深度好文：如何发现及处理 MySQL 主从延迟问题

在 Percona MySQL 支持团队中，我们经常看到客户抱怨复制延迟的问题。当然，这对 MySQL 用户来说并不是什么新鲜事，多年来我们在 MySQL 性能博客上发表过一些关于这个主题的文章（过去有两篇特别受欢迎的文章："Reasons for MySQL Replication Lag" 和 “Managing Slave Lag with MySQL Replication"），两篇文章均由 Percona 首席执行官 Peter Zaitsev 撰写）。

> 译者注：Percona 公司是做 MySQL 发行版的，MySQL 有三大发行版，MySQL、MariaDB、Percona，《高性能 MySQL》这本神作就是出自 Percona 的专家团队。反正就很🐂就行了。

在今天的文章中，我将分享一些发现复制延迟的新方法 - 包括从服务器滞后的可能原因 - 以及如何解决这个问题。

如何发现复制延迟

MySQL 复制有两个线程：IO_THREAD 和 SQL_THREAD。IO_THREAD 连接到 master，从 master 读取 binlog 事件，并将其复制到名为 relay log 的本地日志文件中。另一方面，SQL_THREAD 在从节点上读取 relay log，然后尽可能快地处理这些日志。每当复制出现延迟时，首先要弄清延迟发生在 IO_THREAD 还是 SQL_THREAD。

通常情况下，I/O 线程不会造成巨大的复制延迟，因为它只是从主服务器读取 binlog。不过，这取决于网络连接、网络延迟...即服务器之间的速度有多快。Slave 的 I/O 线程可能会因为带宽拥塞而变慢。通常，当 Slave IO_THREAD 能够足够快地读取 binlog 时，就容易在 Slave 上堆积 relay log – 此时表明 Slave IO_THREAD 是没问题的。

另一方面，如果是 Slave SQL_THREAD 导致延迟，大概率是因为来自 replication stream 的 queries 在 Slave 上执行的时间太长。可能的原因包括 Master、Slave 之间的硬件不同、索引不同、工作负载不同。此外，Slave OLTP 工作负载有时会因为“锁”而导致复制延迟。例如，对 MyISAM 表的长久读请求会阻塞 SQL 线程，或对 InnoDB 表的任何事务都会创建 IX 锁并阻塞 SQL 线程中的 DDL。此外，还要考虑到在 MySQL 5.6 之前，slave 是单线程的，这也是导致 Slave SQL_THREAD 出现延迟的另一个原因。

MySQL 复制延迟的例子

让我通过 master status / slave status 示例向您展示，以确定 Slave 延迟问题到底是由于 IO_THREAD 还是由于 SQL_THREAD。

```
mysql-master> SHOW MASTER STATUS;
+------------------+--------------+------------------+-----------------------------------
---------------------------------+
| File | Position  | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set
|
+------------------+--------------+------------------+-----------------------------------
---------------------------------+
| mysql-bin.018196 | 15818564     |                  | bb11b389-d2a7-11e3-b82b-5cf3fcfc8
f58:1-2331947                    |
+------------------+--------------+------------------+-----------------------------------
---------------------------------+mysql-slave> SHOW SLAVE STATUSG\G
*************************** 1. row ***************************
Slave_IO_State: Queueing master event to the relay log
Master_Host: master.example.com
Master_User: repl
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.018192
Read_Master_Log_Pos: 10050480
Relay_Log_File: mysql-relay-bin.001796
Relay_Log_Pos: 157090
Relay_Master_Log_File: mysql-bin.018192
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 5395871
Relay_Log_Space: 10056139
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
```

```
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 230775
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 2
Master_UUID: bb11b389-d2a7-11e3-b82b-5cf3fcfc8f58:2-973166
Master_Info_File: /var/lib/mysql/i1/data/master.info
SQL_Delay: 0
SQL_Remaining_Delay: NULL
Slave_SQL_Running_State: Reading event from the relay log
Master_Retry_Count: 86400
Master_Bind:
Last_IO_Error_Timestamp:
Last_SQL_Error_Timestamp:
Master_SSL_Crl:
Master_SSL_Crlpath:
Retrieved_Gtid_Set: bb11b389-d2a7-11e3-b82b-5cf3fcfc8f58:2-973166
Executed_Gtid_Set: bb11b389-d2a7-11e3-b82b-5cf3fcfc8f58:2-973166,
ea75c885-c2c5-11e3-b8ee-5cf3fcfc9640:1-1370
Auto_Position: 1
```

这清楚地表明，Slave IO_THREAD 滞后，显然 Slave SQL_THREAD 也因此滞后，从而导致复制延迟。正如你所看到的，Master 日志文件是 mysql-bin.018196（来自 SHOW MASTER STATUS），而 Slave IO_THREAD 在 mysql-bin.018192（来自 Slave status 的 Master_Log_File）上，这表明 Slave IO_THREAD 正在从该文件读取数据，而在 Master 上，它正在写入 mysql-bin.018196，因此 Slave IO_THREAD 落后了 4 个 binlog。与此同时，Slave SQL_THREAD 正在读取同一个文件，即 mysql-bin.018192（Slave status 中的 Relay_Master_Log_File），这表明 Slave SQL_THREAD 正在以足够快的速度应用事件，但它也滞后了，这可以从显示 Slave status 输出中的 Read_Master_Log_Pos 与 Exec_Master_Log_Pos 之间的差值观察到。

`show slave status` 的输出中 Master_Log_File 和 Relay_Master_Log_File 值相同，我们可以根据 `Read_Master_Log_Pos - Exec_Master_Log_Pos` 计算 Slave SQL_THREAD 的滞后时间。这样就能大致了解 Slave SQL_THREAD 应用事件(apply event)的速度。如上所述，如果 Slave IO_THREAD 滞后，那么 Slave SQL_THREAD 当然也会滞后。有关显示 Slave 状态输出字段的详细说明，请点击此处。

此外，Seconds_Behind_Master 显示了以秒为单位的巨大延迟。不过，这可能会产生误导，因为它只度量最近执行的 relay log 与最近被 IO_THREAD 下载的 relay log 条目之间的时间戳差异。如果 Master 上有更多的 relay log，Slave 并不会将它们计入 Seconds_behind_master 的计算中。 你可以使用 Percona 工具包中的 pt-heartbeat 更准确地测量 Slave 日志的滞后情况。至此，我们学会了如何检查复制延迟 – 无论是 Slave IO_THREAD 还是 Slave SQL_THREAD。现在，让我来提供一些提示和建议，看看到底是什么原因导致了这种延迟。

提示和建议 - 导致复制延迟的原因及可能的修复方法

通常，Slave IO_THREAD 滞后是因为主/从之间的网络速度太慢。大多数情况下，启用 Slave 压缩协议（slave_compressed_protocol）有助于缓解 Slave IO_THREAD 的滞后。还有一个建议是禁用 Slave 上的 binlog 记录，因为它也是 IO 密集型的，除非你需要它来进行时间点恢复。

要尽量减少 Slave SQL_THREAD 的滞后，重点是优化查询。我的建议是启用配置选项 log_slow_slave_statements，这样 Slave 执行的耗时超过 long_query_time 的查询就会被记录到慢日志中。为了收集更多有关查询性能的信息，我还建议将配置选项 log_slow_verbosity 设置为"full”。

这样，我们就能看到是否有 Slave SQL_thread 执行的查询需要很长时间才能完成。关于如何在特定时间段内使用上述选项启用慢查询日志，你可以点击这里查看我之前的文章。需要提醒的是，log_slow_slave_statements 变量是在 Percona Server 5.1 中首次引入的，现在从 5.6.11 版起已成为 Vanilla MySQL 的一部分。在上游版本的 MySQL 中，log_slow_slave_statements 被作为命令行选项引入。详情请点击此处，而 log_slow_verbosity 是 Percona Server 的特定功能。

如果使用基于行的 binlog 格式，在 Slave SQL_THREAD 上出现延迟的另一个原因是：如果任何数据库表缺少主键或唯一键，就会在 Slave SQL_THREAD 上扫描表的所有行进行 DML，从而导致复制延迟，因此要确保所有表都有主键或唯一键。有关详细信息，请查看此错误报告 http://bugs.mysql.com/bug.php?id=53375 您可以在 Slave 上使用以下查询来确定哪些数据库表缺少主键或唯一键。

```
mysql> SELECT t.table_schema,t.table_name,engine
FROM information_schema.tables t INNER JOIN information_schema .columns c
on t.table_schema=c.table_schema and t.table_name=c.table_name
GROUP BY t.table_schema,t.table_name
HAVING sum(if(column_key in ('PRI','UNI'), 1,0)) =0;
```

在 MySQL 5.6 中，针对这种情况进行了一项改进，在使用内存散列的情况下，slave_rows_search_algorithms 可以解这个问题。

请注意，当我们读取巨大的 RBR 事件时，Seconds_Behind_Master 并没有更新，因此 “滞后” 可能仅仅与此有关 – 我们还没有完成对事件的读取。例如，在基于行的复制中，庞大的事务可能会导致 Slave 端出现延迟，比如，如果你有一个 1000 万行的表，而你执行了 `DELETE FROM table WHERE id < 5000000` 操作，500 万行将被发送到 Slave 端，每一行都是单独的，速度会慢得令人痛苦。因此，如果必须不时地从庞大的表中删除最旧的行，那么使用分区可能是一个不错的选择，在某些工作负载中，使用 DROP 旧分区可能比使用 DELETE 更好，而且只有语句会被复制，因为这将是 DDL 操作。

为了更好地解释这个问题，假设分区 1 保存的行的 ID 从 1 到 1000000，分区 2 的 ID 从 1000001 到 2000000，以此类推，所以与其通过语句 `DELETE FROM table WHERE ID<=1000000` 进行删除，不如执行 `ALTER TABLE DROP partition1` 。有关更改分区操作，请查阅手册 - 也请查阅我的同事 Roman 的这篇精彩文章，其中解释了复制延迟的可能原因。

pt-stalk 是 Percona 工具包中最优秀的工具之一，它可以在出现问题时收集诊断数据。你可以按如下方式设置 pt-stalk，这样只要出现 Slave 滞后，它就能记录诊断信息，我们随后就可以对这些信息进行分析，看看到底是什么原因导致了滞后。

```
------- pt-plug.sh contents
#!/bin/bash
trg_plugin() {
mysqladmin $EXT_ARGV ping &> /dev/null
mysqld_alive=$?
if [[ $mysqld_alive == 0 ]]
then
seconds_behind_master=$(mysql $EXT_ARGV -e "show slave status" --vertical | grep Seconds
_Behind_Master | awk '{print $2}')
echo $seconds_behind_master
else
echo 1
fi
}
# Uncomment below to test that trg_plugin function works as expected
#trg_plugin
-------
-- That's the pt-plug.sh file you would need to create and then use it as below with pt-
stalk:
$ /usr/bin/pt-stalk --function=/root/pt-plug.sh --variable=seconds_behind_master --thres
hold=300 --cycles=60 --notify-by-email=muhammad@example.com --log=/root/pt-stalk.log --p
id=/root/pt-stalk.pid --daemonize
```

你可以调整阈值，目前是 300 秒，结合 -cycles 选项，这意味着如果 seconds_behind_master 值大于等于 300，持续 60 秒或更长时间，pt-stalk 就会开始捕获数据。添加 `--notify-by-email` 选项后，pt-stalk 捕获数据时就会通过电子邮件通知。你可以相应调整 pt-stalk 的阈值，这样它就会在问题发生时触发采集诊断数据。

结论

滞后 Slave 是一个棘手的问题，但也是 MySQL 复制中的常见问题。在这篇文章中，我试图涵盖 MySQL 复制 Slave 延迟的大多数方面。如果你知道复制延迟的其他原因，请在评论区与我分享。

> *本文翻译自：https://www.percona.com/blog/how-to-identify-and-cure-mysql-replication-slave-lag/*

推荐阅读：

- 太卷了，史上最简单的监控系统 catpaw 简介
- 告警聚合降噪、告警升级、告警认领、告警排班、告警协同，一网打尽
- 面向故障处理的可观测性体系建设