Mysql Group Replication 官方文档译文

京东商城基础平台–数据库技术部

译文指导： 京东老樊
翻　　译： 张璐 王思佳 张树臣 翟敏 赵晨
校　　对： 王伟 刘风才 刘岩

前言

MySQL Group Replication（简称 MGR）是 MySQL 官方于 2016 年 12 月推出的一个全新的高可用与高扩展的解决方案。MySQL 组复制提供了高可用、高扩展、高可靠的 MySQL 集群服务。高一致性，基于原生复制及 paxos 协议的组复制技术，并以插件的方式提供，提供一致数据安全保证；高容错性，只要不是大多数节点坏掉就可以继续工作，有自动检测机制，当不同节点产生资源争用冲突时，不会出现错误，按照先到者优先原则进行处理，并且内置了自动化脑裂防护机制；高扩展性，节点的新增和移除都是自动的，新节点加入后，会自动从其他节点上同步状态，直到新节点和其他节点保持一致，如果某节点被移除了，其他节点自动更新组信息，自动维护新的组信息；高灵活性，有单主模式和多主模式，单主模式下，会自动选主，所有更新操作都在主上进行；多主模式下，所有 server 都可以同时处理更新操作。

MGR 是 MySQL 数据库未来发展的一个重要方向。京东商城数据库技术部对此作出积极响应，在最短的时间内立项，对 MGR 进行研究测试。为了使研究以及后续的运维推广工作更加简便，特此将 MGR 官方文档译为中文，谨供业内人士参考。由于本章内容较多，翻译时间紧迫，尽管我们尽量做到认真仔细，但还是难以避免出现错误和不尽如人意的地方，在此欢迎广大读者批评指正。

目录

本章介绍 MySQL 组复制以及如何安装，配置和监控。 MySQL 组复制是一个能够让您创建弹性化、高可用、容错复制结构的 MySQL 插件。

组复制可以在两种模式下运行。 在单主模式下，组复制具有自动选主功能，每次只有一个 server 成员接受更新。在多主模式下运行时，所有的 server 成员都可以同时接受更新。

组复制中内置的组成员服务，用于保持组信息的视图一致，并在任何给定的时间点对于所有 server 成员可用。 当 Server 成员离开或加入组时，视图会相应地进行更新。 当有 server 成员意外离开组时，故障检测机制会检测到此情况，并自动通知组当前视图已更改。

本章的结构如下：

- 第 19.1 节“组复制背景”介绍组以及组复制的工作原理。
- 第 19.2 节“入门指南”介绍如何通过配置多个 MySQL Server 实例来创建组。
- 第 19.3 节“监控组复制”介绍如何监控组。
- 第 19.5 节“组复制安全性”介绍如何保护组。
- 第 19.9 节“组复制技术详细信息”详细介绍组复制是如何工作的。

19.1 组复制背景

本节介绍有关 MySQL 组复制的背景信息。

创建容错系统的最常见方法是创建组件冗余，换句话说，组件可以删除，而系统应该继续按预期运行。这就造成了一系列的挑战，将这种系统的复杂性提高到一个完全不同的水平。具体来说，复制的数据库需要同时维护和管理若干个 server 成员，而不只是一个。 此外，当多个 server 协同工作时，系统必须处理其他一些常见的分布式系统问题，诸如断网或脑裂等情况。

因此，最大的挑战是将数据库和数据复制的逻辑与若干个 server 以简单一致的方式协调运行的逻辑相融合。 换句话说，也就是使多个 server 成员关于系统的状态和系统每次变更的数据保持一致。 这可以被概括为使多个 server 对于每个数据库状态转换达成共识，从而使它们都作为一个独立的数据库运行，或者说它们最终达到相同状态。 这就意味着它们需要作为（分布式）state machine 运行。

MySQL 组复制提供了一种强大的 server 间协调机制的分布式 state machine 复制。 组中的 server 成员会自动地进行协调。

组复制可以在两种模式下运行。 在单主模式下，组复制具有自动选主功能，每次只有一个 server 成员接受更新。在多主模式下运行时，所有的 server 成员都可以同时接受更新。

这种功能就要求应用程序不得不解决部署所带来的限制。内置的组成员服务，用于保持组视图的一致性，并在任何给定的时间点对于所有 server 可用。当 Server 离开或加入组时，视图会相应地进行更新。 server 也可能会意外离开组，故障检测机制会自动检测到此情况，并通知组该视图已更改。

对于要提交的事务，决定提交或中止事务是由每个 server 单独完成的，但所有组成员必须就该事务在全局事务序列中的顺序达成一致意见。如果存在网络分隔，造成组成员间无法达成协议，则系统在此问题解决前将不会继续运行。 因此，组复制还内置了一个自动的脑裂保护机制。

这种机制都是由系统提供的组通信协议提供支持的。 该协议保障了故障检测机制，组成员服务的安全和消息的完全有序传递。该技术的核心是 Paxos 算法实现的，是组复制中保证数据一致性复制的关键， 它充当了组通信系统的引擎。

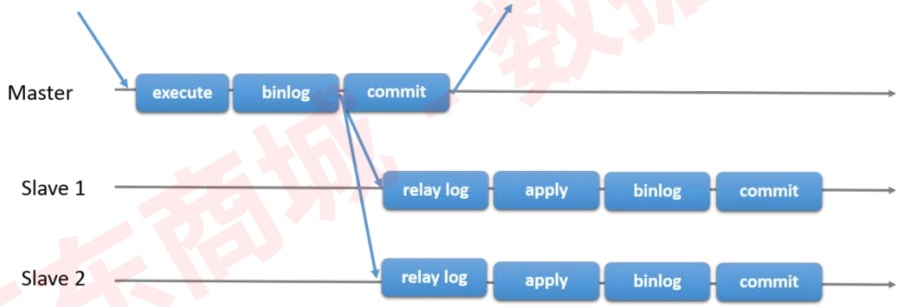
19.1.1 复制技术

在介绍 MySQL 组复制的详细信息之前，本节将简要介绍一些背景概念以及组复制是如何运行的。通过本节我们可以了解组复制中需要什么，以及传统异步 MySQL 复制和组复制之间的区别。

19.1.1.1 主–从复制

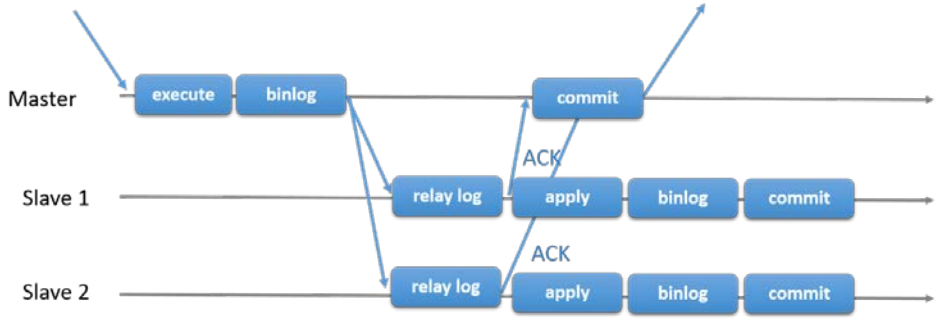
传统的 MySQL 复制提供了一种简单的主–从复制方法。 有一个主，以及一个或多个从。 主节点执行和提交事务，然后将它们（异步地）发送到从节点，以重新执行（在基于语句的复制中）或应用（在基于行的复制中）。 这是一个 shared-nothing 的系统，默认情况下所有 server 成员都有一个完整的数据副本。

图 19.1 MySQL 异步复制



还有一个半同步复制，它在协议中添加了一个同步步骤。 这意味着主节点在提交时需要等待从节点确认它已经接收到事务。只有这样，主节点才能继续提交操作。

图 19.2 MySQL 异步复制



在上面的两个图片中，可以看到传统异步 MySQL 复制协议（以及半同步）的图形展示。 蓝色箭头表示在不同 server 之间或者 server 与 client 应用之间的信息交互。

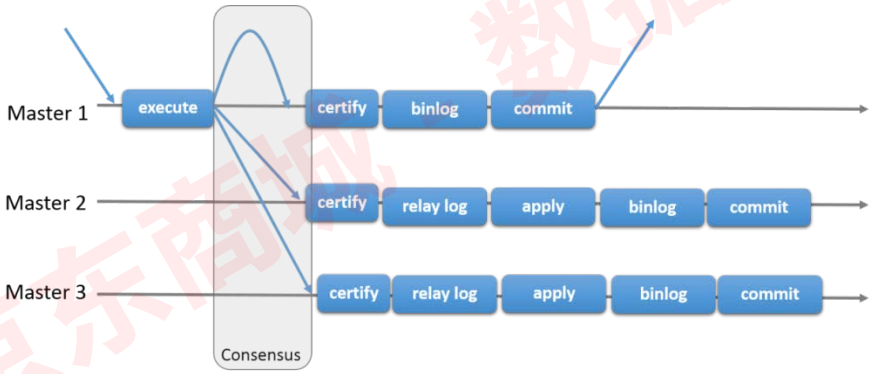
19.1.1.2 组复制

组复制是一种可用于实现容错系统的技术。复制组是一个通过消息传递相互交互的server集群。通信层提供了原子消息（atomic message）和完全有序信息交互等保障机制。 这些是非常强大的功能，我们可以据此架构设计更高级的数据库复制解决方案。

MySQL 组复制以这些功能和架构为基础，实现了基于复制协议的多主更新。复制组由多个 server 成员构成，并且组中的每个 server 成员可以独立地执行事务。但所有读写（RW）事务只有在冲突检测成功后才会提交。只读（RO）事务不需要在冲突检测，可以立即提交。换句话说，对于任何 RW 事务，提交操作并不是由始发 server 单向决定的，而是由组来决定是否提交。准确地说，在始发 server 上，当事务准备好提交时，该 server 会广播写入值（已改变的行）和对应的写入集（已更新的行的唯一标识符）。然后会为该事务建立一个全局的顺序。最终，这意味着所有 server 成员以相同的顺序接收同一组事务。因此，所有 server 成员以相同的顺序应用相同的更改，以确保组内一致。

在不同 server 上并发执行的事务可能存在冲突。 根据组复制的冲突检测机制，对两个不同的并发事务的写集合进行检测。如在不同的 server 成员执行两个更新同一行的并发事务，则会出现冲突。排在最前面的事务可以在所有 server 成员上提交，第二个事务在源 server 上回滚，并在组中的其他 server 上删除。 这就是分布式的先提交当选规则。

图 19.3 MySQL 组复制协议



最后，组复制是一种 share-nothing 复制方案，其中每个 server 成员都有自己的完整数据副本。

上图描述了 MySQL 组复制协议，并通过将其与 MySQL 复制（MySQL 半同步复制）进行比较，可以看到一些差异。 需要注意的是，这个图片中不包含一些基本共识和 Paxos 相关的信息。

19.1.2 组复制用例

组复制使您能够根据在一组 server 中复制系统的状态来创建具有冗余的容错系统。因此，只要它不是全部或多数 server 发生故障，即使有一些 server 故障，系统仍然可用，最多只是性能和可伸缩性降低，但它仍然可用。server 故障是孤立并且独立的。它们由组成员服务来监控，组成员服务依赖于分布式故障检测系统，其能够在任何 server 自愿地或由于意外停止而离开组时发出信号。他们是由一个分布式恢复程序来确保当有 server 加入组时，它们会自动更新组信息到

最新。并且多主更新确保了即使在单个服务器故障的情况下也不会阻止更新，不必进行 server 故障转移。因此，MySQL 组复制保证数据库服务持续可用。

值得注意的一点是，尽管数据库服务可用，但当有一个 server 崩溃时，连接到它的客户端必须重定向或故障转移到不同的 server。 这不是组复制要解决的问题。 连接器，负载均衡器，路由器或其他形式的中间件更适合处理这个问题。

总之，MySQL 组复制提供了高可用性，高弹性，可靠的 MySQL 服务。

19.1.2.1 用例场景示例

以下示例是组复制的典型用例。

- ***弹性复制*** - 需要非常流畅的复制基础架构环境，其中 server 的数量必须动态增长或收缩，并尽可能减少副作用。 例如，云数据库服务。
- ***高可用分片（Shards）*** - 分片是实现写扩展的常用方法。 使用 MySQL 组复制实现高可用性分片，其中每个分片映射到一个复制组。
- ***替代主从复制*** - 在某些情况下，使用单个主服务器会造成单点争用，写入整个组可能更具可扩展性。
- ***自动系统*** - 此外，您可以将 MySQL 组复制直接部署到已有复制协议的自动化系统中（在本章和前面的章节中已经描述过）。

19.1.3 组复制详细信息

本节介绍有关组复制基础服务的详细信息。

19.1.3.1 故障检测

组复制提供了一种故障检测机制，它能够找到并报告哪些 server 成员是无响应的，并且假定这些 server 已死。在更高级别来说，故障检测是提供关于哪些 server 可能已死的信息（猜测）的分布式服务。 然后，如果组同意该猜测可能是真的，则组判定给定的 server 确实已经 failed。 这意味着组中的其余成员进行协调决定以排除给定成员。

某个 server 无响应时触发猜测， 当 server A 在给定时间段内没有从 server B 接收消息时，将会发生超时并且触发猜测。

如果某个 server 与组的其余成员隔离，则它会怀疑所有其他 server 都失败了。 由于无法与组达成协议（因为它无法确保仲裁成员数），其怀疑不会产生后果。 当服务器以此方式与组隔离时，它无法执行任何本地事务。

19.1.3.2 组成员关系

MySQL 组复制依赖于组成员服务。这是一个内置的插件。它定义了哪些 server 在线并在组中。在线 server 列表通常称为视图。 因此，组中的每个 server 对于给定时刻积极参与组中的成员具有一致的视图。

同组 server 不仅需要关于事务提交必须达成一致意见，关于当前视图也是。 因此，如果同组 server 同意新的 server 加入，则该组本身将被重新配置从而将该 server 加入其中，并触发视图更新。 相反，如果 server 离开组，无论自愿或被迫的情况，该组都会动态地重新规划其配置，并触发视图更新。

要注意的是，当成员自愿离开时，它首先启动组的动态重新配置。 这触发一个过程，其中所有成员必须就不包含已离开 server 的新视图达成一致。 然而，如果成员由于发生意外而离开（例如它意外停止或网络连接断开），则故障检测机制检测到后，将提出该组的重新配置，去除故障成员。 如上所述，这需要来自组中大多数服务器达成一致意见。 如果组不能够达成一致（例如，当大多数服务器都不在线的情况），则系统不能动态地改变配置，而且系统会锁定以防止脑裂情况的发生。 最终，这意味着管理员需要介入并解决这个问题。

19.1.3.3 容错

MySQL 组复制构建在 Paxos 分布式算法实现的基础上，以提供不同 server 之间的分布式协调。因此，它需要大多数 server 处于活动状态以达到仲裁成员数，从而做出决定。 这对系统可以容忍的不影响其自身及其整体功能的故障数量有直接影响。 容忍 f 个故障所需的 server 数量（n）为 n = 2×f + 1。
在实践中，这意味着为了容忍一个故障，组必须有三个 server。 因此，如果一个服务器故障，仍然有两个服务器形成大多数(三分之二)来允许系统自动地继续运行。但是，如果第二个 server 意外地 fail 掉，则该组（剩下一个 server）锁定，因为没有多数可以达成决议。
以下是说明上述公式的小表。

组大小	多数	允许的即时故障数
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

下一章将涵盖组复制技术方面的知识。

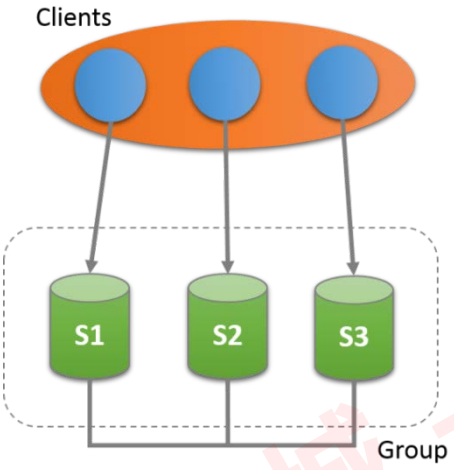
19.2 入门指南

MySQL 组复制是 MySQL server 的插件，组中的每个 server 都需要配置和安装该插件。 本节提供了一个详细的教程，其中包含创建至少三台 server 的复制组所需的步骤。

19.2.1 在单主模式下部署组复制

组中的每个 server 实例可以在独立的物理机器上运行，也可以在同一台机器上运行。 本节介绍如何在一台物理机上创建具有三个 MySQL Server 实例的复制组。这意味着需要三个数据目录，每个 server 实例一个，每个实例都需要单独配置。

图 19.4 组架构



本教程介绍如何使用组复制插件获取和部署 MySQL Server，如何在创建组之前配置每个 server 实例以及如何使用 Performance Schema 来验证一切是否正常。

19.2.1.1 部署组复制的实例

第一步是部署 MySQL 服务器的三个实例。 组复制是 MySQL Server 5.7.17 及更高版本提供的内置 MySQL 插件。有关 MySQL 插件的更多背景信息，请参见第 6.5 节“MySQL 服务器插件”。下载 MySQL 服务器包后，需要解压缩并安装二进制文件。 此过程假定 MySQL 服务器已下载并解压缩到当前目录，该目录需要在 mysql-5.7 的目录下。 由于本教程使用一个物理机，每个 MySQL 实例都需要一个特定的数据目录，用于存储实例的数据。 在名为 data 的目录中创建数据目录，并初始化每个目录。

```
mkdir data

mysql-5.7/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-5.7
--datadir=$PWD/data/s1
```

```
mysql-5.7/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-5.7
--datadir=$PWD/data/s2

mysql-5.7/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-5.7
--datadir=$PWD/data/s3
```

在 data / s1，data / s2，data / s3 中是一个初始化的数据目录，包含 mysql 系统数据库和相关表等。 要了解有关初始化过程的更多信息，请参见第 2.10.1.1 节“使用 mysqld 手动初始化数据目录”。

Warning

不要在生产环境中使用--initialize-insecure，它只用于简化教程。 有关安全设置的更多信息，请参见第 19.5 节“组复制安全性”。

19.2.1.2 配置组复制实例

本节介绍如何配置组复制的 MySQL Server 实例。 有关背景信息，请参见第 19.7.2 节“使用限制”。

组复制 Server 设置

要安装和使用组复制插件，必须正确配置 MySQLserver 实例。 建议将配置存储在 my.cnf 文件中。 有关详细信息，请参见第 5.2.6 节“文件的使用”。 如没有特殊说明，以下是组中第一个实例的配置，在此节中称为 s1。 以下部分展示 server 的示例配置。

```
[mysqld]

# server configuration

datadir=<full_path_to_data>/data/s1

basedir=<full_path_to_bin>/mysql-5.7/

port=24801

socket=<full_path_to_sock_dir>/s1.sock
```

这些设置对 MySQL server 进行了配置，使其使用先前已创建的数据目录，并配置 server 应打开哪个端口，并开始侦听传入连接。

注意

在此使用非默认端口 24801，因为在本教程中，三个服务器实例使用相同的主机名。 在具有三个不同机器的环境中，这种设置不是必需的。

复制框架

以下设置根据 MySQL 组复制要求配置复制。

```
server_id=1

gtid_mode=ON

enforce_gtid_consistency=ON

master_info_repository=TABLE

relay_log_info_repository=TABLE

binlog_checksum=NONE

log_slave_updates=ON

log_bin=binlog

binlog_format=ROW
```

这些设置将 server 配置为使用唯一标识号 1，以启用全局事务标识符，并将复制元数据存储在系统表（而不是文件）中。 此外，它设置 server 打开二进制日志记录，使用基于行的格式并禁用二进制日志事件校验和。 有关详细信息，请参见第 19.7.2 节“使用限制”

组复制设置

确保 server 中 my.cnf 文件此时已按要求配置，且 server 按配置实例化复制基础结构。 以下部分是 server 组复制的配置。

```
transaction_write_set_extraction=XXHASH64

loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa
"


loose-group_replication_start_on_boot=off

loose-group_replication_local_address= "127.0.0.1:24901"

loose-group_replication_group_seeds=
"127.0.0.1:24901,127.0.0.1:24902,127.0.0.1:24903"

loose-group_replication_bootstrap_group= off
```

Note

如果在 server 启动时尚未加载组复制插件，上面 group_replication 变量使用的 loose- 前缀将指示 server 继续启动，

- 第 1 行指示 server 必须为每个事务收集写集合，并使用 XXHASH64 哈希算法将其编码为散列。
- 第 2 行告知插件，正在加入或创建的组要命名为“aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa”。
- 第 3 行指示插件在 server 启动时不自动启动组复制。
- 第 4 行告诉插件使用 IP 地址 127.0.0.1 或本地主机，端口 24901 用于接受来自组中其他成员的传入连接。

  Important

  server 在此端口上侦听组成员之间的连接。 此端口不能用于用户应用程序，它必须保留，用于在运行组复制时组的不同成员之间的通信。

  由 group_replication_local_address 配置的本地地址必须可供所有组成员访问。 例如，如果每个 server 实例位于不同的计算机上，则使用计算机的 IP 和端口，例如 10.0.0.1:33061。group_replication_local_address 的推荐端口为 33061，但在本教程中，我们使用在一台机器上运行的三个 server 实例，因此使用端口 24901 到 24903。
- 第 5 行告诉插件，当下面这些 server 需要加入组时，应该连接到这些主机和端口上访问他们。 这些就是种子成员，当此成员想要连接到组时使用，在申请加入时，server 先访问这些种子成员中的一个，然后它请求组重新配置以允许它加入组。 需要注意的是，此选项不需要列出组中的所有成员，而是当此 sever 需要加入该组时需要访问的 server 列表。
  启动组的 server 不使用此选项，因为它是初始 server，因此它负责引导组。 第二个加入的 server 向组中的唯一成员申请加入，然后组得以扩容。 第三个加入的 server 可以向这两个 server 中的任意一个申请加入，然后组再次扩容。 后续 server 在加入时重复此过程。

  Warning

  当多个 server 同时加入时，请确保它们已在组中的种子成员。 不要使用同时正在申请加入组的种子成员，因为他们可能在访问时尚未加入群组。
  首先启动引导成员并让它创建组， 然后使其成为正在加入的其余成员的种子成员。 这确保了当加入其余成员时已有一个组。
  不支持创建组并同时加入多个成员。 在操作竞争时，这种情况可能会发生，但是加入组的行为最终会出现错误或超时。
- 第 6 行指示插件是否自动引导组。

  Important

  此选项在任何时候只能在一个 server 实例上使用，通常是首次引导组时（或在整个组被崩溃然后恢复的情况下）。 如果您多次引导组，例如，当多个 server 实例设置了此选项，则它们可能会人为地造成脑裂的情况，其中存在两个具有相同名称的不同组。 在第一个 server 实例加入组后禁用此选项。

组中的所有 server 成员的配置都非常相似。您需要更改每个 server 的详细信息（例如 server_id，datadir，group_replication_local_address）。 这将在本教程的后面进行介绍。

19.2.1.3 用户凭据

组复制使用异步复制协议来实现分布式恢复，在将组成员加入组之前将其同步。 分布式恢复过程依赖于 group_replication_recovery 复制通道，该复制通道用于在组成员之间传输事务。 因此，您需要设置具有正确权限的复制用户，以便组复制可以直接建立成员到成员的恢复复制通道。启动服务器：

```
mysql-5.7/bin/mysqld --defaults-file=data/s1/s1.cnf
```

创建具有 REPLICATION-SLAVE 权限的 MySQL 用户。 此操作不应记录到二进制日志中，以避免将更改传递到其他 server 实例。。以下示例演示了创建用户 rpl_user 的过程，密码为 rpl_pass。配置服务器时需要使用正确的用户名和密码。 连接到 server s1 并执行以下语句：

```
mysql> SET SQL_LOG_BIN=0;
Query OK, 0 rows affected (0,00 sec)

mysql> CREATE USER rpl_user@'%';
Query OK, 0 rows affected (0,00 sec)

mysql> GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%' IDENTIFIED BY 'rpl_pass';
Query OK, 0 rows affected, 1 warning (0,00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0,00 sec)

mysql> SET SQL_LOG_BIN=1;
Query OK, 0 rows affected (0,00 sec)
```

用户进行上述配置后，需要使用 CHANGE MASTER TO 语句将 server 配置为，在下次需要从其他成员恢复其状态时，使用 group_replication_recovery 复制通道的给定凭据。 执行以下命令，将 rpl_user 和 rpl_pass 替换为创建用户时使用的值。

```
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='rpl_pass' \\
                    FOR CHANNEL 'group_replication_recovery';
Query OK, 0 rows affected, 2 warnings (0,01 sec)
```

分布式恢复是加入组的 server 执行的第一步。 如果未正确设置这些凭据，server 将无法执行恢复过程并获得与其他组成员同步，因此最终将无法加入组。 类似地，如果成员无法通过 server 的主机名正确识别其他成员，则恢复过程可能会失败。 建议运行 MySQL 的操作系统都正确地配置唯一主机名，使用 DNS 或本地设置。 可以在 performance_schema.replication_group_members 表的 Member_host 列中验证此主机名。 如

果多个组成员使用操作系统设置的默认主机名，则会出现有成员无法解析到正确的成员地址且无法加入组的情况。 在这种情况下，可以使用 report_host 来配置每个 server 唯一主机名。

19.2.1.4 启动组复制

配置并启动 server s1 后，安装组复制插件。 然后连接到 server 并执行以下命令：

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';
```

要检查插件是否已成功安装，请执行 SHOW PLUGINS 并检查输出。 它应该显示如下：

```
mysql> SHOW PLUGINS;

+----------------------------+----------+--------------------+----------
-----------+-------------+

| Name                       | Status   | Type               | Library
| License    |

+----------------------------+----------+--------------------+----------
-----------+-------------+

| binlog                     | ACTIVE   | STORAGE ENGINE     | NULL
| PROPRIETARY |



(...)


| group_replication          | ACTIVE   | GROUP REPLICATION  |
group_replication.so | PROPRIETARY |

+----------------------------+----------+--------------------+----------
-----------+-------------+
```

要启动组，请指示 server s1 引导组，然后启动组复制程序。 此引导应仅由单个 sever 独立完成，该 server 启动组并且只启动一次。 这就是为什么引导配置选项的值不保存在配置文件中的原因。 如果将其保存在配置文件中，则在重新启动时，server 会自动引导具有相同名称的第二个组。 这将导致两个不同的组具有相同的名称。 同样的道理适用于停止和重新启动插件，并且此选项设置为 ON。

```
SET GLOBAL group_replication_bootstrap_group=ON;
```

```
START GROUP_REPLICATION;

SET GLOBAL group_replication_bootstrap_group=OFF;
```

START GROUP_REPLICATION 语句返回后，组就已启动了。 您可以检查该组现在是否已创建并且其中已经有一个成员：

```
mysql> SELECT * FROM performance_schema.replication_group_members;

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE  |

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

| group_replication_applier | ce9be252-2b71-11e6-b8f4-00212844f856 |
myhost     |      24801 | ONLINE       |

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

1 row in set (0,00 sec)
```

此表中的信息确认了组中有一个成员并且具有唯一的标识符 ce9be252-2b71-11e6-b8f4-00212844f856，它是在线的并且在 myhost 侦听端口 24801 上的客户端连接。

为了演示 server 确实在一个组中，并且能够处理加载，创建一个表并向其中添加一些内容。

```
mysql> CREATE DATABASE test;

Query OK, 1 row affected (0,00 sec)



mysql> use test

Database changed

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL);
```

```
Query OK, 0 rows affected (0,00 sec)



mysql> INSERT INTO t1 VALUES (1, 'Luis');

Query OK, 1 row affected (0,01 sec)
```

检查表 t1 和二进制日志的内容。

```
mysql> SELECT * FROM t1;

+----+------+

| c1 | c2   |

+----+------+

|  1 | Luis |

+----+------+

1 row in set (0,00 sec)



mysql> SHOW BINLOG EVENTS;

+---------------+-----+---------------+-----------+-------------+------
-------------------------------------------------------------+

| Log_name      | Pos | Event_type    | Server_id | End_log_pos | Info
|

+---------------+-----+---------------+-----------+-------------+------
-------------------------------------------------------------+

| binlog.000001 |   4 | Format_desc   |         1 |         123 | Server ver:
5.7.17-gr080-log, Binlog ver: 4                      |

| binlog.000001 | 123 | Previous_gtids |         1 |         150 |
|

| binlog.000001 | 150 | Gtid          |         1 |         211 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1'  |
```

```
| binlog.000001 | 211 | Query         |        1 |        270 | BEGIN
|

| binlog.000001 | 270 | View_change    |        1 |        369 |
view_id=14724817264259180:1                                    |

| binlog.000001 | 369 | Query         |        1 |        434 | COMMIT
|

| binlog.000001 | 434 | Gtid          |        1 |        495 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:2'  |

| binlog.000001 | 495 | Query         |        1 |        585 | CREATE
DATABASE test                                          |

| binlog.000001 | 585 | Gtid          |        1 |        646 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:3'  |

| binlog.000001 | 646 | Query         |        1 |        770 | use `test`;
CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL) |

| binlog.000001 | 770 | Gtid          |        1 |        831 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:4'  |

| binlog.000001 | 831 | Query         |        1 |        899 | BEGIN
|

| binlog.000001 | 899 | Table_map     |        1 |        942 | table_id: 108
(test.t1)                                              |

| binlog.000001 | 942 | Write_rows    |        1 |        984 | table_id:
108 flags: STMT_END_F                                      |

| binlog.000001 | 984 | Xid           |        1 |       1011 | COMMIT /*
xid=38 */                                                 |

+---------------+-----+---------------+----------+-------------+------
-------------------------------------------------------------+

15 rows in set (0,00 sec)
```

如上所示，创建了数据库和表对象，并且将其对应的 DDL 语句写入二进制日志。 此外，数据被插入到表中并被写入二进制日志。 当组成员增长并且由于新成员尝试加入并变为在线而执行分布式恢复时，二进制日志条目的重要性将在以下部分中说明。

19.2.1.5 向组中添加实例

此时，组中有一个成员，已经有一些数据存在的 server s1。 此时就可以通过添加先前配置的其他两个 server 来扩展组了。

19.2.1.5.1 添加第二个实例

为了添加第二个实例 server s2，首先为它创建配置文件。该配置类似于用于 server s1 的配置，除了诸如数据目录的位置，s2 将要监听的端口或其 server_id 之类的事情之外。 这些不同的行已在下面的列表中突出显示。

```
[mysqld]



# server configuration

datadir=<full_path_to_data>/data/s2

basedir=<full_path_to_bin>/mysql-5.7/



port=24802

socket=<full_path_to_sock_dir>/s2.sock



#

# Replication configuration parameters

#

server_id=2

gtid_mode=ON

enforce_gtid_consistency=ON

master_info_repository=TABLE

relay_log_info_repository=TABLE
```

```
binlog_checksum=NONE

log_slave_updates=ON

log_bin=binlog

binlog_format=ROW



#

# Group Replication configuration

#

transaction_write_set_extraction=XXHASH64

loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa
"

loose-group_replication_start_on_boot=off

loose-group_replication_local_address= "127.0.0.1:24902"

loose-group_replication_group_seeds=
"127.0.0.1:24901,127.0.0.1:24902,127.0.0.1:24903"

loose-group_replication_bootstrap_group= off
```

与 server s1 的过程类似，在配置文件就位的情况下启动 server。

```
mysql-5.7/bin/mysqld --defaults-file=data/s2/s2.cnf
```

然后按如下所示配置恢复凭据。 由于用户在组中共享，该命令与设置 server s1 时使用的命令相同。 在 s2 上执行以下语句。

```
SET SQL_LOG_BIN=0;
CREATE USER rpl_user@'%';
GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%' IDENTIFIED BY 'rpl_pass';
SET SQL_LOG_BIN=1;
CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='rpl_pass' \\
       FOR CHANNEL 'group_replication_recovery';
```

安装组复制插件，并启动将 server 加入组的程序。 以下示例使用与部署 server s1 时相同的方式安装插件。

```
mysql> INSTALL PLUGIN group_replication SONAME 'group_replication.so';

Query OK, 0 rows affected (0,01 sec)
```

将 server s2 添加到组。

```
mysql> START GROUP_REPLICATION;

Query OK, 0 rows affected (44,88 sec)
```

与之前的步骤不同，这里与 s1 上执行的那些步骤有一个区别，就是不执行 SET GLOBAL group_replication_bootstrap_group = ON 的操作; 在启动组复制之前，因为该组已由 server s1 创建和引导。 此时，server s2 只需要添加到已经存在的组中。
Checking the `performance_schema.replication_group_members` table again shows that there are now two*ONLINE* servers in the group.
再次检查 performance_schema.replication_group_members 表，可以看出组中现在有两个 ONLINE 的 server。

```
mysql> SELECT * FROM performance_schema.replication_group_members;

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE  |

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

| group_replication_applier | 395409e1-6dfa-11e6-970b-00212844f856 |
myhost      |      24801 | ONLINE      |

| group_replication_applier | ac39f1e6-6dfa-11e6-a69d-00212844f856 |
myhost      |      24802 | ONLINE      |

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

2 rows in set (0,00 sec)
```

由于 server s2 也被标记为 ONLINE，它必须已经与 server s1 同步。 按照如下方式验证它确实与 server s1 同步。

```
mysql> SHOW DATABASES LIKE 'test';

+-----------------+

| Database (test) |

+-----------------+

| test            |

+-----------------+

1 row in set (0,00 sec)



mysql> SELECT * FROM test.t1;

+----+------+

| c1 | c2   |

+----+------+

|  1 | Luis |

+----+------+

1 row in set (0,00 sec)



mysql> SHOW BINLOG EVENTS;

+---------------+------+----------------+-----------+-------------+-----
-------------------------------------------------------------------+

| Log_name      | Pos  | Event_type     | Server_id | End_log_pos | Info
|

+---------------+------+----------------+-----------+-------------+-----
-------------------------------------------------------------------+
```

```
| binlog.000001 |   4 | Format_desc    |         2 |         123 | Server ver:
5.7.17-log, Binlog ver: 4                                          |

| binlog.000001 |  123 | Previous_gtids |         2 |         150 |
|

| binlog.000001 |  150 | Gtid           |         1 |         211 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1'  |

| binlog.000001 |  211 | Query          |         1 |         270 | BEGIN
|

| binlog.000001 |  270 | View_change    |         1 |         369 |
view_id=14724832985483517:1                                        |

| binlog.000001 |  369 | Query          |         1 |         434 | COMMIT
|

| binlog.000001 |  434 | Gtid           |         1 |         495 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:2'  |

| binlog.000001 |  495 | Query          |         1 |         585 | CREATE
DATABASE test                                                  |

| binlog.000001 |  585 | Gtid           |         1 |         646 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:3'  |

| binlog.000001 |  646 | Query          |         1 |         770 | use `test`;
CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL) |

| binlog.000001 |  770 | Gtid           |         1 |         831 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:4'  |

| binlog.000001 |  831 | Query          |         1 |         890 | BEGIN
|

| binlog.000001 |  890 | Table_map      |         1 |         933 | table_id:
108 (test.t1)                                                  |

| binlog.000001 |  933 | Write_rows     |         1 |         975 | table_id:
108 flags: STMT_END_F                                            |

| binlog.000001 |  975 | Xid            |         1 |        1002 | COMMIT /*
xid=30 */                                                      |
```

```
| binlog.000001 | 1002 | Gtid          |        1 |       1063 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:5'  |

| binlog.000001 | 1063 | Query         |        1 |       1122 | BEGIN
|

| binlog.000001 | 1122 | View_change    |         1 |       1261 |
view_id=14724832985483517:2                                    |

| binlog.000001 | 1261 | Query         |        1 |       1326 | COMMIT
|

+---------------+------+----------------+-----------+-------------+-----
-------------------------------------------------------------------+

19 rows in set (0,00 sec)
```

如上所示，第二个 server 已添加到组中，并且已自动从 server s1 复制了更改。 按照分布式恢复过程，这意味着加入组之后并且在即将被声明在线之前，server s2 自动地连接到 server s1 并且从其获取丢失的数据。 换句话说，它从 s1 的二进制日志中复制了它在加入组的时间节点之前缺少的事务。

19.2.1.5.2 添加其他实例

向组添加其他实例与添加第二个server基本上是相同的步骤，除了配置必须更改为对应的server。总结所需的命令如下：

1.创建配置文件

```
[mysqld]



# server configuration

datadir=<full_path_to_data>/data/s3

basedir=<full_path_to_bin>/mysql-5.7/



port=24803

socket=<full_path_to_sock_dir>/s3.sock
```

```
#
# Replication configuration parameters
#
server_id=3
gtid_mode=ON
enforce_gtid_consistency=ON
master_info_repository=TABLE
relay_log_info_repository=TABLE
binlog_checksum=NONE
log_slave_updates=ON
log_bin=binlog
binlog_format=ROW

#
# Group Replication configuration
#
transaction_write_set_extraction=XXHASH64
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa
"
loose-group_replication_start_on_boot=off
loose-group_replication_local_address= "127.0.0.1:24903"
loose-group_replication_group_seeds=
"127.0.0.1:24901,127.0.0.1:24902,127.0.0.1:24903"
```

```
loose-group_replication_bootstrap_group= off
```

2. 启动 server

```
mysql-5.7/bin/mysqld --defaults-file=data/s3/s3.cnf
```

3. 配置 group_replication_recovery 通道的恢复凭据

```
SET SQL_LOG_BIN=0;

CREATE USER rpl_user@'%';

GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%' IDENTIFIED BY 'rpl_pass';

FLUSH PRIVILEGES;

SET SQL_LOG_BIN=1;

CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='rpl_pass'  \\

FOR CHANNEL 'group_replication_recovery';
```

4. 安装组复制插件并启动。

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';

START GROUP_REPLICATION;
```

此时，server s3 被引导并且正在运行，并且已经加入组且已与组中的其他 server 成员同步。 访问 performance_schema.replication_group_members 表再次确认真实情况确实如此。

```
mysql> SELECT * FROM performance_schema.replication_group_members;

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE  |

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

| group_replication_applier | 395409e1-6dfa-11e6-970b-00212844f856 |
myhost     |      24801 | ONLINE       |
```

```
| group_replication_applier | 7eb217ff-6df3-11e6-966c-00212844f856 |
myhost     |      24803 | ONLINE       |

| group_replication_applier | ac39f1e6-6dfa-11e6-a69d-00212844f856 |
myhost     |      24802 | ONLINE       |

+---------------------------+--------------------------------------+----
---------+-------------+---------------+

3 rows in set (0,00 sec)
```

在 server s2 或 server s1 上发出此相同的查询会产生相同的结果。 此外，您可以验证 server s3 已经同步：

```
mysql> SHOW DATABASES LIKE 'test';

+-----------------+

| Database (test) |

+-----------------+

| test            |

+-----------------+

1 row in set (0,00 sec)


mysql> SELECT * FROM test.t1;

+----+------+

| c1 | c2   |

+----+------+

|  1 | Luis |

+----+------+

1 row in set (0,00 sec)
```

```
mysql>  SHOW BINLOG EVENTS;

+---------------+------+---------------+-----------+-------------+-----
-------------------------------------------------------------------+

| Log_name      | Pos  | Event_type    | Server_id | End_log_pos | Info
|

+---------------+------+---------------+-----------+-------------+-----
-------------------------------------------------------------------+

| binlog.000001 |    4 | Format_desc   |         3 |         123 | Server ver:
5.7.17-log, Binlog ver: 4                          |

| binlog.000001 |  123 | Previous_gtids |         3 |         150 |
|

| binlog.000001 |  150 | Gtid          |         1 |         211 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1'  |

| binlog.000001 |  211 | Query         |         1 |         270 | BEGIN
|

| binlog.000001 |  270 | View_change   |         1 |         369 |
view_id=14724832985483517:1                                    |

| binlog.000001 |  369 | Query         |         1 |         434 | COMMIT
|

| binlog.000001 |  434 | Gtid          |         1 |         495 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:2'  |

| binlog.000001 |  495 | Query         |         1 |         585 | CREATE
DATABASE test                                          |

| binlog.000001 |  585 | Gtid          |         1 |         646 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:3'  |

| binlog.000001 |  646 | Query         |         1 |         770 | use `test`;
CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL) |

| binlog.000001 |  770 | Gtid          |         1 |         831 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:4'  |
```

```
| binlog.000001 |  831 | Query         |        1 |        890 | BEGIN
|

| binlog.000001 |  890 | Table_map     |        1 |        933 | table_id:
108 (test.t1)                                    |

| binlog.000001 |  933 | Write_rows    |        1 |        975 | table_id:
108 flags: STMT_END_F                                |

| binlog.000001 |  975 | Xid           |        1 |       1002 | COMMIT /*
xid=29 */                                        |

| binlog.000001 | 1002 | Gtid          |        1 |       1063 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:5'  |

| binlog.000001 | 1063 | Query         |        1 |       1122 | BEGIN
|

| binlog.000001 | 1122 | View_change   |        1 |       1261 |
view_id=14724832985483517:2                             |

| binlog.000001 | 1261 | Query         |        1 |       1326 | COMMIT
|

| binlog.000001 | 1326 | Gtid          |        1 |       1387 | SET
@@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:6'  |

| binlog.000001 | 1387 | Query         |        1 |       1446 | BEGIN
|

| binlog.000001 | 1446 | View_change   |        1 |       1585 |
view_id=14724832985483517:3                             |

| binlog.000001 | 1585 | Query         |        1 |       1650 | COMMIT
|

+---------------+------+---------------+----------+-------------+-----
-------------------------------------------------------------+

23 rows in set (0,00 sec)
```

19.3 监控组复制

假设 MySQL 已经在启用了性能模式的情况下编译，使用 Perfomance Schema 表监控组复制。组复制添加以下两个新的 P_S 表：

- `performance_schema.replication_group_member_stats`
- `performance_schema.replication_group_members`

这些现有的 P_S 复制表也显示有关组复制的信息。

- `performance_schema.replication_connection_status`
- `performance_schema.replication_applier_status`

由组复制插件创建的复制通道名为：

- `group_replication_recovery` - This channel is used for the replication changes that are related to the distributed recovery phase.
- `group_replication_applier` - This channel is used for the incoming changes from the group. This is the channel used to apply transactions coming directly from the group.

以下部分描述了每个表中可用的信息。

19.3.1 Replication_group_member_stats

复制组中的每个成员都会验证并应用该组提交的事务。 有关验证和应用程序的统计信息对于了解申请队列增长情况、触发了多少冲突、检查了多少事务、哪些事务已被所有成员提交等等非常有用。 表 performance_schema.replication_group_member_stats 提供与认证过程相关的以下信息。

表 19.1 replication_group_member_stats

Field	描述
Channel_name	组复制通道的名称
Member_id	此值为我们当前连接到的 server 成员的 UUID。组中的每个成员具有不同的值。因为它对每个成员是唯一的，所以它也成为了一个关键字。
Count_Transactions_in_queue	队列中等待冲突检测检查的事务数。冲突检查通过后，他们排队等待应用。
Count_transactions_checked	表示已进行过冲突检查的事务数。
Count_conflicts_detected	表示未通过冲突检测检查的事务数。

Field	描述
Count_transactions_validating	表示冲突检测数据库的当前大小（每个事务经过验证的数据库）。
Transactions_committed_all_members	表示已在当前视图的所有成员上成功提交的事务。 此值以固定的时间间隔更新。
Last_conflict_free_transaction	显示最后一个经检查无冲突的事务标识符。

这些字段对于监控组中的成员的性能很重要。 例如，假设组的成员之一出现延迟，并且不能与该组的其他成员同步。 在这种情况下，您可能会在队列中看到大量的事务。 基于此信息，您可以决定从组中删除成员或延迟组中其他成员的事务处理，从而减少排队的事务的数量。 此信息还可以帮助您决定如何调整组复制插件的流控制。

19.3.2 Replication_group_members

该表用于监控在当前视图中的不同 server 实例的状态，或者换句话说，是该组的一部分，用于组员服务追踪。

表 19.2 replication_group_members

Field	描述
Channel_name	组复制通道的名称
Member_id	server 成员的 UUID
Member_host	组成员的网络地址
Member_port	侦听此成员的 MySQL 连接端口
Member_state	组成员的状态(判断他们是 ONLINE, RECOVERING, OFFLINE 或者 UNREACHABLE 中的某一个)

有关 Member_host 值的详细信息及其对分布式恢复过程的影响，请参见第 19.2.1.3 节“用户凭据”。

19.3.3 Replication_connection_status

连接到组时，此表中的某些字段显示有关组复制的信息。 例如，已从组中接收并在应用队列（中继日志）中排队的事务。

表 19.3 replication_connection_status

Field	Description
Channel_name	组复制通道的名称
Group_name	表示组名称的值。 它始终是有效的 UUID。
Source_UUID	表示组的标识符。 它类似于组名称，并被用作组复制期间生成的所有事务的 UUID。
Service_state	显示某个成员是否是组的一部分。 服务状态的可能值为{ON，OFF 和 CONNECTING}；
Received_transaction_set	此 GTID 集合中的事务已由该组的此成员接收。

19.3.4 Replication_applier_status

可以使用常规 replication_applier_status 表观察组复制相关的通道和线程的状态。 如果有许多不同的工作线程在应用事务，那么工作表也可以用来监控每个工作线程正在做什么。

表 19.4 replication_applier_status

Field	描述
Channel_name	组复制通道的名称。
Service_state	显示应用服务是开启还是关闭。
Remaining_delay	显示是否配置了应用延迟。
Count_transactions_retries	应用事务的重试次数。
Received_transaction_set	此 GTID 集合中的事务已由该组的此成员接收。

19.3.5 组复制中的 server 状态

每当视图更改时，表 replication_group_members 就会更新，例如，当组的配置动态更改时。 在此基础上，server 成员之间交换他们的一些元数据以保持同步并继续协作。
组中的 server 实例可以处于多种状态。如果 server 都正常通信，则所有 server 都报告相同的状态。 但是，如果存在网络分隔，或者组成员离开组，则可能报告不同的信息，这取决于查询了哪个 server。 要注意的是，如果某个组成员已经离开组，那么显然它不能报告关于其他 server 状态的最新信息。 如果发生网络分隔，如果超出仲裁数量的 server 都断开了，那么 server 之

间将不能相互协作。因此，他们无法得知不同 server 成员的状态。因此，他们会报告一些 server 不可访问，而不是猜测他们的状态。

表 19.5 Server State

Field	描述	Group Synchronized
ONLINE	该成员可以作为一个具有所有功能的组成员，这意味着客户端可以连接并开始执行事务。	Yes
RECOVERING	该成员正在成为该组的有效成员，并且正处于恢复过程中，从数据源节点（数据源节点）接收状态信息。	No
OFFLINE	插件已加载，但成员不属于任何组。	No
ERROR	本地成员的状态。 只要恢复阶段或应用更改时出现错误，server 就会进入此状态。	No
UNREACHABLE	每当本地故障检测器怀疑某个给定的 server 可能由于已经崩溃或被意外地断开而不可访问时，server 的状态显示为“UNREACHABLE”。	No

需要注意的是，组复制不是同步复制，但最终是同步的。 更确切地说，事务以相同的顺序传递给所有组成员，但是它们的执行不同步，这意味着在接受事务被提交之后，每个成员以其自己的速度提交。

19.4 组复制操作

本节介绍部署组复制的不同模式，说明管理组的常见操作，并提供有关如何调整组的信息。

19.4.1 以多主模式或单主模式部署

组复制在以下不同模式下运行：

- 单主模式
- 多主模式

默认模式为单主模式。组的不同成员不能部署在不同的模式下，例如一个配置为多主模式，而另一个为单主模式。 要在模式之间切换，需要使用不同配置重新启动组而不是单个 server。 无论部署模式如何，组复制不处理客户端 fail-over，必须由应用程序本身、连接器或中间件框架（如代理或路由器）处理。

在多主模式下部署时，将检查语句以确保它们与模式兼容。 在以多主模式部署组复制时需要进行以下检查：

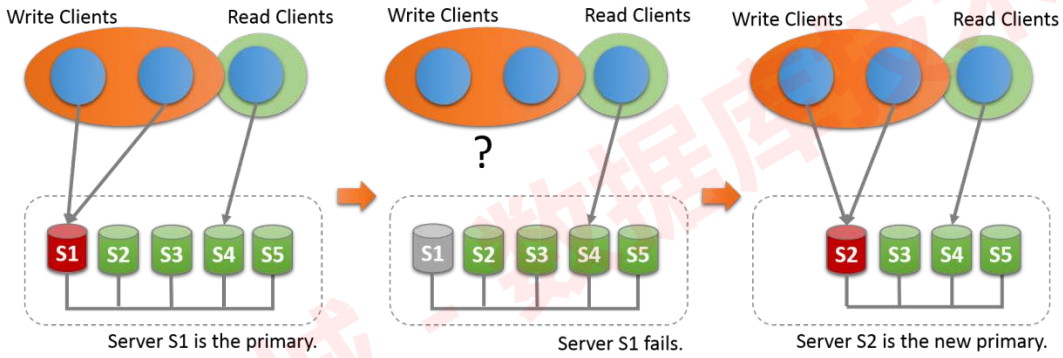
- 如果事务在 SERIALIZABLE 隔离级别下执行，则在与组同步时，它的提交会失败。
- 如果事务对具有外键约束的表进行操作，则事务在与组同步时无法提交。

可以通过将 optiongroup_replication_enforce_update_everywhere_checks 设置为 FALSE 来禁用这些检查。 在单主机模式下部署时，此选项必须设置为 FALSE。

19.4.1.1 单主模式

在此模式下，组有一个设置为读写模式的单主 server。 组中的所有其他成员被自动设置为只读模式（超级只读模式）。主服务器通常是用于引导组的第一个 server，所有其他加入的 server 自动从主服务器同步并设置为只读。

图 19.5 选取新主



在单主机模式下，将禁用在多主机模式下部署的某些检查，因为系统会强制在组中每次只有一个写入 server。 例如，在单主模式下允许对具有外键的表进行更改，而在多主模式下不允许。 在主服务器故障时，自动选主机制选择下一个主服务器。 通过按字典顺序（使用其 UUID）来排序剩余的 server 成员并选择列表中的第一个成员来作为下一个主服务器。

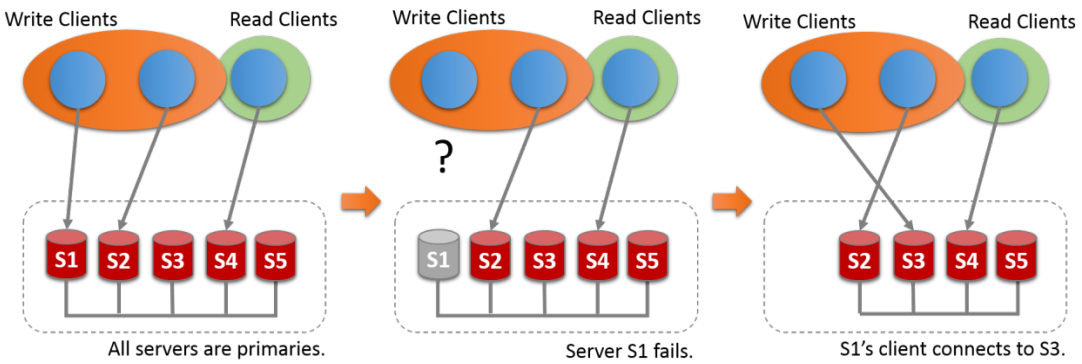
如果主服务器从组中移除，则启动主节点选择程序，然后从组中的其余 server 成员中选择新的主节点。通过查看新视图，按照词典顺序将 server 的 UUID 进行排序并选择第一个作为主节点。选择了新的主节点后，它将自动设置为只读，其他辅助节点仍然为辅助节点，因此也是只读。

在将客户端应用程序重新路由到它之前，等待新的主节点完成所有的数据同步是一个很好的选择。

19.4.1.2 多主模式

在多主模式下，没有单主模式的概念。 没有必要使用选主程序，因为不同 server 成员之间没有什么特殊的差异。

图 19.6 客户端故障转移



加入组时，所有 server 成员都将被设置为读写模式。

19.4.1.3 查找主节点

以下示例显示了在单主模式下部署时，如何确定当前哪个 server 成员是主服务器。

```
mysql> SELECT VARIABLE_VALUE FROM performance_schema.global_status WHERE
VARIABLE_NAME= 'group_replication_primary_member';
+--------------------------------------+
| VARIABLE_VALUE                       |
+--------------------------------------+
| 69e1a3b8-8397-11e6-8e67-bf68cbc061a4 |
+--------------------------------------+
1 row in set (0,00 sec)
```

19.4.2 调整恢复

每当新成员加入复制组时，它将连接到合适的数据源节点数据源节点来获取它所缺失的数据，直到它的状态为 OLINE。 组复制中的这个关键组件是容错和可配置的。 以下部分说明恢复如何进行以及如何调整设置。

数据源节点数据源节点选择

从该组中的现有在线成员中随机选择数据源节点数据源节点。当多个成员进入组时，同一 server 几乎不会被重复选择。

如果访问所选数据源节点数据源节点失败，则自动连接到新的候选数据源节点数据源节点。 一旦达到连接重试限制，恢复过程将终止并报错。

Note

数据源节点数据源节点从当前视图中的在线成员列表中随机选择的。

增强型自动数据源节点数据源节点切换

总体来说，完整恢复的另一个关键点是确保它能够应对故障。 因此，组复制提供了可靠的错误检测机制。 在早期版本的组复制中，当访问一个数据源节点数据源节点时，恢复程序只能检测出认证问题或一些其他问题导致的连接错误。 应对这种场景的策略是切换到新的数据源节点数据源节点，因此会对别的成员进行新的连接尝试。

这种模式被扩展使用到其他故障场景：

- ***已清除的数据*** - 如果所选数据源节点数据源节点上恢复过程所需的某些数据已被清除，则会出现错误。 恢复程序检测到此错误并会选择新的数据源节点数据源节点。
- ***重复数据*** – 如果新加入的 server 已经包含一些与选定数据源节点数据源节点提供的数据相冲突的数据，那么在恢复期间会发生错误。 这可能是由于新加入的 server 中存在一些错误的事务。

  有人可能认为恢复过程应该失败退出，而不是切换到另一个数据源节点数据源节点，但在异构集群中有一些其他成员共享冲突事务，而另一些没有。 因为这个原因，错误发生时，恢复程序将从组中选择另一个数据源节点数据源节点。
- ***其他错误*** - 如果任何恢复线程失败（接收或应用线程失败），则会出现错误，恢复程序切换到新的数据源节点数据源节点。

Note

在一些持续故障甚至短暂故障的情况下，恢复程序自动重试连接到相同或新的数据源节点数据源节点。

数据源节点数据源节点连接重试

恢复的数据传输依赖于二进制日志和现有 MySQL 复制框架，因此一些短暂错误可能会导致接收或应用线程错误。 在这种情况下，数据源节点切换过程具有重试功能，类似于在常规复制中的重试功能。

尝试次数

当要加入的 server 尝试从数据源节点池连接到某个数据源节点时，所尝试次数为 10.这通过 group_replication_recovery_retry_count 这个插件变量进行配置。 以下命令将连接到数据源节点的最大尝试次数设置为 10。

```
SET GLOBAL group_replication_recovery_retry_count= 10;
```

需要注意的是，这是要加入的 server 会尝试连接到每个合适的数据源节点的全局次数。

睡眠程序

group_replication_recovery_reconnect_interval 这个插件变量定义了恢复进程在尝试连接到数据源节点时应休眠多长时间。 此变量的默认设置为 60 秒，您可以动态更改此值。 以下命令设置尝试连接到恢复数据源节点的重试间隔时间为 120 秒。

```
SET GLOBAL group_replication_recovery_reconnect_interval= 120;
```

要注意的是，恢复程序不会在每次尝试连接数据源节点后休眠。 由于要加入的 server 会尝试连接到不同的 server，而不是不断地尝试连接到同一个 server，我们可以假设影响 server A 的问题可能不影响 server B.因此，恢复程序仅当它已经对所有可能的数据源节点进行尝试无果后才会停止。要加入的 server 尝试连接到组中的所有可能的数据源节点后，恢复程序会在由 group_replication_recovery_reconnect_interval 配置的秒数后休眠：

19.4.3 网络分隔

当复制的网络环境发生变化时，组内需要达成共识。 这是常规事务的处理，而且对于组成员更改和保持组一致的某些内部消息传递也是必需的。 组内共识需要大多数组成员就给定的决定达成一致。 当大多数组成员丢失时，由于无法确保多数或仲裁成员是否存活，组将无法继续运行而停止。

当有多个节点意外故障时，仲裁可能会丢失，导致大多数 server 成员突然从组中被移除。例如，在一个 5 个 server 组成的组中，如果其中 3 个 server 突然没有消息，也就是大多数 server 成员被破坏，因此不能实现仲裁。 事实上，剩下的两个 server 不能分辨其他 3 个 server 是否崩溃或网络分隔了这 2 个 server，因此无法自动重新配置组。

另一方面，如果有 server 成员主动退出组，那它们会告知组应该重新进行配置。 实际上，这意味着一个将要离开的 server 成员会告诉其余的 server 成员它将要离开。 这也意味着其他成员可以正确地重新配置组，保持组成员关系的一致性，并重新计算仲裁成员数。 例如，上述的场景中存在 5 个 server 成员，3 个 server 要离开，如果 3 个要离开的 server 成员告知组他们一个接一个地离开，则组成员能够将组的总数从 5 调整到 2，与此同时确保仲裁成员数。

Note

仲裁成员数的丢失本身就是不良规划造成的。 要根据预期故障数（无论它们是否持续地同时发生，还是偶尔发生）计划组的大小。

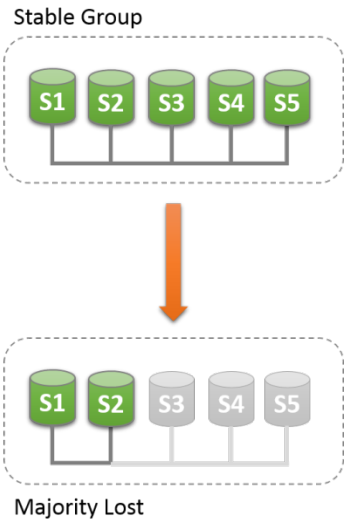
以下部分将介绍，如果系统被分隔了导致组内仲裁成员数无法自动达成时，应该怎么处理。

19.4.3.1 检测分区

performance schema 下的 replication_group_members 表，从每个 server 的角度显示当前视图中每个 server 的状态。 大多数情况下系统不会发生分隔，因此该表显示的信息对于组中的所有 server 成员都是一致的。 换句话说，此表上的每个 server 的状态都由当前视图中的所有 server

成员达成决议的。 但是，如果存在网络分隔，并且仲裁丢失，则表对于访问不到的那些 server 显示状态 UNREACHABLE。 此信息由组复制中内置的本地故障检测器导出。

图 19.7 仲裁丢失



为了理解这种类型的网络分隔，下面的部分描述了最初有 5 个 server 成员的组正确工作的场景，以及只有 2 个 server 成员在线时该组发生的改变。 这种场景在图中进行了描述。

因此，我们假设有一个组中有这 5 个 server：

- server s1，成员标识符为 199b2df7-4aaf-11e6-bb16-28b2bd168d07
- server s2，成员标识符为 199bb88e-4aaf-11e6-babe-28b2bd168d07
- server s3，成员标识符为 1999b9fb-4aaf-11e6-bb54-28b2bd168d07
- server s4，成员标识符为 19ab72fc-4aaf-11e6-bb51-28b2bd168d07
- server s5，成员标识符为 19b33846-4aaf-11e6-ba81-28b2bd168d07

最初，组可以正常运行，server 之间可以畅通地相互通信。

您可以通过登录到 s1 并查看 performance schema 下的 replication_group_members 表来验证这一点。 例如：

```
mysql> SELECT * FROM performance_schema.replication_group_members;

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| group_replication_applier | 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 |
127.0.0.1   |      13002 | ONLINE       |
```

```
| group_replication_applier | 199b2df7-4aaf-11e6-bb16-28b2bd168d07 |
127.0.0.1   |      13001 | ONLINE      |

| group_replication_applier | 199bb88e-4aaf-11e6-babe-28b2bd168d07 |
127.0.0.1   |      13000 | ONLINE      |

| group_replication_applier | 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 |
127.0.0.1   |      13003 | ONLINE      |

| group_replication_applier | 19b33846-4aaf-11e6-ba81-28b2bd168d07 |
127.0.0.1   |      13004 | ONLINE      |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

5 rows in set (0,00 sec)
```

但是，不久之后发生灾难性的故障，server s3，s4 和 s5 意外停止。 几秒钟后，再次查看在 s1 的 replication_group_members 表显示它仍然在线，但其他几个成员不在线。 事实上，如下所示，它们被标记为 UNREACHABLE。 此外，系统不能重新进行自身配置改变成员关系，因为大多数成员已经丢失。

```
mysql> SELECT * FROM performance_schema.replication_group_members;

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| group_replication_applier | 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 |
127.0.0.1   |      13002 | UNREACHABLE  |

| group_replication_applier | 199b2df7-4aaf-11e6-bb16-28b2bd168d07 |
127.0.0.1   |      13001 | ONLINE      |

| group_replication_applier | 199bb88e-4aaf-11e6-babe-28b2bd168d07 |
127.0.0.1   |      13000 | ONLINE      |

| group_replication_applier | 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 |
127.0.0.1   |      13003 | UNREACHABLE  |
```

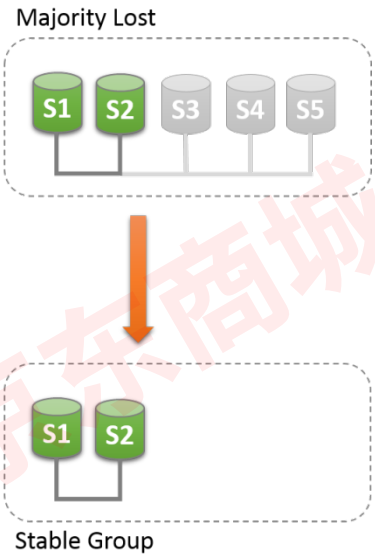
```
| group_replication_applier | 19b33846-4aaf-11e6-ba81-28b2bd168d07 |
127.0.0.1   |      13004 | UNREACHABLE  |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

5 rows in set (0,00 sec)
```

该表显示，s1 现在处于一个在没有外部干预的情况下无法继续运行的组中，因为大多数 server 不可访问。 在这种特殊情况下，为了让系统继续进行，组成员关系列表需要重置，这已经在本节中讲解过。 或者，您也可以选择停止 s1 和 s2 上的组复制（或完全停止 s1 和 s2），找出 s3，s4 和 s5 停止的原因，然后重新启动组复制（或服务器）。

19.4.3.2 解除分隔

组复制使您能够通过强制执行特定配置来重置组成员关系列表。 例如，在上面的情况中，其中 s1 和 s2 是仅有的在线 server，您可以强制地为 s1 和 s2 配置一致的组成员关系。 这就需要检查有关 s1 和 s2 的一些信息，然后使用 group_replication_force_members 变量。

图 19.8 强制配置组成员关系



假设 s1 和 s2 是组中仅有的 server 成员的情况，server s3，s4 和 s5 意外地离开了组。 要使 s1 和 s2 继续运行，您就可能要强制地配置仅包含 s1 和 s2 的成员关系。

Warning

此过程使用 group_replication_force_members，并应被视为最后的补救措施。使用 group_replication_force_members 应该非常小心，它仅用来重置仲裁成员数丢失的情况。 如果被滥用，它可能导致人为的脑裂情况或阻塞整个系统运行。

试想一下，系统被阻塞，并且当前配置如下（这是 s1 上的本地故障检测器所检测到的）：

```
mysql> SELECT * FROM performance_schema.replication_group_members;

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| group_replication_applier | 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 |
127.0.0.1   |      13002 | UNREACHABLE  |

| group_replication_applier | 199b2df7-4aaf-11e6-bb16-28b2bd168d07 |
127.0.0.1   |      13001 | ONLINE       |

| group_replication_applier | 199bb88e-4aaf-11e6-babe-28b2bd168d07 |
127.0.0.1   |      13000 | ONLINE       |

| group_replication_applier | 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 |
127.0.0.1   |      13003 | UNREACHABLE  |

| group_replication_applier | 19b33846-4aaf-11e6-ba81-28b2bd168d07 |
127.0.0.1   |      13004 | UNREACHABLE  |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

5 rows in set (0,00 sec)
```

首先要做的是检查 s1 和 s2 的对等地址（组通信标识符）是什么。 登录到 s1 和 s2 并获取该信息，如下所示。

```
mysql> SELECT @@group_replication_local_address;

+-------------------------------------+

| @@group_replication_local_address |

+-------------------------------------+

| 127.0.0.1:10000                |

+-------------------------------------+
```

```
1 row in set (0,00 sec)
```

然后登录到 s2 并做同样的操作：

```
mysql> SELECT @@group_replication_local_address;

+-----------------------------------+

| @@group_replication_local_address |

+-----------------------------------+

| 127.0.0.1:10001                   |

+-----------------------------------+

1 row in set (0,00 sec)
```

知道 s1（127.0.0.1:10000）和 s2（127.0.0.1:10001）的组通信地址后，就可以在其中一台服务器上配置新的成员关系，从而覆盖现有的仲裁成员数丢失情况下的成员关系。在 s1 上进行如下操作：

```
mysql> SET GLOBAL
group_replication_force_members="127.0.0.1:10000,127.0.0.1:10001";

Query OK, 0 rows affected (7,13 sec)
```

这会强制地使用不同的配置来解锁组。 检查 s1 和 s2 上的 replication_group_members 表以验证更改后的组成员关系。 首先检查 s1。

```
mysql> select * from performance_schema.replication_group_members;

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| group_replication_applier | b5ffe505-4ab6-11e6-b04b-28b2bd168d07 |
127.0.0.1   |      13000 | ONLINE       |
```

```
| group_replication_applier | b60907e7-4ab6-11e6-afb7-28b2bd168d07 |
127.0.0.1   |      13001 | ONLINE      |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

2 rows in set (0,00 sec)
```

然后检查 s2.

```
mysql> select * from performance_schema.replication_group_members;

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST
| MEMBER_PORT | MEMBER_STATE |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

| group_replication_applier | b5ffe505-4ab6-11e6-b04b-28b2bd168d07 |
127.0.0.1   |      13000 | ONLINE      |

| group_replication_applier | b60907e7-4ab6-11e6-afb7-28b2bd168d07 |
127.0.0.1   |      13001 | ONLINE      |

+---------------------------+--------------------------------------+----
---------+-------------+--------------+

2 rows in set (0,00 sec)
```

当进行强制配置成员关系时，需要确保所有将被强制从组中移除的server都已经停止工作了。在上面描述的情形中，如果 s3，s4 和 s5 不是真正不可访问的，而是在线的，则它们可能已经形成了它们自己的功能分隔区（它们是 5 个 server 中的 3 个，因此他们是大多数）。 在这种情况下，强制配置由 s1 和 s2 组成的成员列表会造成人为的脑裂情况。 因此，在强制地配置成员关系前，需要确保将要移除的 server 确实已经关闭，如果没有关闭，则在继续配置之前执行关闭操作是非常重要的。

19.5 组复制安全

本节介绍如何通过保护组成员之间的连接，或者使用地址白名单来确保一个组是安全有效的工作。

19.5.1 IP 地址白名单

组复制插件具有一个配置选项，用于确定哪些主机可以接入的组通信连接。 此选项称为 group_replication_ip_whitelist。 如果在 server s1 上设置此选项，则当 server s2 正在建立与 s1 的组通信连接时，s1 在接受 s2 的连接请求之前，需要首先检查白名单。 如果 s2 在白名单中，则 s1 接受连接，否则 s1 拒绝 s2 的连接尝试。

Note

默认情况下，如果未明确说明，白名单将自动设置为一个该 server 的私有网络接口地址。

如果未配置任何白名单，则 server 会自动将白名单设置为他的私有网络接口地址。 这意味着，即使 server 具有公共 IP 上的接口，默认也不会允许来自外部主机的连接。

每当 IP 白名单设置为 AUTOMATIC 时，在这种情况下错误日志中会显示如下信息：

```
2016-07-07T06:40:49.320686Z 4 [Note] Plugin group_replication reported:
'Added automatically \\

IP ranges 10.120.40.237/18,10.178.59.44/22,127.0.0.1/8 to the whitelist'
```

您可以通过手动设置允许进行组通信连接的 IP 地址列表，来进一步提高组的安全性。 可以使用 CIDR 标识法或简单的 IP 地址来制定列表。 使用逗号分隔各个条目。 例如：

```
mysql> STOP GROUP_REPLICATION;

mysql> SET GLOBAL
group_replication_ip_whitelist="10.120.40.237/18,10.178.59.44/22,127.0.0
.1/8";

mysql> START GROUP_REPLICATION;
```

Note

本地主机 IP 地址（127.0.0.1）始终添加到白名单中。 如果没有明确说明，它是隐式和自动添加的。

19.5.2 安全套接字层的支持（SSL）

MySQL 组复制支持 MySQL Server 的 OpenSSL 和 YaSSL 协议。

组通信连接以及恢复程序的连接使用 SSL 进行保护。 以下部分说明如何配置连接。

为配置恢复 SSL

恢复是通过常规异步复制连接实现的。一旦数据源节点选择好，新加入的 server 通过异步复制与自动其建立连接。

但是，请求 SSL 连接的用户必须在要加入的 server 连接到数据源节点之前被创建。 通常，这是在给要加入组的 server 指定数据源节点时设置的。

```
数据源节点> SET SQL_LOG_BIN=0;

数据源节点> CREATE USER 'rec_ssl_user'@'%' REQUIRE SSL;

数据源节点> GRANT replication slave ON *.* TO 'rec_ssl_user'@'%';

数据源节点> SET SQL_LOG_BIN=1;
```

假设组中的所有 server 成员都已经有一个设置好的复制用户用于使用 SSL，您可以将 server 配置为在连接到数据源节点时使用这些凭证。 这是通过为组复制插件配置的 SSL 选项的值来配置的。

```
new_member> SET GLOBAL group_replication_recovery_use_ssl=1;

new_member> SET GLOBAL group_replication_recovery_ssl_ca=
'.../cacert.pem';

new_member> SET GLOBAL group_replication_recovery_ssl_cert=
'.../client-cert.pem';

new_member> SET GLOBAL group_replication_recovery_ssl_key=
'.../client-key.pem';
```

并且要通过将恢复通道配置为“需要 SSL 连接的用户凭证”来实现。

```
new_member> CHANGE MASTER TO MASTER_USER="rec_ssl_user" FOR CHANNEL
"group_replication_recovery";

new_member> START GROUP_REPLICATION;
```

配置组通信相关的 SSL

可以使用安全套接字在组成员之间建立通信。 此配置取决于 server 的 SSL 配置。 因此，如果 server 配置了 SSL，则组复制插件也配置了 SSL。 有关配置 server 的 SSL 选项的详细信息，请参见第 7.4.5 节“安全连接的命令选项”。 配置组复制的选项如下表所示。

表 19.6 SSL 选项

Server 配置	插件配置说明
ssl_key	密钥文件的路径。 用作客户端和服务器认证
ssl_cert	证书文件的路径。 用作客户端和服务器证书。
ssl_ca	受信任的具有 SSL CA 的文件路径。
ssl_capath	包含受信任的 SSL CA 证书的目录路径。
ssl_crl	包含证书吊销列表的文件路径。
ssl_crlpath	包含已撤销的证书列表文件的目录路径。
ssl_cipher	在通过连接加密数据时允许使用的密码。
tls_version	安全通信将使用此版本及其协议。

这些选项是组配置所依赖的 MySQL server 配置选项。 此外，还有以下专门用于“配置插件本身的 SSL”的组复制选项。

- group_replication_ssl_mode - 指定组复制成员之间的连接的安全状态。

表 19.7 group_replication_ssl_mode 配置值

Value	描述
DISABLED	建立未加密的连接（默认）。
REQUIRED	如果 server 支持安全连接，则建立安全连接。
VERIFY_CA	类似于 REQUIRED，但会额外根据已配置的证书颁发机构（CA）证书进行验证 server TLS 证书。
VERIFY_IDENTITY	与 VERIFY_CA 类似，但会额外验证 server 证书与尝试连接的主机匹配。

以下示例展示了如何在 server 的 my.cnf 文件中配置 SSL 以及如何在组复制中激活它。

```
[mysqld]

ssl_ca = "cacert.pem"

ssl_capath = "/.../ca_directory"

ssl_cert = "server-cert.pem"

ssl_cipher = "DHE-RSA-AEs256-SHA"
```

```
ssl_crl = "crl-server-revoked.crl"

ssl_crlpath = "/.../crl_directory"

ssl_key = "server-key.pem"

group_replication_ssl_mode= REQUIRED
```

列出的选项中唯一专用于插件的配置选项为 group_replication_ssl_mode。 此选项通过“使用提供给 server 的 ssl_ *参数配置 SSL 框架”来激活组成员之间的 SSL 通信。

19.5.3 虚拟专用网（VPN）

组复制在虚拟专用网上运行没有什么限制条件。 它的核心只是依靠一个 IPv4 套接字在 server 成员之间建立连接，以便在它们之间传播消息。

19.6 组复制系统变量

本节是特定于组复制插件的系统变量。 每个配置选项都以“group_replication”作为前缀。

Important
虽然大多数动态变量，并且可以在 server 运行时更改，但大多数更改仅在重新启动组复制插件时生效。 在本节中主要关注那些不需要重启插件进行更改的变量。

- group_replication_group_name

引入	5.7.17	
命令行格式	--group-replication-group-name=value	
系统变量	名称	group_replication_group_name
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 此 server 实例所属组的名称。 必须是有效的 UUID 格式。

- group_replication_start_on_boot

引入	5.7.17	
命令行格式	--group-replication-start-on-boot=value	
系统变量	名称	group_replication_start_on_boot
	变量范围	Global
	动态变量	Yes
允许值	类型	boolean
	默认值	ON

- server 是否应在自身启动期间启动组复制。

- group_replication_local_address

引入	5.7.17	
命令行格式	--group-replication-local-address=value	
系统变量	名称	group_replication_local_address
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 作为本地地址，格式为主机:端口。

- group_replication_group_seeds

引入	5.7.17	
命令行格式	--group-replication-group-seeds=value	
系统变量	名称	group_replication_group_seeds
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 组内其他成员的地址列表，以逗号分隔开，如 host1:port1，host2:port2。

- group_replication_force_members

引入	5.7.17	
命令行格式	--group-replication-force-members=value	
系统变量	名称	group_replication_force_members
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 以逗号分隔开的组内其他成员的地址列表，如 host1:port1，host2:port2。 此选项用于强制建立新的组成员关系，在此过程中已排除的成员不接收新的视图并且被排除在外。 您需要手动移除已排除的 server。

- group_replication_bootstrap_group

引入	5.7.17	
命令行格式	--group-replication-bootstrap-group=value	
系统变量	名称	group_replication_bootstrap_group
	变量范围	Global
	动态变量	Yes
允许值	类型	boolean
	默认值	OFF

- 配置此 server 以引导组。 此选项只能在一个 server 上设置，且只能在首次启动组或重启整个组时设置。组被引导后，将此选项设置为OFF。它被动态地在配置文件中设置为OFF。当设置了此选项后，如果在组运行时启动两个 server 或重启一个 server，可能会导致人为脑裂情况，此过程会造成两个具有相同名称的独立组。

- group_replication_poll_spin_loops

引入	5.7.17	
命令行格式	--group-replication-poll-spin-loops=value	
系统变量	名称	group_replication_poll_spin_loops
	变量范围	Global
	动态变量	Yes
允许值 (32 位平台)	类型	integer
	默认值	0
	最小值	0
	最大值	4294967295
允许值 (64 位平台)	类型	integer
	默认值	0
	最小值	0
	最大值	18446744073709547520

- 在组通信线程等待传入更多的网络信息之前，等待 mutex 被释放的次数。

- group_replication_recovery_retry_count

引入	5.7.17	
命令行格式	--group-replication-recovery-retry-count=value	
系统变量	名称	group_replication_recovery_retry_count
	变量范围	Global
	动态变量	Yes
允许值	类型	integer
	默认值	10
	最小值	0
	最大值	31536000

- 要加入的成员尝试连接到可用数据源节点的次数。

- group_replication_recovery_reconnect_interval

引入	5.7.17	
命令行格式	--group-replication-recovery-reconnect-interval=value	
系统变量	名称	group_replication_recovery_reconnect_interval
	变量范围	Global
	动态变量	Yes
允许值	类型	integer
	默认值	60
	最小值	0
	最大值	0

- 在组中找不到数据源节点时，重新尝试连接的时间间隔（以秒为单位）。

- group_replication_recovery_use_ssl

引入	5.7.17

命令行格式	--group-replication-recovery-use-ssl=value	
系统变量	名称	group_replication_recovery_use_ssl
	变量范围	Global
	动态变量	Yes
允许值	类型	boolean
	默认值	OFF

- 组复制恢复通道是否使用 SSL。

- group_replication_recovery_ssl_ca

引入	5.7.17	
命令行格式	--group-replication-recovery-ssl-ca=value	
系统变量	名称	group_replication_recovery_ssl_ca
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 包含受信任 SSL 证书颁发机构列表的文件的路径。

- group_replication_recovery_ssl_capath

引入	5.7.17	
命令行格式	--group-replication-recovery-ssl-capath=value	
系统变量	名称	group_replication_recovery_ssl_capath
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 包含受信任的 SSL 证书颁发机构证书的目录的路径。

- group_replication_recovery_ssl_cert

引入	5.7.17	
命令行格式	--group-replication-recovery-ssl-cert=value	
系统变量	名称	group_replication_recovery_ssl_cert
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 用于建立安全连接的 SSL 证书文件的名称。

- group_replication_recovery_ssl_key

引入	5.7.17	
命令行格式	--group-replication-recovery-ssl-key=value	
系统变量	名称	group_replication_recovery_ssl_key
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 用于建立安全连接的 SSL 密钥文件的名称。

- group_replication_recovery_ssl_cipher

引入	5.7.17	
命令行格式	--group-replication-recovery-ssl-cipher=value	
系统变量	名称	group_replication_recovery_ssl_cipher
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 用于 SSL 加密的密码列表。

- group_replication_recovery_ssl_crl

引入	5.7.17	
命令行格式	--group-replication-recovery-ssl-crl=value	
系统变量	名称	group_replication_recovery_ssl_crl
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 包含具有证书撤销列表文件的路径。

- group_replication_recovery_ssl_crlpath

引入	5.7.17	
命令行格式	--group-replication-recovery-ssl-crlpath=value	
系统变量	名称	group_replication_recovery_ssl_crlpath
	变量范围	Global
	动态变量	Yes
允许值	类型	string

- 包含“具有证书撤销列表的文件的路径。

- group_replication_recovery_ssl_verify_server_cert

引入	5.7.17	
命令行格式	--group-replication-recovery-ssl-verify-server-cert=value	
系统变量	名称	group_replication_recovery_ssl_verify_server_cert
	变量范围	Global
	动态变量	Yes
允许值	类型	boolean
	默认值	OFF

- 在恢复过程中用于检查数据源节点数据源节点发送证书中的“通用名称”。

- group_replication_recovery_complete_at

引入	5.7.17	
命令行格式	--group-replication-recovery-complete-at=value	

系统变量	名称	group_replication_recovery_complete_at
	变量范围	Global
	动态变量	Yes
允许值	类型	enumeration
	默认值	TRANSACTIONS_APPLIED
	有效值	TRANSACTIONS_CERTIFIED
		TRANSACTIONS_APPLIED

- 状态传输后处理缓存中的事务时的恢复策略。 此选项指定成员在“接收到加入群组（TRANSACTIONS_CERTIFIED）之前遗漏的所有事务”或“收到并应用了这些事务（TRANSACTIONS_APPLIED）”之后，是否标记为在线。

- group_replication_components_stop_timeout

引入	5.7.17	
命令行格式	--group-replication-components-stop-timeout=value	
系统变量	名称	group_replication_components_stop_timeout
	变量范围	Global
	动态变量	Yes
允许值	类型	integer
	默认值	31536000

- 组复制在关闭时等待每个组件的超时时间（以秒为单位）。

- group_replication_allow_local_lower_version_join

引入	5.7.17	
命令行格式	--group-replication-allow-local-lower-version-join=value	
系统变量	名称	group_replication_allow_local_lower_version_join
	变量范围	Global
	动态变量	Yes
允许值	类型	boolean
	默认值	OFF

- 即使当前 server 的插件版本比组的插件版本低，也允许它加入组。

- group_replication_allow_local_disjoint_gtids_join

引入	5.7.17	
命令行格式	--group-replication-allow-local-disjoint-gtids-join=value	
系统变量	名称	group_replication_allow_local_disjoint_gtids_join
	变量范围	Global
	动态变量	Yes
允许值	类型	boolean
	默认值	OFF

- 即使含有组中不存在的事务，也允许当前 server 加入组，。

  - **Warning**

- 启用此选项时请谨慎，因为不恰当的使用可能会破坏组内的一致性。

- group_replication_auto_increment_increment

引入	5.7.17	
命令行格式	--group-replication-auto-increment-increment=value	
系统变量	名称	group_replication_auto_increment_increment
	变量范围	Global
	动态变量	Yes
允许值	类型	integer
	默认值	7
	最小值	1
	最大值	65535

- 确定在此 server 实例上执行的连续事务之间的步长。

- group_replication_compression_threshold

引入	5.7.17	
命令行格式	--group-replication-compression-threshold=value	
系统变量	名称	group_replication_compression_threshold
	变量范围	Global
	动态变量	Yes
允许值	类型	integer
	默认值	1000000
	最小值	0
	最大值	4294967296

- 以字节为单位的值，超过该值将强制执行（LZ4）压缩。 当设置为零时，压缩设置无效。

- group_replication_gtid_assignment_block_size

引入	5.7.17	
命令行格式	--group-replication-gtid-assignment-block-size=value	
系统变量	名称	group_replication_gtid_assignment_block_size
	变量范围	Global
	动态变量	Yes
允许值(32 位平台)	类型	integer
	默认值	1000000
	最小值	1
	最大值	4294967295
允许值 (64 位平台)	类型	integer
	默认值	1000000
	最小值	1
	最大值	18446744073709547520

- 为每个成员保留的连续 GTID 的数量。 每个成员在开始时会消耗掉一些，当需要时在获取更多个。

- group_replication_ssl_mode

引入	5.7.17	
命令行格式	--group-replication-ssl-mode=value	
系统变量	名称	group_replication_ssl_mode
	变量范围	Global
	动态变量	Yes
允许值	类型	enumeration
	默认值	DISABLED
	有效值	DISABLED
		REQUIRED
		VERIFY_CA
		VERIFY_IDENTITY

- 指定组复制成员之间的连接的安全状态。

- group_replication_single_primary_mode

引入	5.7.17	
命令行格式	--group-replication-single-primary-mode=value	
系统变量	名称	group_replication_single_primary_mode
	变量范围	Global
	动态变量	Yes
允许值	类型	boolean
	默认值	ON

- 设置组自动选择一个 server 来处理读/写工作。 这个 server 是主（PRIMARY），所有其他的都是从（SECONDARIES）。

- group_replication_enforce_update_everywhere_checks

引入	5.7.17	
命令行格式	--group-replication-enforce-update-everywhere-checks=value	
系统变量	名称	group_replication_enforce_update_everywhere_checks
	变量范围	Global
	动态变量	Yes
允许值	类型	boolean
	默认值	OFF

- 多主模式下为多主更新启用或禁用严格一致性检查。

- group_replication_flow_control_mode

引入	5.7.17	
命令行格式	--group-replication-flow-control-mode=value	
系统变量	名称	group_replication_flow_control_mode
	变量范围	Global
	动态变量	Yes
允许值	类型	enumeration
	默认值	QUOTA
	有效值	DISABLED
		QUOTA

- 指定启用限流模式。 不需要重置组复制就可以更改此变量。

- group_replication_flow_control_certifier_threshold

引入	5.7.17	
命令行格式	--group-replication-flow-control-certifier-threshold=value	
系统变量	名称	group_replication_flow_control_certifier_threshold
	变量范围	Global
	动态变量	Yes
允许值	类型	integer
	默认值	25000
	最小值	0
	最大值	2147483648

- 触发限流的验证队列的阈值。不需要重置组复制就可以更改此变量。

- group_replication_flow_control_applier_threshold

引入	5.7.17	
命令行格式	--group-replication-flow-control-applier-threshold=value	
系统变量	名称	group_replication_flow_control_applier_threshold
	变量范围	Global
	动态变量	Yes
允许值	类型	integer
	默认值	25000
	最小值	0
	最大值	2147483648

- 触发限流的应用队列的阈值 。不需要重置组复制就可以更改此变量。

- group_replication_ip_whitelist

引入	5.7.17	
命令行格式	--group-replication-ip-whitelist=value	
系统变量	名称	group_replication_recovery_ip_whitelist

	变量范围	Global
	动态变量	Yes
允许值	类型	string
	默认值	AUTOMATIC

- 指定允许哪些主机可以访问组。 默认设置为 AUTOMATIC，它允许来自主机上的私有子网的连接。主机上的活动网络接口将被扫描，其中私有网络网段的地址将被自动添加到允许列表中。 除此之外，您可以使用逗号分隔的 IPv4 地址列表或子网 CIDR 表示法来指定允许的主机。 例如 192.168.1.0/24,10.0.0.1。

  - **Note**
  - 始终允许地址 127.0.0.1 的访问，即使在 group_replication_ip_whitelist 中没有明确说明。

19.7 要求和限制

本节列出并阐述了组复制的要求和限制。

19.7.1 组复制要求

要用于组复制的 server 实例必须满足以下要求。

基础结构

- **InnoDB 存储引擎**。数据必须存储在 InnoDB 事务存储引擎中。事务以乐观的方式执行，然后在提交时检查冲突。如果存在冲突，为了保持整个组的一致性，某些事务将会回滚。这意味着需要使用事务存储引擎。此外，InnoDB 提供了一些额外的功能，当与组复制一起使用时，可以更好地管理和处理冲突。
- **主键**。每个在组中要被复制的表都必须定义一个显式主键。主键在识别每个事务修改了哪些行，哪些事务发生冲突时发挥了极其重要的作用。
- **IPv4 网络**。MySQL 组复制使用的组通信引擎仅支持 IPv4。因此，组复制需要 IPv4 网络基础结构。
- **网络性能**。组复制需要部署在相对集中的集群环境中，而且会受网络延迟和网络带宽的影响。

server 实例配置

组成员的中的 server 实例上必须配置以下选项。

- **Binary Log Active**。 设置--log-bin [= log_file_name]。 MySQL 组复制会复制二进制日志的内容，因此需要打开二进制日志才能操作。 请参见第 6.4.4 节“二进制日志”。
- **Slave Updates Logged**。 设置--log-slave-updates。 server 需要记录 applier 应用的二进制日志。 组中的 server 需要记录从组接收和应用的所有事务。 这些是必不可少的，因为恢复是依赖于各个组成员的二进制日志的。因此，每个 server 上都要有所有事务的副本，即使有些事务不是在当前 server 上发出的。

- **Binary Log Row Format**。 设置--binlog-format = row。 组复制依赖基于行的复制格式，以保证在组内 server 成员之间进行变更地一致性
- 传播。它基于行的基础架构能够获取必要的信息并在组中不同的 server 间对并发执行的事务做冲突检测。 请参见第 18.2.1 节“复制格式”。
- **Global Transaction Identifiers On**。 设置--gtid-mode = ON。 组复制使用全局事务标识符，来精确追踪哪些事务已在所有 server 实例上提交，从而能够推断哪些 server 执行了与已提交事务冲突的事务。 换句话说，显式事务标识符作为组复制框架的基础，保证了组能够判断出哪些事务可能会造成冲突。 请参见第 18.1.3 节“使用全局事务标识符复制”。
- **Replication Information Repositories**。 设置--master-info-repository = TABLE 和 --relay-log-info-repository = TABLE。 进行复制的 server 需要将主库信息和中继日志元数据写入 mysql.slave_master_info 和 mysql.slave_relay_log_info 系统表。 这确保了组复制插件具有一致的可恢复性和对复制元数据的事务管理的一致性。 请参见第 18.2.4.2 节“从成员的状态日志”。
- **Transaction Write Set Extraction**。 设置--transaction-write-set-extract = XXHASH64，以便在 server 收集写集合的同时将其记录到二进制日志。 写集合基于每行的主键，并且是行更改后的唯一标识。此标识将用于检测冲突。

19.7.2 使用限制

组复制存在以下已知的限制。

- **Replication Event Checksums**。 由于复制事件校验和的设计限制，组复制当前无法使用它。 因此需要设置--binlog-checksum = NONE。
- **Gap Locks**。认证过程没有考虑间隙锁，因为有关间隙锁的信息在 InnoDB 范畴外不可用。有关详细信息，请参阅“间隙锁”这一章节。

  Note

  除非应用程序依赖于 REPEATABLE READ，否则我们建议将 READ COMMITTED 隔离级别与组复制配合使用。。InnoDB 在 READ COMMITTED 模式下没有间隙锁，这会使 InnoDB 本身的冲突检测机制与组复制的分布式检测机制协同工作。
- **Table Locks and Named Locks**。 认证过程不考虑表锁（参见第 14.3.5 节“LOCK 表和 UNLOCK 表语法”）或命名锁（参见 GET_LOCK（））。
- **Savepoints Not Supported**。 不支持事务保存点。

- **可串行隔离级别**。 默认情况下，多主组不支持可串行隔离级别。 将事务隔离级别配置为 SERIALIZABLE（可串行）时组复制将会拒绝提交事务。
- **并发 DDL 与 DML 操作**。 当使用多主模式时，在不同 server 上针对同一对象（object）并发执行数据定义和数据操作语句是不支持的。当在对象上执行 DDL 语句时，对同一对象但是不同 server 实例上的并发 DML 操作，会存在“未检测到不同实例上执行冲突的 DDL 语句”风险。
- **具有级联约束的外键**。 多主模式下（所有配置为 group_replication_single_primary_mode = OFF 的成员）不支持具有多级外键依赖关系的表，特别是已定义了级联外键约束的表。这是因为外键约束可能会导致多主模式组执行的级联操作造成检测不到的冲突，并且导致该组的成员之间的数据不一致。 因此，我们建议在多主模式组中的 server 实例上设置

group_replication_enforce_update_everywhere_checks = ON，以避免未检测到的冲突发生。

在单主模式下，不存在此问题，因为它不允许对组的多个成员的并发写入，所以不存在未检测到的冲突的风险。

19.8 常见问题

本节提供常见问题的答案。

组中 MySQL server 的最大数量是多少？

一个组最多可以包含 9 台 server。 如果尝试向已经具有 9 个成员的组中添加其他 server，请求会被拒绝。

组中的 server 之间如何连接？

组中的 server 通过打开对等的 TCP 连接来连接到组中的其他 server。 这些连接仅用于组中 server 成员之间的内部通信和消息传递。

选项 group_replication_bootstrap_group 有什么用途？

引导标志指示一个成员创建组并充当初始种子 server。 第二个加入组的成员需要向引导组的那个 server 请求动态地更改配置，以便将其添加到组中。

一个server会在两种情况下引导组。一个是最初创建组时，另一个是关闭并重新启动整个组时。

如何设置恢复过程的凭据？

您可以使用 CHANGE MASTER TO 语句预配置组复制恢复通道凭据。

我可以使用组复制横向扩展我的写入负载吗？

不能直接的进行扩展。但是 MySQL 组复制是一个 shared nothing 的完整复制解决方案，其中组内所有 server 都会复制相同数量的数据。 因此，如果组中的一个成员提交事务操作将 N 个字节进行存储，则其他成员也会存储大约 N 个字节，因为事务在所有 server 成员中都会被复制。

然而，考虑到其他成员不必进行与初始成员在最初执行事务时必须做的处理相同的处理量，所以它们能更快地执行变更。 事务仅以行转换的格式进行复制，而不必再次重新执行事务（基于行的格式）。

此外，考虑到变更操作是以行的格式进行传输和应用的，这意味着它们将以优化过的紧凑的格式被接收，并且与源成员相比，所需的 IO 操作的数量可能会减少。

总而言之，您可以通过在组中的不同成员之间传播无冲突的事务来横向扩展写处理。 由于远程服务器只接收对稳定存储的“读取-修改-写入”更改，因此您就有可能减少一小部分 IO 操作。

在相同工作负载下，对比单纯的复制，组复制是否需要更多的网络带宽和 CPU？

Server 成员之间为了保持同步需要不断地进行交互，因此会需要一些额外的负载。但是这个数据很难量化。它还取决于组的大小(三个 server 的组对带宽要求的压力小于九个 server 的组)。

此外，因为 server 为保持组内同步和传递组内消息需要完成更复杂的工作，会占用更多的内存和 CPU。

我可以跨广域网部署组复制吗？

是的，但每个节点之间的网络连接必须是可靠的且具有相匹配的性能。 为了实现最佳性能，就要有低延迟、高带宽的网络连接。

如果网络带宽有问题，则可以参考第 19.9.7.2 节“消息压缩”来降低对带宽的要求。 然而，如果网络丢包，导致重传和更高层的端到端延迟，吞吐量和延迟都会受到负面影响。

Warning

当任何组成员之间的网络往返时间（RTT）为 2 秒或更长时，您就可能会遇到问题，因为内置的故障检测机制可能会错误触发。

在遇到临时的连接问题时，节点是否自动重新加入组？

这取决于连接问题的原因。 如果连接问题是暂时的并且恢复连接足够快，在故障检测器还没检测到时就进行了重新连接，则可以不将 server 从组中移除。 如果是长时间的连接问题，则故障检测器最终会检测到问题，并且该 server 将被从组中移除。

server 从组中移除后，您就需要重新将它加入组。 换句话说，从组中删除 server 后，您就需要手动重新加入（或使脚本自动执行）。

什么时候从组中移除成员？

如果某个成员无响应，其他成员将把它从组配置中移除。 在实践中，这种情况可能在某个成员崩溃或网络连接断开时发生。

当某个成员超时的时候，系统会检测到故障，并且创建一个不包含该成员的新配置。

当一个节点非常明显地延迟时会怎么处理？

没有用于定义何时从组中自动排除成员的策略。 你需要找出该成员延迟的原因，并修复它或从组中删除该成员。 否则，如果 server 慢到足以触发流量控制，那么整个组的运行也将减慢。 流量控制可以根据您的需要进行配置。

在怀疑组中存在问题时，组中是否有某个特定成员负责触发重新配置组？

不，在组中没有特定的成员负责触发重新配置。

任何成员都可以怀疑组中存在问题。 所有成员需要（自动）对“某个给定成员故障”达成一致意见。 有一个成员负责触发重新配置从组中将故障成员移除。 具体哪个成员负责移除故障成员是不可以自行配置的。

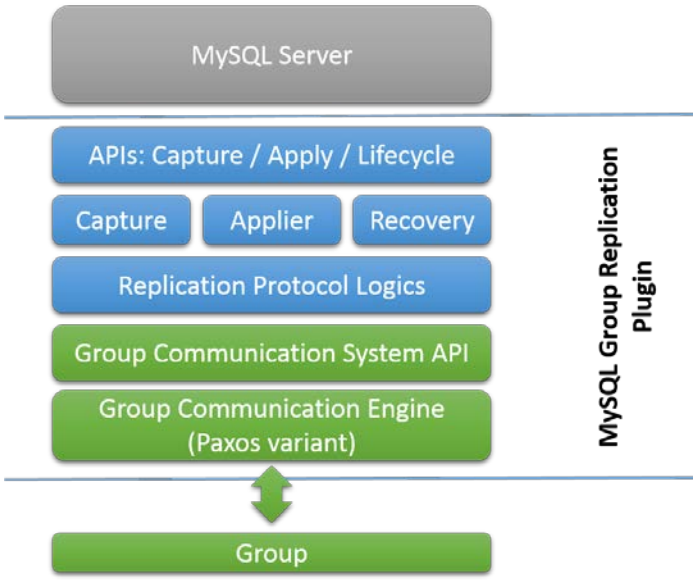
19.9 组复制技术详细信息

本节提供有关 MySQL 组复制的更多的技术细节。

19.9.1 组复制插件架构

MySQL 组复制是一个 MySQL 插件，它是建立在现有的 MySQL 复制架构的基础上的，并且利用了二进制日志、基于行的日志记录和全局事务标识符等功能。 它集成了当前的一些 MySQL 框架，如性能模式或插件和服务基础架构。 下图展示了描述 MySQL 组复制的整体架构图。

图 19.9 组复制插件框图



在图的顶部可以看到，有一组API用来控制组复制插件如何与MySQL server进行交互（灰色框）。有一些接口可以使信息在 server 和插件之间交互。 这样的接口将 server 的核心与插件隔离，并且主要是放置在事务执行管道中的模块。 在从 server 到插件的方向上，有 server 正在启动、server 正在恢复、server 准备接受连接、server 即将提交事务等等这样的通知。 在反方向上，有插件指示 server 提交或中止正在进行的事务、给中继日志中的事务排队等等的通知。

在 API 块下面是一组组件，当通知路由到它们会做出相应的响应。 捕获组件负责跟踪与正在执行的事务相关的上下文。应用组件负责在数据库上执行远程事务。 恢复组件管理分布式恢复，并且负责通过选择数据源节点、协调数据恢复程序，将加入组的 server 数据更新到最新,并对源服务器异常做出响应。

接着往下是复制协议模块，它包含复制协议的特定逻辑。 它用于处理冲突检测，接收和向组传播事务。

第一个绿色框是组通信 API（Group Communication API）。 它是一个用于提取构建复制状态机所需属性的高级 API（详细信息参见后面的章节）。 因此，它将“消息层的实现”与“插件层中的剩余上层模块”进行了解耦。 MySQL 组复制插件包含了基于 Paxos 的组通信引擎。

19.9.2 组

在 MySQL 组复制中，一组 server 构成一个复制组。 组具有一个 UUID 形式的名称。 该组是动态的，server 可以随时离开（自愿或非自愿），并随时加入组。 每当有 server 加入或离开时，组都会自行调整。

如果某个 server 加入组，则它会通过从组中现有 server 成员中获取“缺失状态”来自我更新。此状态通过异步 MySQL 复制进行传输。如果某个 server 离开组，例如它被移除出组进行维护，

则剩余的 server 会意识到它已经离开了并自动重新配置该组。 这些都是有第 19.1.3.2 节“组成员关系”驱动的。

19.9.3 数据操作语句

由于没有用于某个特定数据集的主服务器（masters），所以组中的每个 server 都可以在任何时间执行事务，即使是改变数据状态的事务（RW 事务）。

任何 server 成员都可以在没有任何预先协调的情况下执行事务。 但是，在提交时，它需要与组中的其余 server 成员协商以决定怎么处理该事务。 这种协商有两个目的：（i）检查事务是否应该提交;（ii）传播更改，以便其他 server 成员也可以应用该事务。
因为事务是通过原子广播发送的，所以组中的所有 server 成员都会接收到事务，或者都接收不到事务。 如果他们接收到该事务，那么它们都会按照与之前发送的其他事务相同的顺序接收该事务。 冲突检测是通过检查和比较写入事务集来执行的。 因此，这种检测是行级别的。 冲突解决遵循“首次提交者胜出”原则。 如果 t1 和 t2 在不同的节点同步执行，因为 t2 排在 t1 前面，并且都改变了同一行，则 t2 在冲突中胜出，t1 中止。 换句话说，t1 正在试图更改在 t2 看来已经过时的数据。

Note
更多的情况下，如果两个事务存在冲突，那么最好在同一个 server 上执行它们。 它们就有机会在本地的锁管理器上进行同步，而不是稍后在复制协议中异中止。

19.9.4 数据定义语句

在执行 DDL 语句（通常被称为数据定义语言）时需要格外小心。 由于 MySQL 不支持原子或事务性 DDL，因此不能乐观地执行 DDL 语句并在稍后必要的时候进行回滚。 因此，原子性的缺失不直接适用于组复制的乐观复制范例。

因此，在复制 DDL 语句时需要更加小心。在数据库模式（schema）操作尚未完成并复制到组内所有 server 成员上时，对数据库模式和该对象包含的数据更改需要通过同一 server 处理。 否则可能导致数据不一致。

Note

如果组以单主模式部署，则这就不是问题，因为所有更改都会通过同一 server（主服务器）执行。

Warning
MySQL DDL 执行不是原子性的或事务性的。 Server 在未确保组内达成一致的情况下，会首先执行并提交操作。 因此，在某一 DDL 正在执行并且尚未复制到所有 server 成员时，您必须将对于该对象的 DDL 和 DML 操作路由到同一 server 上。

19.9.5 分布式恢复

本节介绍正在加入的成员与其余 server 成员进行同步的过程，称为分布式恢复阶段。

19.9.5.1 组复制基础

组复制分布式恢复过程可以概括为，一个新的 server 从组中的某个在线 server 成员获取缺失的数据，同时侦听组中正在发生的事件的过程。 在恢复期间，server 会侦听组成员关系事件以及恢复过程中组中正在发生的事务。 这是一个高层级的总结。 以下部分通过描述过程的两个阶段来提供更多的细节信息。

阶段 1

在第一阶段，加入者（正在加入的 server）将选择组中的一个在线 server 成员作为其缺少的状态的源服务器。源服务器负责提供其加入该组的那一刻之前所遗漏的所有数据。 这是通过依靠建立在源服务器和加入者之间的标准异步复制通道来实现的。 通过该通道，二进制日志向上流动，直到当加入者成为组的一部分时视图发生改变的那一刻。 当加入者接收来自数据源节点的二进制日志时，它也在应用这些日志。
此外，当二进制日志传送正在进行时，加入者也会缓存在组内交互的每个事务。 也就是说，它在应用来自源服务器的丢失状态时，同时也在侦听在加入该组之后发生的事务。 当第一阶段结束并且到源服务器的复制通道被关闭时，加入者就会进入第二阶段：追赶。

阶段 2

在这个阶段，加入者继续执行已缓存的事务。当排队等待执行的事务的数量最终为零时，该成员将被声明为在线。

弹性

当加入者从源服务器获取二进制日志时，恢复过程可以处理源服务器失败的情况。 在这种情况下，源服务器无论何时在阶段 1 中失败，加入者都会转移到新的源服务器并从该源服务器处进行恢复。 当发生这种情况时， 加入者会明确地关闭与发生故障的源服务器之间的连接，并建立与新的源服务器的连接。 这是自动进行的。

19.9.5.2 从时间点恢复

为了使加入者与源服务器同步到特定的时间点，加入者和源服务器需要利用 MySQL 全局事务标识符（GTID）机制。 但这还不够，因为这仅仅提供了一种辨别加入者缺少哪些事务的方法。 它不能标记加入者必须更新到的特定时间点，也不会传送认证信息。 二进制日志视图标记在这里就可以派上用场了。 它们标记二进制日志流中的视图更改，而且还包含其他元数据信息，从而为加入者提供缺失的认证相关数据。

19.9.5.2.1 视图和视图更改

为了解释视图变化标记的概念，理解视图和视图变化是很重要的。

一个视图对应于一组活跃在当前配置下（即特定时间节点）的组成员。 他们是正常的并且在系统中是在线状态。
当组配置修改（例如有成员加入或离开）时发生视图改变。 任何组成员关系的更改，都会导致在同一逻辑时间点向所有组成员传送独立的视图更改。

视图标识符唯一地标识视图。 每当视图更改时生成视图标识符。

在组通信层，具有“与其相关联的视图 ID”的视图改变则是“组成员加入前后交换的数据”之间的边界。 这个概念是通过一个新的二进制日志事件实现的：“视图更改日志事件”。 因此，视图 id 也成为了组成员关系变化前后发送的事务的标记。

视图标识符本身由两部分构成：（i）一个随机生成的和（ii）一个单调递增的整数。 第一部分在创建组时生成，并且在组中至少有一个成员时保持不变。 第二部分在每次视图更改发生时递增。

需要使用这种构成视图 id 的异构对的原因是，每当有成员加入或离开时，或者当所有成员离开组并且该组所在的视图没有剩余信息时，需要清楚地标记组更改。实际上， 单独使用单调增加的标识符可能导致在完全关闭组之后重用相同的 ID，破坏了恢复所依赖的二进制日志数据标记的唯一性。 总而言之，第一部分在某个组初始创建时标记，逐渐增加的第二部分在组创建后发生改变时进行标记。

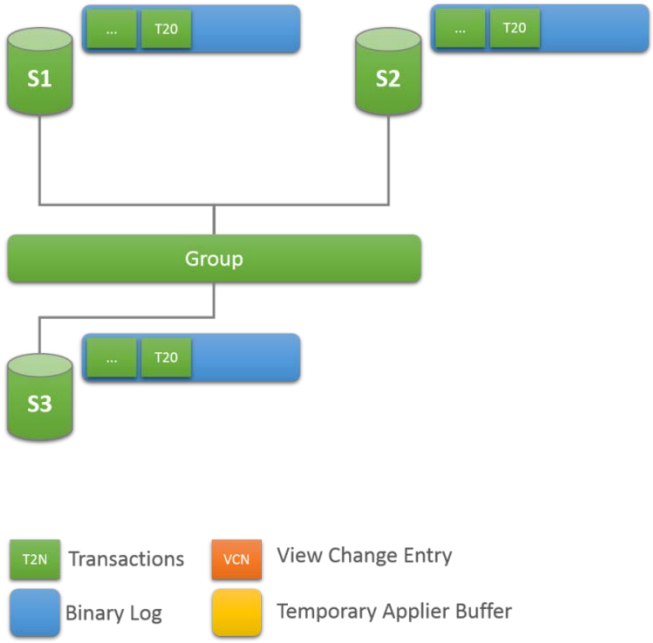
19.9.5.3 视图更改

本节介绍如何将视图更改标识符合并到二进制日志事件并写入日志的过程，执行以下步骤：

开始：稳定组

所有 server 成员都处于在线状态，并处理来自组的传入事务。 有一些 server 成员可能在事务复制方面稍微落后，但最终它们会趋于一致。 该组充当一个分布式的复制数据库。

图 19.10 稳定组

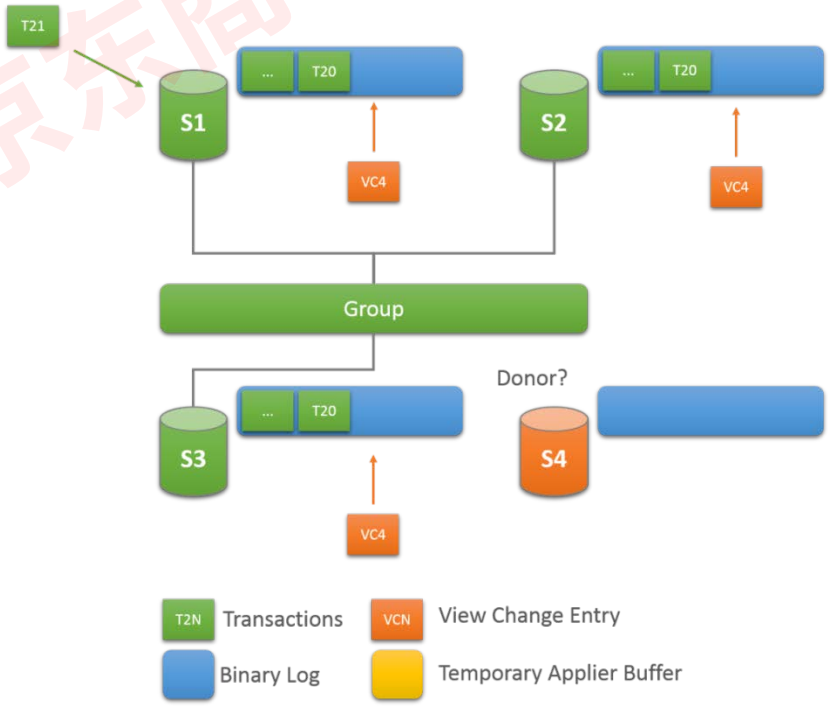


视图更改：成员加入

每当新成员加入组并且因此执行视图更改时，每个在线 server 都会将视图更改日志事件加入执行队列以供应用。 这是需要排队的，因为在视图更改之前，可能已有几个事务在 server 上排队等待应用，因此，他们属于旧视图。将视图更改事件排在他们后面，可以保证正确标记这种情况的发生时间。

同时，加入者通过视图抽象，从组成员服务所声明的在线 server 列表中选择源服务器。 成员加入视图 4 中，在线的成员会将视图更改事件写入二进制日志中。

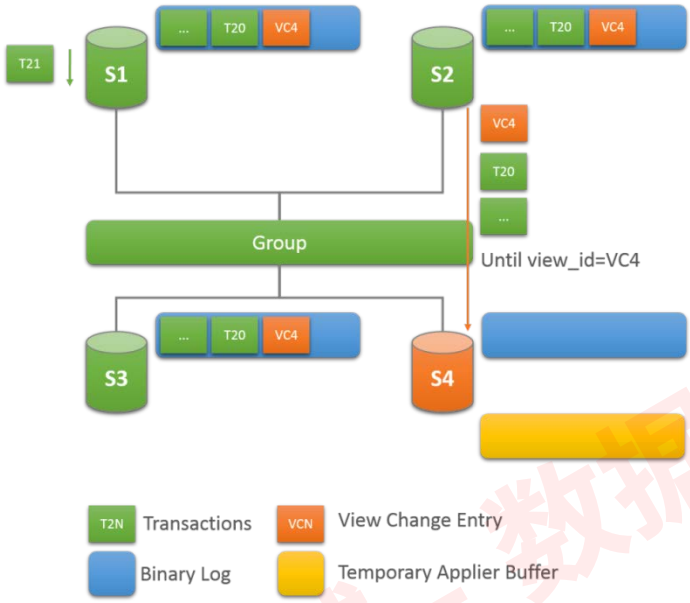
图 19.11 成员加入



状态传送：追赶同步

当加入者已经选择好组中的哪个 server 成员将作为源服务器后，则在两者之间建立新的异步复制连接，并且开始进行状态转移（阶段 1）。 与源服务器的这种交互，持续到加入者的应用线程处理对应于“加入者进入组中时触发的视图改变”的视图更改日志事件为止。 换句话说，加入者从源服务器复制数据，直到它到达具有与它已有的视图标识符匹配的标识符为止。

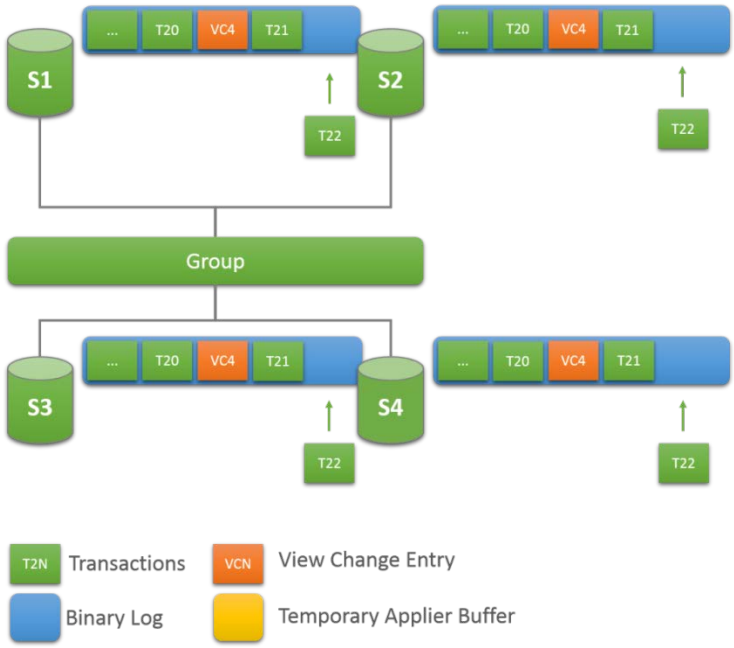
图 19.12 状态传送：追赶同步



由于视图标识符在同一逻辑时间被发送到组中的所有成员，所以加入者知道它应该停止复制哪个视图标识符。这避免了复杂的 GTID 集计算，因为视图 id 清楚地标记每个组视图都有哪些数据。

当加入者从源服务器复制数据时，它也会缓存来自组的传入事务。 最终，它停止从源服务器复制，转而去应用那些缓存中的事务。

图 19.13 排队的事务



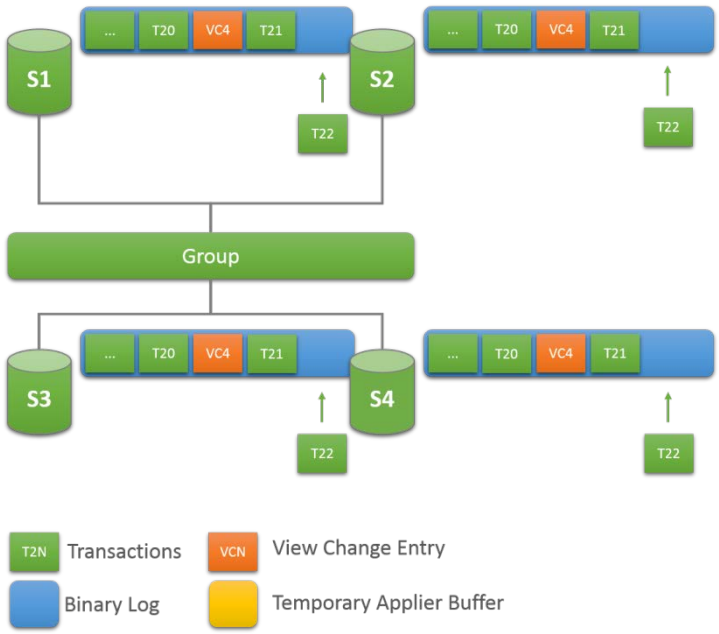
完成：已赶上并同步

当加入者识别到具有预期视图标识符的视图更改日志事件后，到源服务器的连接终止，并且它会开始应用缓存中的事务。 要理解的一个重点是最终的恢复过程。视图更改日志事件作为二进制日志中的标记，定义视图更改，但是它也起到另一个作用。 当加入者进入组时（最后一次视图改变时），视图更改日志事件传送被组中所有 server 成员认可的认证信息。 没有它，加入者将不具有能够证明（检测冲突）后续事务的必要信息。

追赶（阶段 2）阶段的持续时间是不确定的，因为它取决于工作负载和传入组的事务的速率。 此过程完全处于在线状态，并且加入者在追赶时不会锁定组中的任何其他 server 成员。 因此，当加入者进入到阶段 2 时，落后的事务数量会依工作负荷而增加或减少。

当加入者的排队事务达到零并且其存储的数据与其他成员相等时，其对外状态更改为在线。

图 19.14 在线实例



19.9.5.4 分布式恢复的使用建议和限制

分布式恢复有一些限制。 它基于典型的异步复制，因此如果加入者没有提供或者提供非常旧的备份镜像，复制可能会很慢。 这意味着如果在阶段 1 要传输的数据量太大，则 server 可能需要很长时间才能恢复。 因此，建议在将 server 添加到组之前，应该使用该组中已有的 server 成员提供的最新快照。 这可以将阶段 1 的长度最小化，并减少对源服务器的影响，因为它保存和传输的二进制日志都将减少。

Warning

建议在将 server 添加到组之前对其进行预配置。 这样可以最小化在恢复阶段上花费的时间。

19.9.6 可观察性

Group Replication 插件中内置了大量自动化。 尽管如此，你有时可能需要了解后台发生了什么。组复制和性能模式的检测在这里就变得很重要了。 可以通过 performance_schema 表查询系统的整个状态（包括视图，冲突统计和服务状态）。 由于复制协议具有分布式性质而且 server 实例达成共识并同步事务和元数据，组状态的检查因此变得更加容易。 例如，连接到组中的某个 server 后，可以通过在组复制相关的性能模式表上发出 select 语句来获取本地和全局信息。有关详细信息，请参见第 19.3 节“监控组复制”。

19.9.7 组复制性能

本节介绍如何使用可用的配置选项，以获得组的最佳性能。

19.9.7.1 微调组通讯线程

加载组复制插件后，组通信线程（GCT）将会循环运行。 GCT 从组和插件接收消息，处理仲裁成员数和故障检测相关任务，发送一些保活消息，并且还处理 server 成员和组之间的事务传递。GCT 等待队列中的传入消息。 当没有消息时，GCT 持续等待。在实际进入睡眠之前， 通过将此等待时间配置得更长（进行主动等待），在某些情况下可以证明是有益的。 这是因为替代方案是操作系统从处理器切换 GCT 并做上下文切换。

要强制 GCT 进行主动等待，请使用 group_replication_poll_spin_loops 选项，这使得 GCT 在对下一个消息进行实际轮询队列之前，在执行已配置的循环次数内进行循环时不做任何相关操作。

例如：

```
mysql> SET GLOBAL group_replication_poll_spin_loops= 10000;
```

19.9.7.2 消息压缩

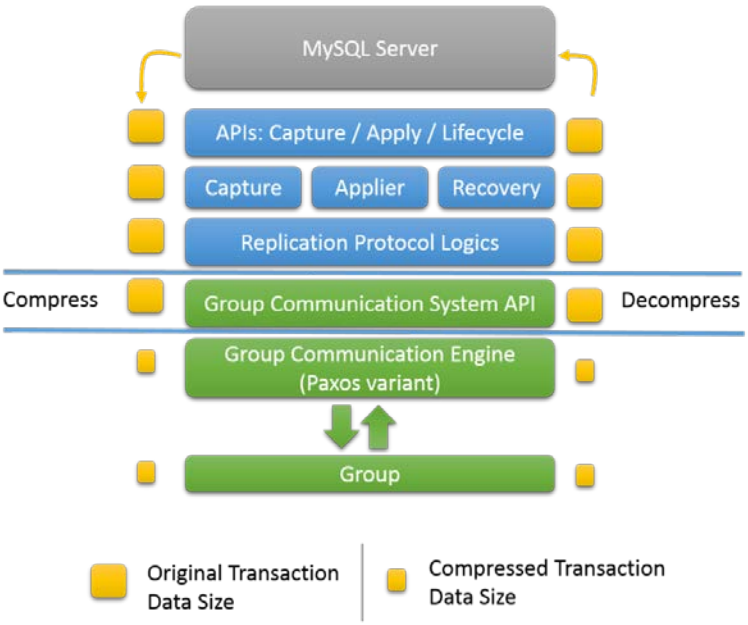
当网络带宽是瓶颈时，消息压缩可以在组通信级别改进高达 30-40%的吞吐量。 这在负载较大的大型 server 组上尤其重要。

表 19.8 不同二进制日志格式的 LZ4 压缩比。

	比率	
Workload	ROW	STMT
mysqlslapd	4,5	4,1
sysbench	3,4	2,9

组中 N 个成员之间的互连具有 TCP 对等的性质，使得发送者需要发送 N 次相同数量的数据。此外，二进制日志可能呈现出较高的压缩比（见上表）。 这使得压缩成为了一个对于包含大事务的工作负载来说引人注目的功能。

图 19.15 压缩支持



压缩发生在数据被移交给组通信线程之前的组通信引擎级别，所以它发生在 mysql 用户会话线程的上下文中。 事务有效载荷可能在被发送到组之前被压缩，并且在被接收时解压缩。 压缩是有条件的，并且取决于已配置好的阈值。 默认情况下压缩是启用状态。
此外，组中的 server 成员没有必要都启用压缩来共同工作。 在接收到消息时，组成员检查消息信封以确认它是否被压缩。 如果需要，则该成员解压缩该事务，然后将其传递到上层。

所使用的压缩算法是 LZ4。 默认情况压缩是启用状态，阈值为 1000000 字节。 压缩阈值（以字节为单位）可以设置为大于默认值的值。 在这种情况下，只有具有大于阈值的有效负载的事务会被压缩。 下面是如何设置压缩阈值的示例。

```
STOP GROUP_REPLICATION;

SET GLOBAL group_replication_compression_threshold= 2097152;

START GROUP_REPLICATION;
```

这将压缩阈值设置为 2MB。 如果事务生成了有效负载大于 2MB 的复制消息，例如大于 2MB 的二进制日志事务条目，则它将会被压缩。 如果要禁用压缩，需要设置阈值为 0。

19.9.7.3 流量控制

组复制会确保事务仅在“组中的大多数成员接收到它，并且对并发发送的所有事务之间的相对顺序达成共识”时提交。

如果对组的写入总数不超过组中任何成员的写入容量，则此方法可以正常运转。如果对组的写入总数超过了组中某些成员的写入容量，一些成员的写入吞吐量比其他成员少，特别是小于写入成员，则这些成员会开始落后于写入者。

有一些成员滞后于该组将带来一些不良的后果，特别是，对这些成员的读取可能得到旧数据。 根据成员滞后的原因，组中的其他成员可能必须或多或少地保存复制上下文，以满足来自滞后成员的潜在数据传输请求。

然而，在复制协议中存在一种机制，以避免快速和慢速成员之间在应用的事务数量方面具有太大的差距。 这被称为流控制机制。 它试图达到以下几个目标：

1.保持成员状态足够接近，使成员之间缓冲和不同步的问题尽可能变小;

2.快速适应组中的变化，如不同的工作负载、写入者增加等;

3.使组成员间公平地分享可用的写入能力;

4.在非严格必要的情况下，不减少吞吐量，以避免资源浪费。

考虑到组复制的设计，是否节流需要考虑到如下两个工作队列：（i）认证队列;（ii）和二进制日志应用队列。 每当这两个队列之一的大小超过用户定义的阈值时，就会触发节流机制。 只需要进行如下配置：（i）是否在验证者或应用者级别或两者同时进行流量控制;和（ii）每个队列的阈值是多少。

流控制取决于两个基本机制：

1. 成员的监控，用以收集关于所有组成员的吞吐量和队列大小的一些统计数据，进而对每个成员可以承受的最大写入压力进行有根据的猜测;

2. 时刻关注，及时地对于试图超越其可用能力的公平份额的成员进行限制。

19.9.7.3.1 探测和统计

监视机制通过让每个成员部署一组探测器，来收集有关其工作队列和吞吐量的信息。 然后，它定期将该信息传播到组，以与其他成员共享该数据。

这样的探测器散布在整个插件堆栈中，并允许建立度量标准，例如：

- 验证者队列大小;
- 复制应用队列大小;
- 已认证的事务的总数;
- 在成员上应用的远程事务的总数;

•本地事务的总数。

成员收到来自另一成员的统计信息后，它将计算有关“在最后一个监控周期中认证、应用和本地执行的事务数量”的其他度量指标。

监视数据会定期地与组中的其他成员共享。 监视周期必须足够高，以允许其他成员决定当前写入请求，但是也要足够低，以将对组带宽的影响降到最小。 信息每秒进行共享，这个时间段足以解决这两个问题。

19.9.7.3.2 组复制节流

基于从组中的所有 server 成员收集的度量标准，节流机制启动并决定是否限制某个成员执行/提交新事务的速率。

因此，从所有成员获取的度量标准是计算每个成员容量的基础：如果某个成员有一个大队列（用于认证或应用线程），则用以执行新事务的容量应接近经认证或应用的最后一个时间段。

组中所有成员的最低容量决定了组的实际容量，而本地事务的数量决定了有多少成员正在向其写入，也因此决定了应该与多少成员共享可用容量。

这意味着每个成员都具有已建立的基于可用容量的写配额，也就是它可以在下一个周期安全地发出的事务数量。 如果验证者或二进制日志应用队列大小超过了用户定义的阈值，则写配额将由限制机制强制执行。

配额会在减少上一个周期中延迟的事务数后，进一步减少 10%，以使触发该问题的队列减小其大小。 为了在队列大小超过阈值时避免大的吞吐量跳跃，在此之后，仅允许吞吐量每个周期增长相同的 10%。

当前节流机制不会处理低于配额的交易，而是会延迟完成那些超过配额的交易，直到该监控周期结束。 因此，如果配额对于发出的写请求来说非常小，一些事务可能需要等待接近一个监控周期的时间。