ORDER BY CONNECTION_COUNT DESC;
```

5. 根据查询结果，采取以下措施：
   - 降低应用程序侧连接池的最大连接数配置，使其与 `max_connections` 合理匹配。
   - 排查是否存在连接泄漏（连接未正确关闭）。
   - 对非核心业务进行限流或错峰执行。
6. 如需调整最大连接数（`max_connections`），详细请参见 设置实例参数。

步骤5：查看 Block Cache 命中数

Block Cache 是 TDStore 存储引擎的核心缓存，命中数持续偏低说明当前 Block Cache 大小不足，大量读请求穿透到磁盘导致 I/O 压力增大，建议评估是否需要扩容内存规格以增大 Block Cache。

1. 登录 TDSQL Boundless 控制台，在**实例列表**中单击目标**实例 ID**，进入**实例详情**页。
2. 选择**监控告警** > **指标监控**页签，切换到节点维度，查找 **DataDB BlockCache 命中数**指标曲线。
3. 关注命中数趋势：
   - 命中数持续偏低，说明 Block Cache 缓存效果不佳，大量读请求穿透到磁盘。
   - 可结合读 IOPS、读 I/O 吞吐量等指标综合判断磁盘读压力。
4. 若命中数持续偏低，可考虑以下措施：

- 优化查询模式，减少大范围扫描。
- Block Cache 大小由实例内存规格决定，不支持单独调整，如需增大 Block Cache 请参见 升级实例内存规格 。

步骤6：查看事务内存参数

大事务和批量导入会在事务执行期间占用大量内存。

1. 登录 TDSQL Boundless 控制台 ，在**实例列表**中单击目标**实例 ID**，进入**实例详情**页。
2. 选择**参数设置** > **数据库参数**页签，查看以下事务内存相关参数的当前值：

参数名	说明
tdstore_max_txn_size	单个事务的内存使用上限，默认1GB。
tdstore_bulk_load_total_merge_buffer_size	批量导入合并缓冲区总大小。

3. 若发现内存使用率在批量任务执行期间突增，可适当调低相关参数限制。

> **注意：**
> 调低事务内存上限可能导致大批量写入操作被限制或报错，请根据业务实际情况谨慎调整，并在低峰期验证。

步骤7：通过控制台调整参数

根据排查结果，调整以下用户可见的内存相关参数，详细请参见 设置实例参数 。

参数名	说明	调整建议
max_connections	最大连接数。连接数过多会导致会话级私有内存总量过大。	根据业务实际并发需求设置，避免设置过高。
table_open_cache	表打开缓存数量。设置过大会占用过多内存。	根据实例表数量合理设置，不宜过大。
table_definition_cache	表定义缓存数量。设置过大会占用过多内存。	根据实例表数量合理设置，不宜过大。
tdstore_max_txn_size	单个事务的内存使用上限。	若大事务导致内存突增，可适当调小。
tdstore_bulk_load_total_merge_buffer_size	批量导入合并缓冲区总大小。	若批量导入导致内存突增，可适当调小。

步骤8：升级实例内存规格

若经过以上优化后，内存使用率仍持续超过80%且影响业务正常运行，建议升级实例内存规格。升级后实例内存规格会自动更新，系统将根据新的内存规格自动联动调整最大连接数、Block Cache 大小、表缓存内存上限等参数。详细操作请参见 调整实例配置 。

预防措施

为避免内存使用率过高问题反复发生，建议采取以下预防措施：

- **配置告警策略**：在控制台为实例配置内存使用率告警，建议阈值设为80%，提前发现内存增长趋势。
- **定期巡检慢 SQL**：通过 DBbrain 智能诊断定期检查慢 SQL，及时优化低效查询。
- **合理设置连接池**：应用程序侧连接池大小应与实例 `max_connections` 合理匹配，避免连接数过多。
- **控制批量任务规模**：大批量写入操作建议分批执行，单批次数据量控制在合理范围内。
- **监控 Block Cache 命中数**：定期关注控制台监控面板中的 DataDB BlockCache 命中数指标，命中数持续偏低时考虑扩容。

慢查询数过高

最近更新时间：2026-07-10 10:54:30

现象描述

TDSQL Boundless 实例在运行过程中，慢查询数持续偏高或出现突增，可能伴随以下现象：

- 实例监控面板中慢查询数指标持续升高。
- CPU 利用率同步飙升，查询响应时间变长。
- 业务侧出现请求超时或响应延迟。
- SQL 失败数增加，部分查询无法正常执行。

您可在 TDSQL Boundless 控制台的**实例详情 > 监控告警 > 指标监控**页面查看慢查询数、CPU 利用率、SQL 执行时间等指标。

可能原因

慢查询数过高通常由 SQL 执行效率低或实例负载超出承载能力导致。可能原因如下：

序号	可能原因
1	SQL 语句未使用索引或未使用较佳的索引，导致全表扫描或低效执行计划。
2	QPS 压力超过当前实例规格的承载上限，大量请求排队堆积。
3	并行度设置不当，并行查询资源争抢导致单条 SQL 执行时间过长。
4	数据量增长后统计信息过期，优化器选择了次优执行计划。

解决思路

针对不同原因，解决思路如下：

可能原因	解决思路
SQL 未使用索引或执行计划不佳	通过 DBbrain 智能诊断分析慢 SQL，添加合适的索引，优化 SQL 语句。
QPS 超出实例承载能力	优化业务请求模式，降低无效查询；必要时升级实例规格。
并行度设置不当	通过控制台调整 `max_parallel_degree` 参数，合理控制并行查询并发度。
统计信息过期	对相关表执行 `ANALYZE TABLE` 更新统计信息，使优化器选择更优执行计划。

处理步骤

步骤1：查看慢查询监控

1. 登录 TDSQL Boundless 控制台，在**实例列表**中单击目标**实例 ID**，进入**实例详情**页。
2. 选择**监控告警 > 指标监控**页签，查看以下指标的变化趋势：

监控指标	说明	关注点
慢查询数	每秒慢查询数量	是否持续升高或突增
CPU 利用率	实例 CPU 使用率	是否与慢查询数同步飙升
SQL 平均执行时间	SQL 平均耗时	是否明显变长
P95 SQL 执行时间	95% 分位执行时间	长尾查询耗时情况
P99 SQL 执行时间	99% 分位执行时间	极端慢查询耗时情况
总 QPS	每秒 SQL 总数	是否超出实例承载能力
SQL 失败数	每秒失败 SQL 数	是否出现大量失败

- 重点关注慢查询数突增的时间点，与业务变更（如发版、大促）时间是否吻合。
- 查看 SQL 耗时分布指标，判断慢查询主要集中在哪个耗时区间，为后续优化提供参考。

步骤2：通过 DBbrain 分析慢 SQL

登录 DBbrain 控制台，在左侧导航选择**诊断优化**，选择**慢 SQL 分析**页签。选择查看是否存在慢 SQL 诊断建议。详细请参见 慢 SQL 分析 。

步骤3：通过系统视图分析慢 SQL

如需更细粒度的 SQL 性能分析，可通过系统视图查询 SQL 执行统计信息。

1. 连接数据库实例，执行以下 SQL 查看 SQL 摘要统计信息。

```
SELECT
    SCHEMA_NAME,
    DIGEST_TEXT,
    COUNT_STAR AS EXEC_COUNT,
    ROUND(AVG_TIMER_WAIT / 1000000000, 2) AS AVG_MS,
    ROUND(MAX_TIMER_WAIT / 1000000000, 2) AS MAX_MS,
    SUM_ROWS_EXAMINED AS TOTAL_ROWS_EXAMINED,
    SUM_ROWS_SENT AS TOTAL_ROWS_SENT
FROM performance_schema.events_statements_summary_by_digest
```

```
WHERE SCHEMA_NAME IS NOT NULL
ORDER BY AVG_TIMER_WAIT DESC
LIMIT 20;
```

该查询列出平均执行时间最长的20条 SQL 模板，重点关注以下信息：

- `AVG_MS`：平均执行时间（毫秒），数值越大说明 SQL 越慢。
- `TOTAL_ROWS_EXAMINED`：总扫描行数，若扫描行数远大于返回行数，说明存在低效扫描。
- `DIGEST_TEXT`：规范化后的 SQL 文本，用于定位具体 SQL。

2. 对识别出的慢 SQL，执行 `EXPLAIN` 查看执行计划。

```
EXPLAIN SELECT ...;
```

重点关注以下信息：

- `type` 列为 `ALL` 表示全表扫描，需要添加索引。
- `rows` 列值过大说明扫描行数过多。
- `key` 列为 `NULL` 表示未使用索引。

步骤4：优化 SQL 语句

根据 步骤2 和 步骤3 的分析结果，对慢 SQL 进行优化。

1. **添加索引**：对 `WHERE`、`JOIN`、`ORDER BY`、`GROUP BY` 涉及的列添加合适的索引。
2. **改写 SQL**：
   - 避免 `SELECT *`，只查询需要的列。
   - 避免在 `WHERE` 条件中对索引列使用函数或运算。
   - 将大查询拆分为多个小查询，利用分页减少单次返回数据量。
   - 避免在 `OR` 条件中混合使用索引列和非索引列。
3. **更新统计信息**：若数据量增长后慢查询增多，执行以下 SQL 更新统计信息。

```
ANALYZE TABLE table_name;
```

更新统计信息后，优化器可以基于最新的数据分布选择更优的执行计划。

步骤5：调整并行度参数

若慢查询与 HTAP 并行查询有关，可通过控制台调整并行度参数，详细请参见 设置实例参数 。

- 若并行查询资源争抢导致整体性能下降，适当调小 `max_parallel_degree`。
- 若单条大查询执行过慢且资源充足，适当调大 `max_parallel_degree` 以提升并行度。

步骤6：调整慢查询阈值

若需要调整慢查询的判定标准，可通过控制台修改 `long_query_time` 参数，详细操作请参见 设置实例参数 。

- 调小 `long_query_time` 可以捕获更多慢查询，便于全面排查问题，但可能增加日志量。
- 调大 `long_query_time` 可以只关注极慢查询，减少干扰。

步骤7：设置 SQL 执行超时

若需要限制单条 SQL 的最大执行时间，避免慢查询长时间占用资源，可通过控制台修改 `max_execution_time` 参数。例如设置为 `5000` ，表示 SELECT 语句执行超过5秒将自动终止。详细操作请参见 设置实例参数 。

> **警告：**
> `max_execution_time` 仅对 `SELECT` 语句生效。设置过小的超时时间可能导致正常的长查询被意外终止，请根据业务实际最长查询时间合理设置。

步骤8：升级实例规格

若经过以上优化后，慢查询数仍持续偏高且 QPS 已超出当前实例规格的承载能力，建议升级实例规格，详细操作请参见 调整实例配置 。

预防措施

为避免慢查询数过高问题反复发生，建议采取以下预防措施：

- **配置告警策略**：在控制台为慢查询数和 CPU 利用率配置告警，建议慢查询数阈值根据业务基线设置，CPU 利用率阈值设为80%。
- **定期巡检慢 SQL**：通过 DBbrain 智能诊断定期检查慢 SQL，及时优化低效查询。
- **定期更新统计信息**：对数据变化频繁的表定期执行 `ANALYZE TABLE` ，确保优化器选择最优执行计划。
- **规范 SQL 开发**：新上线 SQL 需经过 `EXPLAIN` 验证执行计划，确保使用索引且扫描行数合理。
- **监控 QPS 趋势**：关注 QPS 增长趋势，提前规划实例扩容。

事务相关
事务过大

最近更新时间：2026-07-10 10:54:30

现象描述

TDSQL Boundless 实例在运行过程中，业务侧出现事务过大相关报错，可能伴随以下现象：

- 应用程序日志中出现事务写入大小超限的错误提示。
- 应用程序日志中出现事务参与者数量超限的错误提示。
- 实例监控中 SQL 失败数指标升高。
- 大事务执行期间，内存利用率升高，其他业务请求响应变慢。
- 大事务执行期间，可能触发锁等待或死锁。

您可在 TDSQL Boundless 控制台的**实例详情 > 监控告警 > 指标监控**页面查看 SQL 失败数、内存利用率、每秒处理的事务数等指标。

可能原因

事务过大通常由单事务写入数据量过多、参与者数量超限或批量操作未分批导致。可能原因如下：

序号	可能原因
1	单个事务写入数据量过大，超过 `tdstore_max_txn_size` 限制，导致事务被拒绝写入。
2	单个事务涉及的参与者（复制组）数量超过 `tdsql_transaction_max_participants` 限制。
3	批量导入操作（如 `LOAD DATA`、`INSERT ... SELECT`）未分批执行，单次操作数据量过大。
4	长事务执行时间过久，持有锁和内存资源时间过长，影响其他业务。
5	`ALTER TABLE` 等大表 DDL 操作产生大量数据拷贝，事务过大。

解决思路

针对不同原因，解决思路如下：

可能原因	解决思路
单事务写入量过大	拆分大事务为多个小事务，通过控制台调整 `tdstore_max_txn_size` 参数。

参与者数量超限	减少单事务涉及的表和分区数量，分批执行跨表操作。
批量导入未分批	将大批量导入拆分为多批次，每批次控制在合理数据量内。
长事务持锁过久	优化事务内 SQL 执行效率，缩短事务范围，设置合理的超时时间。
DDL 数据拷贝过大	确认 `tdsql_split_trans_during_alter_copy` 参数已开启，使 DDL 自动拆分事务。

处理步骤

步骤1：查看监控指标

1. 登录 TDSQL Boundless 控制台，在**实例列表**中单击目标**实例 ID**，进入**实例详情**页。
2. 选择**监控告警 > 指标监控**页签，查看以下指标的变化趋势：

监控指标	说明	关注点
SQL 失败数	每秒失败的 SQL 数量	事务超限时失败数会突增
内存利用率	实例内存使用率	大事务执行期间内存是否飙升
每秒处理的事务数	每秒事务提交数	事务数是否异常下降
慢查询数	每秒慢查询数量	大事务是否导致其他查询变慢
活跃线程数	当前活跃线程数	是否有线程长时间被大事务阻塞

重点关注 SQL 失败数突增和内存利用率飙升的时间点，与批量任务或大事务执行时间是否吻合。

步骤2：查看大事务信息

通过系统视图查看当前是否存在大事务，以及大事务的资源占用情况。

1. 连接数据库实例，执行以下 SQL 查看当前的大事务信息。

```
SELECT
    replication_group_id,
    transaction_id,
    disk_usage_bytes,
    memory_usage_bytes,
    rollback,
    destroyed
FROM information_schema.TDSTORE_LARGE_TXN
WHERE destroyed = 0
```

```
ORDER BY disk_usage_bytes DESC
LIMIT 20;
```

该视图展示当前活跃的大事务信息，字段说明如下：

- `replication_group_id`：复制组 ID。
- `transaction_id`：事务 ID。
- `disk_usage_bytes`：磁盘使用量（字节），反映事务数据在 SST 文件中的大小。
- `memory_usage_bytes`：内存使用量（字节），反映事务在 MemTable 中的大小。
- `rollback`：是否已回滚。
- `destroyed`：是否已销毁。

2. 若存在 `disk_usage_bytes` 或 `memory_usage_bytes` 值较大的事务，记录其 `transaction_id`，用于后续排查。
3. 执行以下 SQL 查看当前正在执行的会话，定位大事务对应的 SQL。

```
SELECT
    ID,
    USER,
    HOST,
    DB,
    COMMAND,
    TIME,
    STATE,
    INFO
FROM information_schema.PROCESSLIST
WHERE COMMAND != 'Sleep'
ORDER BY TIME DESC
LIMIT 20;
```

重点关注 `TIME` 值较大的会话，这些会话可能正在执行长事务或大事务。

步骤3：终止大事务（可选）

若大事务已严重影响业务正常运行，可终止对应会话以释放资源。

1. 根据步骤2查到的大事务信息，结合步骤2第3步 `PROCESSLIST` 查询结果中 TIME 值较大的会话，通过执行时间、SQL 文本等信息综合定位大事务对应的会话 ID。
2. 执行以下 SQL 终止该会话。

```
KILL <会话ID>;
```

> **警告：**
> `KILL` 操作会中断正在执行的事务并触发回滚，回滚过程中可能继续占用资源。请确认目标会话后再执行，避免误杀正常业务会话。

步骤4：调整单事务内存上限

若业务确实需要执行较大事务，可通过控制台调整 `tdstore_max_txn_size` 参数，详细操作请参见 设置实例参数 。

- 默认值为1GB。若单事务数据量超过此限制，可适当调大。
- 调大后需关注实例内存利用率，避免多个大事务并发执行导致内存不足。

> **注意：**
> - 调大 `tdstore_max_txn_size` 会增加单事务可占用的内存，在高并发场景下可能导致内存压力增大。
> - 建议优先通过拆分事务解决，仅在业务确实需要大事务时才调整此参数。

步骤5：优化批量导入操作

批量导入是导致事务过大的常见原因，建议从业务层面优化批量操作。

1. **分批导入**：将大批量数据拆分为多个批次，每批次使用独立事务提交。
   - 例如，将 `INSERT INTO ... SELECT ...` 拆分为多个带 `LIMIT` 的子查询。
   - 每批次的数据量建议控制在 `tdstore_max_txn_size` 限制以内。
2. **调整批量导入参数**：若使用 `LOAD DATA` 等批量导入方式，可通过控制台调整以下参数，详细操作请参见 设置实例参数 。
   - `tdstore_bulk_load_total_merge_buffer_size`：批量导入合并缓冲区总大小，控制批量导入过程中的内存使用。
   - `tdstore_bulk_load_merge_chunk_size`：批量导入单个合并块大小，控制单次归并排序的数据量。

步骤6：优化长事务

长事务会长时间持有锁和内存资源，增加事务过大的风险。

1. **缩短事务范围**：
   - 事务中只包含必要的数据库操作，将非数据库操作（如远程调用、文件 IO）移到事务外部。
   - 尽早提交事务，避免事务长时间处于未提交状态。
2. **设置合理的超时时间**：通过控制台调整以下参数，避免长事务无限期占用资源，详细操作请参见 设置实例参数 。
   - `max_execution_time`：SELECT 语句的最大执行时间（毫秒），设置为合理值可防止慢查询长时间运行。
   - `wait_timeout`：连接空闲超时时间（秒），超过此时间未活动的连接将被自动关闭。

步骤7：确认 DDL 事务拆分

`ALTER TABLE` 等大表 DDL 操作会进行数据拷贝，可能产生大事务。TDSQL Boundless 支持在 DDL 期间自动拆分事务，避免单事务过大。请根据业务需要调整以下参数，详细操作请参见 设置实例参数 。

- `tdsql_split_trans_during_alter_copy` 参数：确认该参数值为 `ON` 。若为 `OFF` ，建议修改为 `ON` ，使 DDL 数据拷贝期间自动拆分小事务提交。
- `tdsql_bulk_commit_records` 参数：控制 DDL 拷贝期间每批提交的记录数。默认值为3000，可根据表数据量适当调整。

步骤8：调整事务保活时间

若业务存在长时间执行的合法大事务（如大数据量 ETL），需确保事务保活时间足够长，避免事务因超时被自动释放。
调整 `tdsql_txn_keep_alive_lease_sec` 参数。该参数控制 TDStore 在多久未收到事务心跳后释放事务上下文，默认值为 3600 秒（1 小时）。若大事务执行时间超过此值，可适当调大。详细操作请参见 设置实例参数 。

> **注意：**
> 调大事务保活时间会增加事务上下文的内存占用时间。若实例中同时存在大量长事务，可能导致内存压力增大。请根据实际业务需求合理设置。

预防措施

为避免事务过大问题反复发生，建议采取以下预防措施：

- **配置告警策略**：在控制台为 SQL 失败数和内存利用率配置告警，及时发现事务过大问题。
- **规范事务设计**：制定事务开发规范，控制单事务的数据量和执行时间，避免大事务和长事务。
- **分批处理批量任务**：大批量数据导入和更新操作必须分批执行，每批次数据量控制在合理范围内。
- **监控事务指标**：定期关注每秒处理的事务数和 SQL 失败数指标，及时发现事务异常。
- **DDL 操作规范**：大表 DDL 操作前确认 `tdsql_split_trans_during_alter_copy` 已开启，避免 DDL 产生过大事务。

应用死锁

最近更新时间：2026-07-10 10:54:30

现象描述

TDSQL Boundless 实例在运行过程中，业务侧出现死锁报错，可能伴随以下现象：

- 应用程序日志中出现死锁错误，错误码为 `1213`（`ER_LOCK_DEADLOCK`）。
- 应用程序日志中出现锁等待超时错误，错误码为 `1205`（`ER_LOCK_WAIT_TIMEOUT`）。
- 实例监控中 SQL 失败数指标升高。
- 高并发场景下，部分事务被自动回滚并提示 Deadlock found when trying to get lock。

您可在 TDSQL Boundless 控制台的**实例详情 > 监控告警 > 指标监控**页面查看 SQL 失败数、活跃线程数等指标，判断死锁发生频率和时间点。

可能原因

死锁通常由多个事务相互持有对方需要的锁资源，形成等待环导致。可能原因如下：

序号	可能原因
1	多个事务以不同顺序访问相同的表或行，形成锁等待环。
2	事务中执行时间过长，持有锁的时间过长，增加死锁概率。
3	并发量突增，锁竞争加剧，短时间内大量事务争抢同一批资源。
4	死锁检测未开启或锁等待超时设置不合理，导致死锁无法被及时检测和解除。

解决思路

针对不同原因，解决思路如下：

可能原因	解决思路
事务访问顺序不一致	统一事务中表的访问顺序，确保并发事务按相同顺序加锁。
事务执行时间过长	拆分大事务为小事务，减少单事务持锁时间；优化事务内 SQL 执行效率。
并发量突增	业务侧进行限流或错峰执行，降低并发争抢强度。
死锁检测未开启	通过控制台开启死锁检测，并合理设置锁等待超时时间。

处理步骤

步骤1：查看监控指标

1. 登录 TDSQL Boundless 控制台，在**实例列表**中单击目标**实例 ID**，进入**实例详情**页。
2. 选择**监控告警 > 指标监控**页签，查看以下指标的变化趋势：

监控指标	说明	关注点
SQL 失败数	每秒失败的 SQL 数量	死锁发生时，失败数会突增
活跃线程数	当前活跃线程数	并发量是否突增
慢查询数	每秒慢查询数量	锁等待是否导致查询变慢
CPU 利用率	实例 CPU 使用率	锁竞争是否导致 CPU 升高

重点关注 SQL 失败数突增的时间点，与业务并发高峰或批量任务执行时间是否吻合。

步骤2：查看死锁信息

通过系统视图查看死锁记录，分析死锁发生的事务和锁等待关系。

1. 连接数据库实例，执行以下 SQL 查看死锁记录。

```
SELECT
    rollback_trans_id,
    cycle_transactions,
    occurred_at
FROM information_schema.TDSTORE_PESSIMISTIC_DEADLOCK_INFO
ORDER BY occurred_at DESC
LIMIT 20;
```

该视图展示最近发生的死锁信息，字段说明如下：

- `rollback_trans_id`：被回滚的事务 ID。
- `cycle_transactions`：构成死锁环的事务列表。
- `occurred_at`：死锁发生时间。

2. 如需查看死锁详情，执行以下 SQL 查看死锁的锁等待详情。

```
SELECT
    rollback_trans_id,
    requesting_node,
    requesting_trans_id,
    blocking_trans_id,
    req_lock_range,
```

```
    blk_lock_range,
    occurred_at,
    requesting_sql
FROM information_schema.TDSTORE_PESSIMISTIC_DEADLOCK_DETAIL_INFO
ORDER BY occurred_at DESC
LIMIT 20
```

该视图展示死锁发生时的锁等待关系，帮助您定位冲突的具体资源和事务。

步骤3：查看当前锁等待会话

若死锁正在发生或存在长时间锁等待，查看当前会话的锁等待情况。

1. 执行以下 SQL 查看当前正在执行的会话，重点关注 `TIME` 值较大的会话，这些会话可能正在等待锁资源。

```
SELECT
    ID,
    USER,
    HOST,
    DB,
    COMMAND,
    TIME,
    STATE,
    INFO
FROM information_schema.PROCESSLIST
WHERE COMMAND != 'Sleep'
ORDER BY TIME DESC
LIMIT 20
```

2. 若发现长时间锁等待的会话，可使用 `KILL` 命令终止该会话以释放锁资源。

```
KILL <会话ID>;
```

> **警告：**
> `KILL` 操作会中断正在执行的事务并触发回滚，请确认目标会话后再执行，避免误杀正常业务会话。

步骤4：开启死锁检测

若死锁检测未开启，建议通过控制台开启，使系统自动检测并解除死锁。

1. 登录 TDSQL Boundless 控制台，在**实例列表**中单击目标**实例 ID**，进入**实例详情**页。

2. 选择**参数设置** > **数据库参数**页签，查找以下参数：

参数名	说明	默认值
`tdstore_deadlock_detect`	是否开启死锁检测。开启后系统会自动检测死锁环并回滚其中一个事务。	OFF
`tdstore_deadlock_victim`	死锁发生时选择牺牲事务的策略。`WRITE_LEAST` 优先回滚写数据量较少的事务；`START_LATEST` 优先回滚较晚开启的事务。	WRITE_LEAST

3. 将 `tdstore_deadlock_detect` 的值修改为 `ON`，开启死锁检测。
4. 根据业务场景设置 `tdstore_deadlock_victim` 的值：
   - `WRITE_LEAST`：优先保护写数据量大的事务，适合以数据写入为主的场景。
   - `START_LATEST`：优先保护先开启的事务，适合事务执行顺序明确的场景。
5. 在目标参数所在行的**参数当前值**列，单击修改参数值，单击**保存**，在确认对话框中单击**确定**。

> **注意：**
> 开启死锁检测会带来少量的性能开销，但在死锁频发的场景下，开启检测可以避免事务长时间阻塞，整体收益大于开销。

步骤5：调整锁等待超时时间

若锁等待超时时间设置不合理（过长或过短），可通过控制台调整 `tdsql_lock_wait_timeout` 参数。

1. 在**参数设置** > **数据库参数**页签，查找 `tdsql_lock_wait_timeout` 参数。
2. 根据业务场景调整参数值：
   - 默认值为10秒。若业务事务执行时间较长，可适当调大，避免正常事务因锁等待被误杀。
   - 若希望死锁和锁等待尽快被解除，可适当调小，减少业务阻塞时间。
3. 在目标参数所在行的**参数当前值**列，单击修改参数值，单击**保存**，在确认对话框中单击**确定**。

步骤6：优化业务事务

从业务层面优化事务设计，从根本上减少死锁发生概率。

1. **统一访问顺序**：确保所有并发事务以相同的顺序访问表和行。例如，事务 A 和事务 B 都先更新表1再更新表2，避免交叉等待。
2. **缩短事务范围**：
   - 将大事务拆分为多个小事务，减少单事务持锁时间。
   - 事务中避免包含耗时操作（如远程调用、文件 IO），尽快提交事务。
   - 使用 `SELECT ... FOR UPDATE` 时，尽量只锁定需要的行，避免锁定过多资源。
3. **降低并发强度**：

- 对高并发写入场景进行限流，控制同时执行的事务数量。
- 批量操作分批执行，避免单批次操作锁定大量资源。
- 将非核心业务的写入操作错峰执行。

4. **使用合适的索引**：
   - 确保更新和删除操作使用索引定位行，避免全表扫描加锁。
   - 无索引的更新操作会锁定大量行，大幅增加死锁概率。

步骤7：优化 SQL 语句

锁竞争往往与低效 SQL 有关，优化 SQL 可以减少锁持有时间。

1. 对涉及锁竞争的 SQL 执行 `EXPLAIN` 查看执行计划。

```
EXPLAIN UPDATE table_name SET ... WHERE ...;
```

2. 重点关注以下信息：
   - `type` 列为 `ALL` 表示全表扫描，会锁定大量行，需要添加索引。
   - `rows` 列值过大说明扫描行数过多，锁定的行也越多。
3. 优化建议：
   - 为 `WHERE` 条件列添加索引，减少扫描和锁定行数。
   - 避免在没有索引的列上执行 `UPDATE` 或 `DELETE` 。
   - 使用 `LIMIT` 限制单次操作影响的行数。

预防措施

为避免死锁问题反复发生，建议采取以下预防措施：

- **开启死锁检测**：确保 `tdstore_deadlock_detect` 参数设置为 `ON` ，使系统自动检测并解除死锁。
- **配置告警策略**：在控制台为 SQL 失败数指标配置告警，及时发现死锁问题。
- **规范事务设计**：制定事务开发规范，统一表访问顺序，控制事务大小和持锁时间。
- **定期巡检慢 SQL**：通过 DBbrain 智能诊断定期检查慢 SQL，优化低效查询以减少锁持有时间。
- **监控 SQL 失败数**：定期关注控制台监控面板中的 SQL 失败数指标，若出现突增，及时排查是否存在死锁或锁等待问题。