

# 如何准确获取 MySQL 主从延迟时间？

原创 陈宇 爱可生开源社区 2025年03月19日 18:55 上海

作者：陈宇，现任爱可生南区项目经理，负责项目整体质量、安全、进度、成本管理责任保证体系。对开源技术执着，为客户负责，喜欢极限运动，足球。

爱可生开源社区出品，原创内容未经授权不得随意使用，转载请联系小编并注明来源。

本文约 1600 字，预计阅读需要 6 分钟。

## 1 背景

MySQL 5.7 已于 2023 年 10 月 EOL<sup>[1]</sup>，但仍然有大量的生产环境依赖此版本。本文撰写时间 2025 年 3 月。

不久前，在一套采用 MySQL 5.7 作为部署版本的生产环境中，由于业务执行了大规模事务，进而引发了 MySQL 主从复制的延迟，最终暴露出数据一致性方面的严重问题。

由于业务做了读写分离，从库读取的数据与主库不一致，影响了应用逻辑。业务团队提出明确需求：需要知道主从延迟的具体时间值，以评估影响并优化系统。

请读者思考一下：

1. 如何获取主从延迟时间值？

2. 如何判断获取的值是准确的？

随后我们分析了 MySQL 5.7 的内置指标 `Seconds_Behind_Master`<sup>[2]</sup> 的可靠性，并探索更精准的替代方案。

## 2 `Seconds_Behind_Master` 可靠吗？

`Seconds_Behind_Master` 是 `SHOW SLAVE STATUS` 输出中的字段，表示从库应用二进制日志事件时落后主库的秒数。

- 理论上，值为 0 表示从库已同步，较高的值则反映延迟。
- 实际上，你会发现该指标与真实延迟数值不符：数据明显差异时显示 0 或出现与复制性能无关的峰值。

这种现象的根源在于该值的计算方法和 MySQL 5.7 的复制架构设计。让我们结合源码剖析一下。

## 根源一：计算方法的局限性

`Seconds_Behind_Master` 的计算逻辑定义在 MySQL 5.7 的代码中：

```
longlong slave_seconds_behind_master(Master_info* mi)
{
    longlong t0 = mi->clock_diff_with_master;
    longlong t1 = mi->rli->last_master_timestamp;
    longlong t2 = mi->get_master_log_pos() ? time(NULL) : 0;
    return (t2 > t1) ? (t2 - t1 - t0) : 0;
}
```

### 变量说明

- `t0` (`clock_diff_with_master`)：校正主从时钟偏差。
- `t1` (`last_master_timestamp`)：主库二进制日志事件的时间戳。
- `t2` (`time(NULL)`)：从库当前时间（源码中实际使用 POSIX 时间函数）。
- 返回值为秒，源码中直接返回时间差，未除以 1000000。

### 问题点

该计算假定 `t1` 是事件在主库执行的时间，但实际上它是事件写入二进制日志的时间，受事务提交顺序和 `sync_binlog` 配置影响。例如，若 `sync_binlog=0`，日志写入可能滞后，导致 `t1` 与实际执行时间脱节。

## 根源二：单线程 SQL 线程的延迟掩盖

MySQL 5.7 默认使用单线程 SQL 线程应用事件，时间戳更新逻辑在代码中：

```
void Relay_log_info::set_master_log_pos(ulonglong pos)
{
    // 简化表示，实际更复杂
    if (pos)
        last_master_timestamp = log_pos_to_timestamp(pos);
}
```

### 问题点

若一个大事务（如批量 UPDATE）在从库执行耗时 10 秒，`last_master_timestamp` 仅在事务完成时更新。在此期间，`Seconds_Behind_Master` 不变，完成后才跳至 10，无法实时反映延迟。这正是我们生产环境中大事务导致延迟的关键原因。

### 根源三：并行复制的误报

MySQL 5.7 支持多线程复制 (slave\_parallel\_workers)，但 `Seconds_Behind_Master` 未有效处理并行执行：

```
if (mi->rli->slave_parallel_workers > 0 && mi->rli->last_master_timestamp)
    return time(NULL) - mi->rli->last_master_timestamp;
```

#### 问题点

该值仅基于最后应用的事件时间戳，未聚合各线程的延迟。若一个线程因大事务滞后，其他线程已同步，指标仍可能显示 0，掩盖真实延迟。

### 根源四：网络和 I/O 延迟的忽略

#### 问题点

`Seconds_Behind_Master` 不反映 I/O 线程从主库拉取事件或写入中继日志的延迟。若网络问题导致 I/O 线程落后，但 SQL 线程已处理完中继日志，指标仍误报 0。

#### 小结

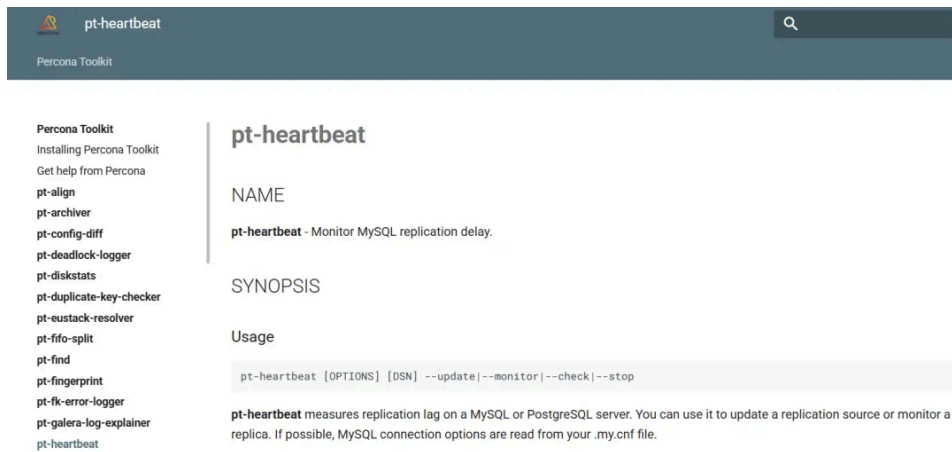
`Seconds_Behind_Master` 因依赖不准确的事件时间戳、缺乏实时更新、无法反映并行复制和 I/O 延迟，成为一个不可靠的指标。面对业务需求，它无法提供精确的延迟时间。

MySQL 5.7 的 `Seconds_Behind_Master` 并不可靠！

## 3 解决方案：pt-heartbeat

### 如何获取主从延迟时间值？

`pt-heartbeat`<sup>[3]</sup> 是 Percona Toolkit 中的工具，通过在主库注入心跳记录并在从库比较时间戳，提供精确的延迟测量。



来源：<https://docs.percona.com/percona-toolkit/pt-heartbeat.html>

操作步骤如下：

**主库：**运行更新心跳表。

```
-- 主库创建表 heartbeat.heartbeat，包含 ts（时间戳）和 server_id
-- 每秒更新一行记录 --interval=1
pt-heartbeat --user=root --password=xxx --create-table --update --interval=1 -D heartbeat
```

**从库：**监控或检查延迟。

```
-- 输出实时延迟，如 0.02s
pt-heartbeat --user=root --password=xxx --monitor -D heartbeat
```

## 如何判断获取的值是准确的？

从 *pt-heartbeat* 的 Perl 源码分析其原理：

### 心跳注入代码块

```
sub update_heartbeat {
    my ($dbh) = @_ ;
    my $ts = $dbh->selectrow_array('SELECT NOW(6)'); 主库当前时间
    my $server_id = $dbh->selectrow_array('SELECT @@server_id');
    $dbh->do("INSERT INTO heartbeat.heartbeat (id, ts, server_id) VALUES (1, ?, ?) "
        . "ON DUPLICATE KEY UPDATE ts = ?, server_id = ?", undef, $ts, $server_id, $ts, $server_id);
}
```

### 延迟计算代码块

```
sub check_heartbeat {  
    my ($dbh) = @_;  
  
    my $row = $dbh->selectrow_hashref("SELECT ts FROM heartbeat.heartbeat WHERE id = 1");  
  
    my $master_ts = $row->{ts};          主库时间戳  
  
    my $slave_ts = $dbh->selectrow_array('SELECT NOW(6)'); 从库当前时间  
  
    my $lag = time_diff($slave_ts, $master_ts); 计算延迟  
  
    return sprintf("%.2f", $lag);  
}
```

准确性分析：

- **实时性**：心跳记录每秒更新，延迟反映记录从主库写入到从库应用的时间，精确到微秒。
- **独立性**：不依赖 MySQL 复制线程的时间戳，避免了 `Seconds_Behind_Master` 的缺陷。
- **局限性**：需确保主从时钟同步，否则需用 `--skew` 调整。

## 4 结论与建议

在 MySQL 5.7 中，`Seconds_Behind_Master` 因设计缺陷无法满足业务对精确延迟的需求。源码分析揭示其依赖不准确的时间戳和缺乏实时性，尤其在大数据场景下表现不佳。

相比之下，`pt-heartbeat` 通过心跳机制提供实时、精确的延迟测量，是解决此类问题的理想工具。

### 建议

1. 在生产环境部署 `pt-heartbeat`，设置合理的 `--interval`（如 0.5 秒）以平衡精度和负载。
2. 配置主从时钟同步（如 NTP），确保延迟值可靠。
3. 结合业务需求，设置延迟阈值告警，优化大数据处理。

通过这一方案，我们不仅满足了业务需求，还提升了复制监控的可靠性，为系统稳定性提供了保障。

### 参考资料

- [1] eol-notice: <https://www.mysql.com/support/eol-notice.html>
- [2] show-slave-status: <https://dev.mysql.com/doc/refman/5.7/en/show-slave-status.html>
- [3] pt-heartbeat: <https://docs.percona.com/percona-toolkit/pt-heartbeat.html>

本文关键字：#MySQL# #pt-heartbeat# #事务# #主从复制# #percona-toolkit#



## 推荐阅读

MySQL 主从复制异常？试试 pt-slave-repair 吧！

pt-osc：特定场景下的数据清理神器

SQLC 是一款全方位的 SQL 质量管理平台，  
覆盖开发至生产环境的 SQL 审核和管理。



支持主流开源、商业、国产数据库。  
为开发者、DBA、运维人员提供流程自动化  
能力，提升上线效率，提高数据质量。




请添加小助手加入  
SQLC 技术交流群


SQLC 企业版  
商务咨询/预约演示



✦ Github : <https://github.com/actiontech/sqlc>

📖 文档 : <https://actiontech.github.io/sqlc-docs/>

 官网：<https://opensource.actionsky.com/sqlle/>

 微信群：请添加小助手加入 ActionOpenSource

 商业支持：<https://www.actionsky.com/sqlle>



MySQL 231    pt-heartbeat 1    事务 24    主从复制 12    percona-toolkit 3

MySQL · 目录

上一篇

第 52 期：根据 EXPLAIN EXTRA 栏提示进行优化（四）

下一篇

第 53 期：EXPLAIN 中最直观的 rows

[阅读原文](#)