MySQL 组复制官方手册翻译

手册原文地址：Group Replication

译者：马龙帅

日期：2018-06-18(端午节)

关于本译文：

1.有两部分没有翻译，分别是"组复制的安全"和"组复制相关的系统变量"，我觉得没必要翻译。
2.MySQL的的官方手册是会更新的，所以本译文对应的内容和官方最新版的英文手册可能有些区别。
3.全文翻译后没有做过校对，加之本人蹩脚的英语水平，所以可能会有些错误。还望见谅。
4.本人博客地址：https://www.cnblogs.com/f-ck-need-u/
5.本人写的组复制系列文章：http://www.cnblogs.com/f-ck-need-u/p/7586194.html#mysqlcluster

Chapter 18 Group Replication

Table of Contents

本章介绍MySQL Group Replication(MySQL组复制)技术，以及如何安装、配置和监控复制组。MySQL组复制通过MySQL插件实现弹性的、高可用的、可容错的复制拓扑结构。

MySQL复制组可以在单主模式(single-primary/single-master)下自动选举出一个主节点，从而只允许在同一时刻只有该主节点可以更新数据。另外，对于MySQL的高级使用人员，可以通过复制组实现多主模型(multi-primary)，这种模型下，所有的主节点都可以在同一时刻接受更新操作，即并发写。

MySQL有一个内置的组成员服务(group membership service)，该服务动态地维护了组中成员以及可用成员的信息视图。每当成员离开组、加入组时，该视图都会随之更新。如果某MySQL实例在非预期的情况下离开了组，失败探测机制可以探测出这种情况并通知组，告知视图已经更改，这是自动完成的。

本章内容的结构如下：

- Section 18.1, “组复制的背景” 介绍组以及组复制是如何工作的。

- Section 18.2, “组复制入门” 介绍如何配置MySQL来创建组。
- Section 18.3, “监控组复制” 介绍如何监控组。
- Section 18.5, “组复制的安全” 介绍如何使组安全。
- Section 18.9, “组复制技术的细节” 深入介绍组复制的工作原理。

18.1 组复制的背景

本节介绍组复制的背景信息。

创建一个可容错的系统最通用的方法是使组件冗余，也就是说在移除一些组件后，容错系统还能按预期的行为运行下去。但这会将拓扑结构的复杂性提升到一定的程度。特别是对于数据库复制，它需要维护和管理多个服务节点而并非简简单单的一个节点。此外，由于各服务器是协同工作的，必然会带来分布式系统的一些通用问题，例如如何处理网络分裂(network partitioning)以及脑裂(split brain)。

因此，最大的挑战是如何将数据库逻辑与多节点复制的逻辑融合在一起，并以简单的、一致性协同方式工作。换句话说，是要让多个节点的状态、每个节点的数据、每个节点经历过的变化都达成一致。这可以概括为让每个节点就每个数据库的状态转换达成一致，就像单节点数据库那样，或者它们最终趋于一致。这意味着它们需要提供一个(分布式)状态机。

MySQL组复制提供的分布式状态机要求各节点之间是强一致性的。组中的各节点会自动协调它们自身。复制组可以以单主模型运行，并自动选举出这个主节点。对于高级使用人员，可以配置多主模型，使得多个主节点可以同时接受数据更新操作。但这种能力要求应用程序必须解决这种部署带来的限制，这是使用组复制需要付出的代价。

MySQL组复制有一个内置的组成员服务(group membership service)，该服务动态维护组中成员以及可用成员的信息视图。每当成员离开组、加入组时，该视图都会随之更新。如果某MySQL实例在非预期的情况下离开了组，失败探测机制可以探测出这种情况并通知组，告知视图已经更改，这是自动完成的。

对于一个将要提交的事务，必须要求组中的大部分成员就"某事务在全局事务序列中的顺序(位置)"达成一致。虽然决定一个事务的提交或中断一个事务是由各节点自身处理的，但所有的节点必须做出相同的决定。如果出现网络分裂问题，将会隔离组中的成员，这样就无法让各节点对决定达成一致，那么在问题解决之前，这个MySQL复制集群就无法继续处理。因此，MySQL组复制中提供了一个内置的可以自动检测并防止脑裂的机制。

所有的这些都由组通信协议提供。包括：故障探测机制，组成员服务，安全且完全有序的信息投递功能。这些特性是创建一个能保证数据一致性的组复制的关键。该技术的核心是 Paxos 算法的实现，它充当了组通信系统引擎的角色。

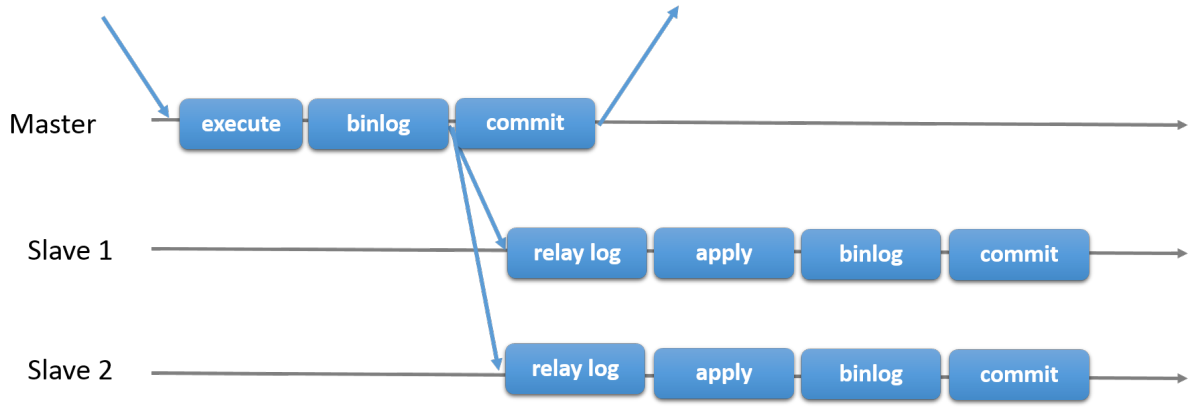
18.1.1 复制技术

在详细介绍MySQL组复制之前，本节先介绍一些背景概念以及它们的基本工作方式。如此可方便理解组复制以及传统的MySQL异步复制与组复制之间的区别。

18.1.1.1 主从复制

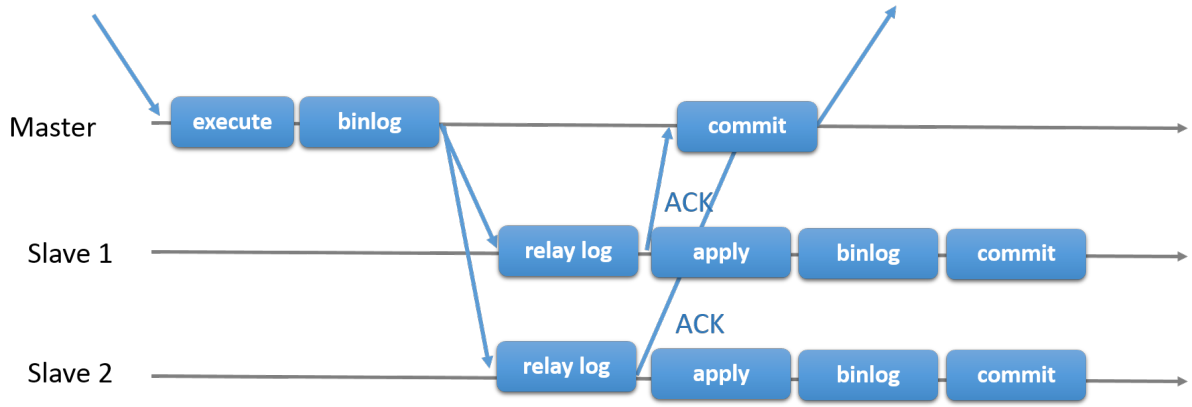
传统的MySQL复制提供了一个简单的主从复制方法。一个master节点(primary)，一个或多个slave节点。master执行事务、提交事务，然后(因此是异步)将所作的操作发送给slave，slave重放这些操作。这是一种share-nothing的系统，所有的节点默认都具有一个数据的完整拷贝。

Figure 18.1 MySQL异步复制



除了异步复制，还有半同步复制，它在异步复制基础上添加了一个同步操作。master需要等待slave的ack回复，然后才能提交事务。

Figure 18.2 MySQL半同步复制



上面两张图中，你可以看到一个经典的MySQL异步复制(以及变体的半同步复制)图解。蓝色箭头代表的是master和slave之间、master和客户端应用程序之间的信息交换过程。

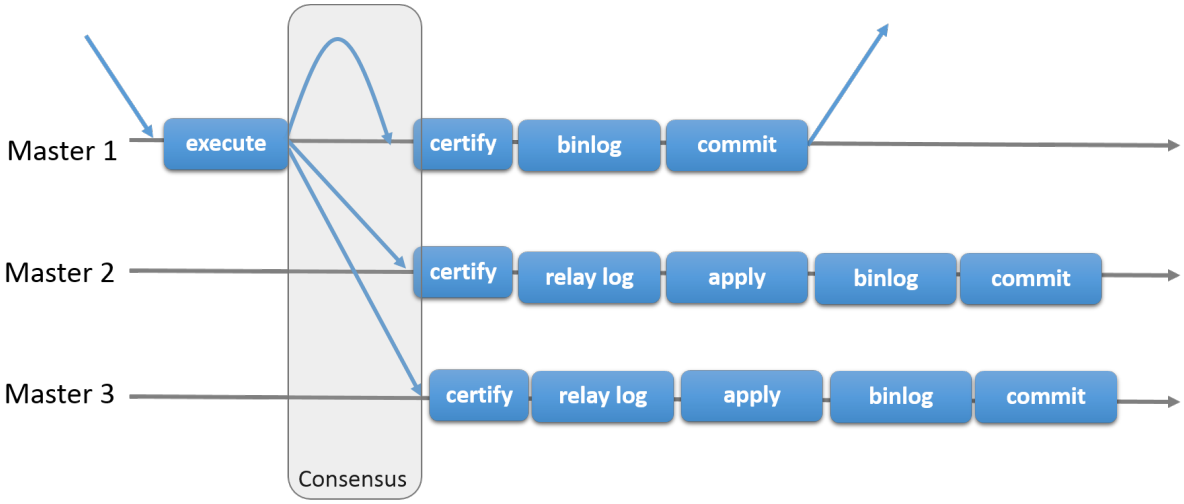
18.1.1.2 组复制

组复制可以实现复制系统的容错功能。组中各节点之间通过信息传递来交互通信，这个通信层提供了一系列的保证，例如信息原子性、信息完全按序投递等。这些强有力的特性可以转换为非常有用的抽象化，从而可以构建更高级的数据库复制解决方案。

MySQL组复制技术建立在这些属性和抽象之上，并实现了可以在任意地方更新的多主模型复制协议。在本质上，一个复制组是多个服务器节点组成的，组中的每个节点都独立地执行事务。但是，所有读写(read-write，RW)事务的提交操作只能在被复制组验证检查之后才能进行，所有只读(read-only，RO)事务则可以立即提交，无需复制组的协调。换句话说，对于任意RW事务，需要复制组来决定它是否能提交，而不是由事务发起节点的单方面决定。更准确地说，当源节点准备提交一个写事务时，该节点将待写入的值和(用于)通信的写集合(被更新行的唯一标识符)进行原子性的广播。然后为该事务生成一个全局顺序号。最后，这意味着所有节点都按照完全相同的顺序收到完全相同的事务，所以所有节点都按照相同的顺序对数据进行相同的更新，因此，它们在组内能保持一致性。

但是，在不同节点上并发执行的事务之间可能会存在冲突。这些冲突会被探测到，因为在一个称之为*认证(certification)*的阶段中，通过检查并发事务之间的写集合(译注：在MySQL组复制中，写集合可以理解为每个写事务对行设置的行标识符)就能发现问题。如果不同节点上的并发事务更新了同一行，它们就是冲突的。在事务解析过程中有如下声明：全局顺序号在前的先提交并应用到所有节点上，顺序号在后的提交被打断(译注：换句话说，谁先提交谁就获胜)，因此被打断的事务会在源节点上回滚，并被其他节点丢弃。这实际上是分布式的优先提交胜利规则。

Figure 18.3 MySQL Group Replication Protocol



最后，MySQL的组复制技术是一种shared-nothing的复制架构，每个服务器节点上都具有完整的数据副本。

上图描述了MySQL组复制协议，相比MySQL复制(或者半同步复制)，你可以看到一些区别。注意，为了清晰起见，上图中缺少了一些基本的共识和Paxos相关信息。

18.1.2 组复制使用案例

组复制技术使你可以通过复制多个节点的系统状态来创建具有冗余的容错系统。所以，即使某些服务器节点故障了，但只要大部分节点正常，这个容错系统就能继续可用，虽然这会降低性能和伸缩性，但重点在于它仍然可用。故障的节点是被隔离且独立的，它们被组成员服务**(group membership service)**跟踪，该服务依赖于分布式故障探测机制，当组中成员离开组，或自动、意外宕机时都会发送信号。另外，还有一个分布式的恢复过程**(procedure)**来保证当服务器加入组中时，它们会自动更新数据以保证同步。当主节点故障时，没有必要进行故障切换，因为多主模型下可以到处更新的特性保证了即使单节点故障也不会阻塞写操作。因此，MySQL组复制保证了数据库服务的持续可用性。

必须要注意，即使单节点故障后数据库服务仍然可用，但原来连接到该故障节点的客户端必须重定向到其他节点上。这个问题不是组复制需要解决的，通常连接器、负载均衡器、路由或其他形式的中间件更适合处理这类问题。

总而言之，MySQL组复制提供了一个高可用、高弹性且可靠的MySQL服务系统。

18.1.2.1 使用场景

下面是组复制的一些典型应用场景：

- *可伸缩的复制* - 需要非常流畅的复制架构环境，可以动态的增加、减少服务器节点数量且尽可能少地带来副作用。例如，数据库服务部署在云上。
- *高可用的分片(sharding)* - Sharding是实现写伸缩(scale-out)非常流行的方法。使用MySQL组复制来实现高可用的sharding，每一个分片对应一个复制组。
- *可切换的master-slave复制* - 在某些特定环境下，使用单主模型会出现单主瓶颈和单点故障的问题。而写入整个组，可能会有可以有更好的可扩展性。
- *自动化集群系统* - 此外，你可以部署MySQL组复制，只为使用它的自动切换能力(前文和本章已经描述过了)。

18.1.3 组复制细节

本节详细解释组复制中的一些服务。

18.1.3.1 故障探测服务

有一个故障探测机制可以发现并报告那个节点是沉默(译注：即不再心跳)的或者认为是宕机的。在更高层面上看(译注：上层服务的意思)，故障探测器是一个提供节点是否死机(或怀疑死机)信息的分布式服务。稍后，当复制组同意了这次故障报告，该组将认为该节点确实已经故障。这意味着，组内剩余的成员会进行协调并将故障节点排除在外。

当节点"哑"了后就会触发故障怀疑机制。当节点A在一段时间内接收不到节点B的消息后，表示超时，此时将触发怀疑并报告怀疑给复制组。

如果一个节点从组中隔离后，它将怀疑所有节点都是故障的。由于它无法和复制组达成一致(因为达不到法定票数)，它的怀疑是没有结果的。如果一个节点以这种方式从组中隔离，它将无法执行任何本地事务。(译注：这是防止非死机隔离的主机还继续执行事务)

18.1.3.2 组成员服务

MySQL组复制依赖于组成员服务。该服务内置在组复制插件中。它定义哪些节点在线且参与了组。在线的节点列表通常被称为*视图(view)* 。因此，在任何一个时间点，组中所有节点获取到的组中在线节点列表都是完全一致的。

组中节点不仅需要就事务是否提交达成一致，还必须对当前视图达成一致。因此，如果复制组同意一个新节点加入组，那么将会自动重新配置组成员并触发视图更新。反之，节点退出组也一样。(译注：总之，视图是动态维护的，能实时地反映出组中在线成员的信息)

注意，如果一个成员是自愿离开的，在离开前，它首先会启动动态组成员的重新配置。这会触发一个过程：组中所有成员就新视图达成一致后，该成员才能如愿离开组(译注：节点自愿离开组时，组的大小会改变，换句话说，重新配置成功后，等同于这个将要离开的成员从来都不曾出现在这个组中)。

但是，如果成员是非自愿离开的(即意料之外的，例如宕机、网络故障)，则故障探测机制会检测到这个问题并报告给组，复制组将重新配置视图，并将故障节点从视图中移除，如前所述，这需要组中大多数成员就新视图达成一致。如果复制组无法达成一致(例如在线节点数量达不到大多数的要求)，将无法动态重新配置，且为了防止脑裂而进行阻塞。这意味着需要管理员的介入，并修复该问题。(译注：节点非自愿离开组时，这个节点会从视图中移除，并重新配置组大小，但如果大多数节点一次性离开组，会导致重新配置组大小的操作无法完成，从而阻塞后续操作)

18.1.3.3 容错服务

MySQL组复制建立在Paxos分布式算法的实现上，以提供节点间的分布式协调。正因如此，它要求组中大多数节点在线才能达到法定票数，从而做出决定。这直接影响了系统正常提供服务的可容忍故障数量。对于要容忍 f 个节点故障的系统，需要总节点数量达到 n = 2 x f + 1。(译注：例如5节点的复制组最多允许2个节点故障，7节点的复制组最多允许3个节点故障)

这意味着要容忍任何一次故障，组中必须至少有3个节点。这种情况下，如果一个节点故障了，剩余两个节点(三分之二)仍能达到"大多数"的要求，所以允许组复制自动继续制定决策并取得进展。但是，如果第二个节点再*非自愿*故障(组中将只剩一个节点)，那么组复制将被阻塞，因为达不到"大多数"来做决定。

下表解释了上面的公式：

组的大小	大多数数量	故障容忍数量
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

下一节将解释组复制的技术方面内容。

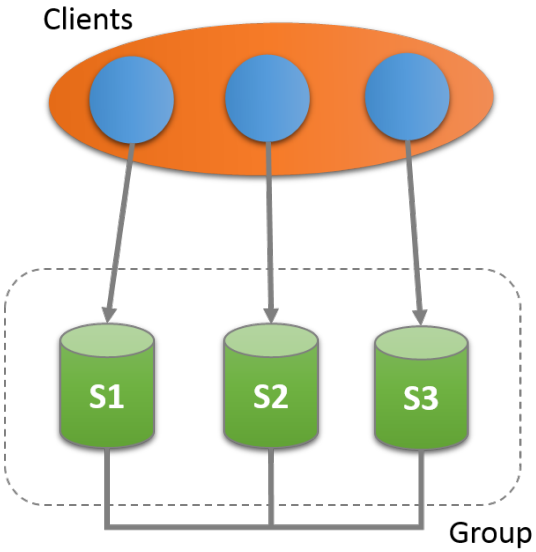
18.2 组复制使用入门

MySQL组复制是通过MySQL插件形式提供的，复制组中的所有节点都需要安装和配置该插件。本节介绍创建一个最少3节点的组复制所需的详细步骤。

18.2.1 部署单主模型的组复制

组中的每个mysql实例都可以运行在一个独立的物理主机上，也可以运行在同一主机上。本节介绍如何在同一主机上创建一个包含3实例的组复制。这意味着需要3个数据目录，每个实例一个数据目录。

Figure 18.4 Group Architecture



本节介绍如何获取并部署带有组复制插件的MySQL Server，在创建复制组之前如何配置每个实例，以及如何通过Performance Schema monitoring来验证配置是否正确。

18.2.1.1 部署MySQL实例

第一步是部署3个MySQL Server实例。组复制插件是内置在MySQL 5.7.17 + 版本中的。更多关于MySQL插件的内容，参见 Section 6.5, “MySQL Server Plugins”。此处假定MySQL Server已经下载好且解压在名为 mysql-5.7 的目录下。由于3个实例配置在同一主机上，因此每个MySQL Server实例都需要指定一个单独的数据目录。在名为 data 的目录下创建这些数据目录并逐一初始化。

```
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
```

`data/s1`, `data/s2`, `data/s3` 是已经初始化的数据目录，包含了mysql系统数据库和表等。更多关于初始化的内容，见 Section 2.10.1.1, “Initializing the Data Directory Manually Using mysqld”。

警告

不要使用在生产环境中使用 `--initialize-insecure` 进行初始化，此处使用该选项仅是出于简化操作的原因。

18.2.1.2 配置组复制实例

本节解释要使用组复制所需要进行的设置。关于组复制的限制，见 Section 18.7.2, “Limitations”。

组复制节点的选项设置

要使用MySQL的组复制，必须要正确配置MySQL实例。建议将配置保存在 `my.cnf` 文件中。除非另有说明，否则下面的配置操作都是称为 *s1* 实例对应的配置。下面是一个节点的配置示例：

```
[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
```

这些配置指定了MySQL Server使用的数据目录为之前创建的/data/s1，并且指定了该实例的监听端口。

注意

这里使用的24801端口是非默认端口，因为是在同一主机上配置3个实例。如果3个实例是配置在不同主机上，则无需手动指定端口号。

复制框架

下面的选项是开启MySQL组复制之前所需的配置。

```
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
```

上面配置了server_id为1，并启用了基于全局事务ID(GTID)的复制类型，并指定使用表而非文件来存储复制相关的元数据。此外，还开启了基于行格式(row-based)的二进制日志，并禁用了二进制日志的事件checksum。更多关于组复制的限制，见 Section 18.7.2, “Limitations”。

组复制相关的设置

此时 `my.cnf` 中的配置已经能让MySQL复制运行起来。下面是配置组复制的部分选项。

```
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
```

注意

上面的组复制变量中使用了前缀 `loose-` ，这表示启动MySQL实例时如果没有成功加载组复制插件也继续启动MySQL。(译者加：上面几项设置off是为了后面手动在第一个节点上启动这几个配置，强烈建议如此)

- *第 1 行*表示节点必须收集每个事务的写集(write set)，并使用 XXHASH64 哈希算法将其编码为hash值。
- *第 2 行*用来告诉插件，有一个名为"aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"的组需要加入，或者需要创建。
- *第 3 行*告诉插件，在启动MySQL实例时不要自动启动组复制功能。
- *第 4 行*告诉插件，该实例使用 127.0.0.1 或 localhost 以及24901端口作为组中成员之间的通信地址。选项 `group_replication_local_address` 配置的本地地址必须能和组中其他所有成员通信。例如，如果组中每个MySQL实例都单独安装在一台物理机上，那么必须不能指定回环地址。选项 `group_replication_local_address` 的推荐端口是33061，但因为这里我们将3个实例配置在一台主机上，所以使用24901-24903这3个端口。
- *第 5 行* 告诉插件，如果节点要加入该组，需要联系这些主机。这些是种子成员(seed members)，当节点要连接该组时需要使用这些成员。在某节点要加入组时，需要联系其中的一个(种子)，然后请求组重新配置成员列表，以允许在组中接受该节点。注意，不需要在该选项中列出组中的所有成员，只需要列出节点希望加入组时应该联系的服务器列表。
  组中第一个节点启动时不需要考虑这个选项，因为它用来启动复制组，它是组中的第一个成员，它负责引导组。第二个节点加入组时，需要询问组中唯一已启动的节点，然后组被扩展。第三个节点加入组时，需要询问前面两个节点中的任何一个，然后组再次扩展。如果还有第4个、第5个新节点要加入，则需要询问这3个节点中的任何一个，更多节点依此类推。

警告

当多个节点在同一时刻加入组时，需要确保它们指向的种子成员已经在组中。不要指向那些也正在加入组的种子成员，因为联系它们时，它们自身可能还没成功加入组。

首先启动引导成员，并让它来创建组，这是一个好习惯。然后将它作为其他成员加入组时的种子成员。

不应该在创建组的时候同时加入多个成员。虽然可能允许这样的操作，但因为它们是争抢的行为，这可能会导致加入组的行为终止或超时。

- *第 6 行* 告诉插件是否要引导该组。

重要

该选项在任何时候都必须只能在一个节点上使用，通常用在第一次启动组的节点上(或者整个组下线，然后再重新上线的情况)。如果多次引导组，例如多个节点上配置了该选项，这可能会人为地创建一个脑裂场景，因为这时会存在名称相同但实际不同的两个或多个组。应该在第一个节点加入组后禁用该选项。

配置组中的其他节点是类似的。你只需修改几项指定的选项 (例如 server_id, datadir, group_replication_local_address)。后文将会对此说明。

18.2.1.3 用户凭证

在成员需要加入组之前，组复制使用异步复制协议进行分布式恢复、组成员信息的同步。分布式恢复过程依赖于一个名为 group_replication_recovery 的复制通道，它用于成员之间传输事务。因此，你必须设置一个权限正确的专门用于复制的用户，以便组复制在成员间建立正确的用于分布式恢复的复制通道。

启动实例：

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

创建一个具有 REPLICATION-SLAVE 权限的MySQL用户。该操作不应该写进binlog中，以免被其他节点复制走(译注：实际上，个人建议写进binlog让其他节点复制走，否则第一个种子节点离组后，再加入新成员时讲无法建立通道。本文档的方法是在每个节点上手动创建复制用户)。下面的示例中，建立的用户是 *rpl_user* ，密码是 *rpl_pass* 。当配置你自己的服务器时，应该使用合适的用户名和密码。连接到 s1 实例后，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)
```

配置了上述用户后，使用 CHANGE MASTER TO 语句指定服务器下次恢复状态时所使用通道 group_replication_recovery 的凭据。执行下述语句，替换 *rpl_user* 和 *rpl_pass* 为你自己创建的用户名和密码。

```
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)
```

分布式恢复是节点加入组时的第一个步骤。如果这些凭据没有正确设置，节点将无法执行恢复的过程，从而会与其他成员重新开始同步，因而最终加组失败。类似地，如果成员通过节点的hostname无法正确识别其他成员，恢复过程也会失败。所以建议运行MySQL的节点合理配置具有唯一性的hostname。 该hostname可以在 performance_schema.replication_group_members 表中的Member_host列进行验证。如果组中多个成员使用一个默认操作系统设置的hostname，有一定可能会无法正确解析成员的地址，从而导致新节点加组失败。这种情况下，可以使用report_host来配置每一个节点的唯一外部化hostname。

18.2.1.4 登陆复制组

在 s1配置完成并启动后，可以安装组复制插件了。连接到该实例，输入下述语句：

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

要检查插件是否安装成功，使用 SHOW PLUGINS; 语句，安装成功的输出结果看上去大概是这样的：

```
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，需要指示s1实例引导这个组，然后启动组复制。引导操作应当只在一个节点上进行，建议在启动组中第一个成员时引导。这就是为什么引导组的配置选项没有保存在配置文件中。如果将其配置在配置文件中，每当重启该节点时都将自动引导具有完全相同名称的第二个组。同样的原因，将其设置为ON来停止和重启插件。

```
SET GLOBAL group_replication_bootstrap_group=ON;
START GROUP_REPLICATION;
SET GLOBAL group_replication_bootstrap_group=OFF;
```

当 `START GROUP_REPLICATION` 语句返回后，就表示组已经启动了。你可以检查该组是否已经创建，组中是否已有一个成员：

```
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)
```

该表中的信息表明该组中有一个唯一标识符为 `ce9be252-2b71-11e6-b8f4-00212844f856` 的成员，它的状态是 `ONLINE` ，主机名为 `myhost` (译注：这不一定真的是该主机的主机名，可能是`report_host`设置的主机名)，监听客户端连接的端口为 `24801`。

为了演示该主机确实在组中，且能处理请求，现在创建一张表，并插入一些内容：

```
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` 的内容以及二进制日志。

```
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)
```

如上所见，数据库和表对象已经创建，且它们对应的DDL语句已经写入到二进制日志中。此外，数据也已经插入到了表中且写入了二进制日志中。二进制日志中条目的重要后面会说明，因为新成员要加入组并赶上组的数据时，需要进行分布式恢复，此时会执行二进制日志中的记录。

18.2.1.5 向组中加入新实例

此刻组中已经有一个成员 s1，且该成员中还有一些新数据。现在可以向组中加入前面已经配置好的两个实例来扩展组了。

加入第二个实例

为了增加第二个实例 s2，首先配置它的配置文件。该配置文件类似于前面 s1的配置文件，除了数据目录、s2的端口、server_id。

```
[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
```

类似于实例 s1的操作过程，使用配置文件启动实例 s2。

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

然后按照下面的过程配置恢复凭据。这些命令和前面设置实例 s1时是一样的，因为用户名是组中共享的。在 s2上执行如下语句：

```
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，并开始节点加入组的过程。

```
mysql> INSTALL PLUGIN group_replication SONAME 'group_replication.so';
Query OK, 0 rows affected (0,01 sec)
```

加入 s2 到组中：

```
mysql> START GROUP_REPLICATION;
Query OK, 0 rows affected (44,88 sec)
```

不像前面的步骤都和配置 s1实例时都一样，这里有一个不同之处，你不能在启动组复制之前执行语句 `SET GLOBAL group_replication_bootstrap_group=ON;` ，因为组已经被实例 s1 创建并引导完成了。所以 s2 实例只需加入到已存在的组中即可。

再次查看表 `performance_schema.replication_group_members` ，检查 s2 实例是否在组中，且状态为*ONLINE* 。

```
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        |
+---------------------------+--------------------------------------+-------------+-------------+---------------+
```

由于 s2 也被标识为 ONLINE，说明它已经自动赶上了 s1 实例上的数据。执行下面的语句检查 s2 实例是否确实已经和 s1 保持同步了。

```
mysql> SHOW DATABASES LIKE 'test';
+------------------+
| Database (test) |
+------------------+
| test             |
+------------------+

mysql> SELECT * FROM test.t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 | Luis |
+----+------+
```

```
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                                                                |
+--------------+------+---------------+-----------+-------------+-----------------------------------------------------------------------+
```

可见，第二个实例已经添加到了组中，且已经自动从实例 s1 中复制了数据。根据分布式恢复过程，这意味着在 s2 成功加入组并标记为 ONLINE 之前， s2实例已经联系过 s1 实例并自动从中抓取了缺失的那部分数据。换句话说，它拷贝了 s1 实例的二进制日志中的自己缺失的那部分事务，所拷贝事务的的结束位置是它加入组那一刻的位置。

向复制组中添加更多的实例

再向组中添加其他实例，操作和刚才添加实例 s2 是基本类似的，除了之前提到的需要更改的一些配置。总结一下所需要执行的命令：

1. 提供配置文件。

```
[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MySQL实例

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

3. 配置凭据恢复通道 group_replication_recovery channel。

```
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它。

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

到此为止，s3 实例已经启动并正在运行，它会试图加入组，并自动抓取组中它所缺失的那部分数据。通过检查表 `performance_schema.replication_group_members` 来确认是否一切OK。

```
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       |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
```

执行下面的查询看看 s3 上的数据是否和 s1 或 s2 实例一样。

```
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                                                            |
+--------------+-----+----------------+-----------+-------------+-------------------------------------------------------------------+
```

18.3 监控组复制

使用 Perfomance Schema 中的表来监控组复制，假定你的MySQL编译时已经启动了 Performance Schema 表。组复制将添加如下两张 P_S 表：

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

下面这些已存在的 P_S 复制表同样也显示一些组复制的信息。

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

组复制插件创建的通道名称为：

- `group_replication_recovery` - 该通道在分布式恢复阶段进行复制。
- `group_replication_applier` - 该通道在组中有新写入操作时进行复制。该通道是组中有新事务流入时使用的通道。

下面一些小节中将描述这些表中可以获取到的信息。

18.3.1 成员状态

复制组中的每个成员都要验证并应用组提交的事务。关于验证者(certifier)和应用者(applier)相关的统计数据，有助于理解应用者队列(applier queue)是如何增长的、检测了多少次冲突、检测到多少个事务、哪些事务是到处提交的等等问题。`performance_schema.replication_group_member_stats` 表提供了以下认证相关信息：

Table 18.1 replication_group_member_stats

字段	描述
Channel_name	组复制通道的名称。
Member_id	当前连接到的节点的UUID。组中每个节点的UUID值都必须唯一，也正因为如此，它也提供了一种key。
Count_Transactions_in_queue	冲突检测队列中事务的数量。一旦探测到冲突，且通过了检查，则它们也会排队后被应用和提交。
Count_transactions_checked	已检测到的冲突事务数量。
Count_conflicts_detected	未通过冲突检查的事务数量。
Count_transactions_validating	冲突检测数据库的大小(根据该数据库对每个事务进行确认)
Transactions_committed_all_members	显示在当前视图中所有成员都成功提交的事务。该字段会在固定的时间间隔内更新。
Last_conflict_free_transaction	显示最后检测到的无冲突事务的事务ID。

这些字段对于监控连接组中成员的性能很重要。例如，假设组中某成员有些延迟，它没有赶上组中其他成员的数据。这种情况下，你可能会看到队列中有大量事务。根据这些信息，你可以决定是将该成员踢出组还是延迟组中其他成员处理事务，以便减少队列中的事务数量。这些信息同样可以帮助你决定如何调整组复制插件的流程控制。

18.3.2 组成员

该表用于监控当前视图所跟踪的节点状态。或者换句话说，作为复制组的一部分，这些节点会被组成员服务跟踪。

Table 18.2 replication_group_members

字段	描述
Channel_name	组复制的通道名称。
Member_id	成员节点的 UUID。
Member_host	成员的地址。
Member_port	用于成员间通信的监听端口。
Member_state	成员的状态(可能的状态值`ONLINE`, `ERROR`, `RECOVERING`, `OFFLINE` 或 `UNREACHABLE`)

更多关于 `Member_host` 值的信息以及它对分布式恢复过程的影响，见 Section 18.2.1.3, “User Credentials”.

18.3.3 连接状态

当连接到组时，该表中的一些字段显示组复制相关的信息。例如，已从组中接收到并在应用者(applier)队列(relay log)中排队的事务。

Table 18.3 replication_connection_status

字段	描述
Channel_name	组复制通道的名称。
Group_name	复制组的名称。通常它是一个有效的UUID值。
Source_UUID	组的标识符。它类似于组的名称，它用做组复制期间产生的所有事务的UUID。
Service_state	显示该成员是否是组的一部分。可能的值有 {ON, OFF 和 CONNECTING}。
Received_transaction_set	该成员已经接收到的GTID集。

18.3.4 applier状态

也可以通过普通的表 replication_applier_status 来获取组复制相关通道的状态和线程信息。如果有很多不同工作线程正在应用(执行)事务，该worker表同样可以用来监控每个工作线程正在做什么。

Table 18.4 replication_applier_status

字段	描述
Channel_name	组复制通道的名称。
Service_state	显示应用(applier)服务的状态是ON 还是 OFF。
Remaining_delay	显示是否有applier被延迟。
Count_transactions_retries	应用事务的重试次数。
Received_transaction_set	该成员已经接收到的GTID集。

18.3.5 节点状态

在每次视图发生改变时，`replication_group_members` 表会随之更新。例如，组配置被动态更改。此时，节点之间会交换它们的元数据信息并自我同步，然后协调达成一致。

组中节点可能有多种状态。如果节点之间能正常通信，则所有节点的报告信息都是相同的。但如果发生了网络分裂，或节点离开了组，则可能会报告不同的信息，这依赖于被查询的是哪一个节点。注意，如果节点已经离开了组，那么显然它不能报告其他节点相关的最新信息。如果发生了网络分裂，以至于丢失了法定票数，则各节点将无法进行协调。因此，它们无法猜测其他节点的状态是什么。因此，它们不会报告它们猜测出来的状态，而是会报告节点无法到达。

Table 18.5 Server State

字段	描述	组同步状态
ONLINE	该节点已经准备好一切，可供客户端连接，并可以执行事务。	Yes
RECOVERING	该节点正在加入组，它当前正处于分布式恢复阶段，接收来自组的状态信息。	No
OFFLINE	插件已加载，但还不是任何组中的成员	No
ERROR	无论是恢复阶段还是应用阶段发生错误，都会出现该状态。	No
UNREACHABLE	当本地故障探测器怀疑该节点不可到达时，将显示该状态。可能的原因是目标宕机、非自愿离开组。	No

注意，组复制是非同步的，但会达到最终的同步(译注：即最终一致性，强一致性)。更准确地说，事务按照相同的顺序投递到所有的成员上，但是它们每个节点对事务的执行非同步的，意味着在允许事务的提交后，每个成员按照它们自己的步调来执行事务。

18.4 组复制操作

本节介绍部署组复制的不同模式，说明管理组的常用操作并提供有关如何调整组的内容。

18.4.1 部署多主或单主模型

组复制有以下两种模型：

- 单主模型(single-primary)
- 多主模型(multi-primary)

默认模型为single-primary。组中部署不同的模型是不现实的，例如一个节点配置为单主模型，而另外一个节点配置为多主模型。要想切换工作模型，需要整个组(而非节点)都重启并应用对应模型的配置。无论是哪种工作模型，组复制都不会处理client端的故障转移，这种问题只能在应用程序自身来处理，或者使用连接器或中间件(例如路由、代理)来处理。

当部署为多主模型时，将会检查语句以确保它们能兼容这种模型。以下是部署为多主模型时需要做的检查：

- 当在SERIALIZABLE隔离级别下执行事务时，那么当它自身和组进行同步时，提交将失败。
- 如果事务涉及操作具有外键级联约束的表时，那么当它自身与组进行同步时，提交将失败。

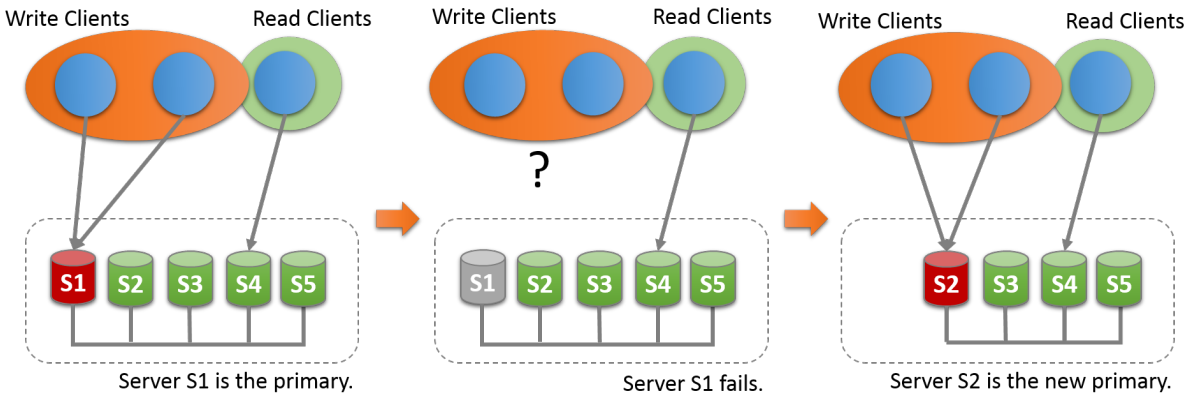
可通过设置选项group_replication_enforce_update_everywhere_checks 为 FALSE来禁用这些检测。如果部署为单主模型，该选项必须设置为 FALSE。

18.4.1.1 单主模型

该模型下只有一个主节点被设置为read-write，组中剩余所有节点都设置为read-only(如 super-read-only)。这些设置是部署为单主模型时自动设置的。.

该模型下的主节点一般是第一个启动用来引导组的节点，其余所有后续加入组的节点会自动学习该设置，并设置为read-only。

Figure 18.5 New Primary Election



但使用单主模型时，会禁用一些多主模型下的检测，因为系统强制某一时刻只允许单节点写操作。例如，允许修改具有外键级联约束的表，而这在多主模型下是不允许的。当主节点故障后，主节点自动选举机制将会自动选出另一个主节点。在选举主节点时，将查找新的成员视图，并根据 group_replication_member_weigth 的值排列出潜在的主节点，如果所有成员的MySQL版本都相同，则权重最高的节点被选举为下一个主节点，如果权重值相同，则根据字典顺序对它们的server_uuid进行排序，然后选出列表中的第一个节点作为下一个主节点。当选举出新的主节点后，该主节点将自动设置为 read-write，其他节点继续作为slave，且保留设置为read-only。

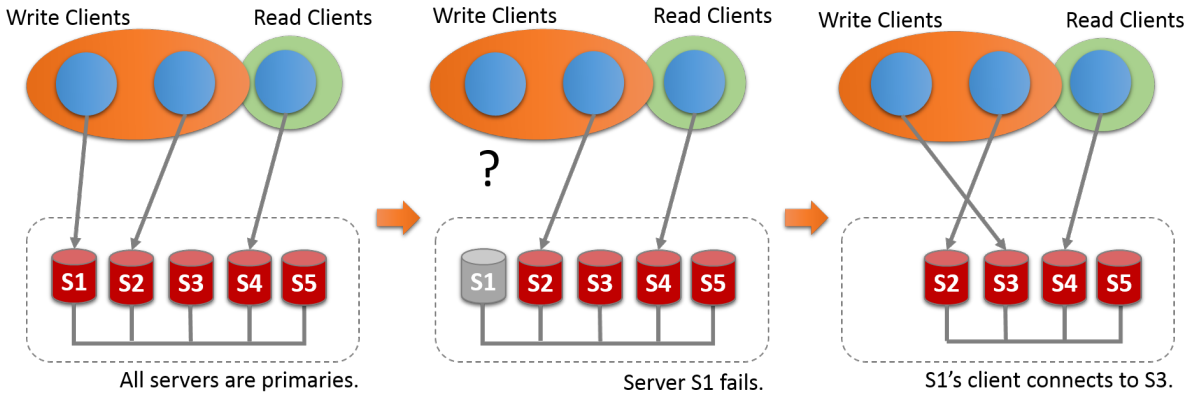
如果MySQL版本不支持group_replication_member_weigth，那么将根据 server_uuid的字典顺序来选举新节点。

在将客户端应用程序重新路由到新节点之前，等待新节点应用完 relay-log是一个好习惯。

18.4.1.2 多主模型

在多主模型下，没有单主的概念，没有必须进行选举的过程，因为多主模型下没有节点扮演特殊的角色。

Figure 18.6 Client Failover



当节点加入组时，所有节点都会设置为 `read-write`。

18.4.1.3 查找主节点

下面的示例演示了如何查询单主模型下当前的主节点。

```
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)
```

或者：

```
mysql> SHOW STATUS LIKE 'group_replication_primary_member';
```

18.4.2 调整恢复过程

当一个新成员要加入复制组，它会连接组中一个合适的信息供应者(donor)，并从它身上抓取它所缺失的那部分数据。这是组复制中的一个关键组件，它是可容错的、可配置的。下一节将解释恢复过程是如何工作的以及如何调整相关设置。

选择Donor

Donor是从组中在线成员中随机选择的。这种方式可以降低某成员被多个同时加入的成员选中为Donor的几率。

如果连接到选定的donor失败，则会自动尝试选择一个候选donor来连接。当连接失败次数达到最高限制数量后，恢复过程将失败并报错。

注意

donor是从当前视图中的在线节点列表中随机挑选的。

Donor自动切换

另一个恢复过程中需要关注的关键点是对待失败的处理方式。组复制为此提供了健壮的错误探测机制。在组复制的早期版本中，当新成员联系donor时，恢复过程只能探测到因为认证问题或其他一些问题而导致的连接错误。对于这些问题，会切换到一个新的donor上，并因此而尝试和另一个成员建立连接。

以下这些错误也会选择另一个donor：

- *Purged data scenarios* - 当donor上已purge过一段日志，但新节点上没有这段日志对应的数据，新节点就无法获取完整得到数据，这时候恢复过程就会报错，并重新选择一个新的donor。
- *Duplicated data* - 当待加入的成员已有一部分数据，但和选中的donor上准备复制到自身的数据有冲突时，恢复过程会发生错误。这可能是因为待加入成员上存在一些不合理的事务造成的。
  有人可能会说，这样的问题应该要直接失败而不是切换到另一个donor，但是有时候有一些奇怪的复制组，可能会有其他成员共享了冲突的事务。基于此，发生上面的错误后，恢复过程会选择另一个donor而不是直接失败。
- *Other errors* - 如果恢复过程的线程失败(receiver或applier线程)，也会错误并选择另一个donor。

重连Donor

恢复过程的传输依赖于donor上的二进制日志以及MySQL复制框架，因此传输过程中的任何一个错误都回导致receiver和applier线程错误。这种情况下，donor切换具有重试功能，就像常规复制一样。

Donor重试次数

待加组成员重试donor的次数为10。可通过插件变量`group_replication_recovery_retry_count`修改重试次数。下面的命令设置最大重试次数为10。

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

Sleep Routines

插件变量 `group_replication_recovery_reconnect_interval` 用于设置两次连接donor的时间间隔(译注：见下面的解释)。该变量默认值为60秒，可动态修改。下面的命令设置该时间间隔为120秒。

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

但注意，这个时间间隔不是连续两次选**donor**时的时间间隔，而是所有**donor**都轮遍了后还没有选好**donor**时，恢复过程进入睡眠的时间。所以，它影响的不是在同一个轮询内选**donor**的时间间隔，因为每次所选的**donor**都是不同节点。

18.4.3 网络分裂(网络分区, Network Partitioning)

无论什么时候，组中发生了一个需要做复制的改变后，复制组都要对此改变达成一致。例如产生的常规事务就如此要求。此外，对于组成员变更以及其他一些内部消息也要求组内保持一致性。对于一个决定，要达到组内共识，需要组内大多数成员都同意此决定。当组内在线成员数达不到"大多数"的要求，也就是达不到法定票数，则这个组中所有需要做决定的操作都会被阻塞。

当多个成员非自愿离开组时，这表示组的大多数节点突然间离开组，而非自愿离开组的成员的法定票数会丢失(译注：等价于弃权)，这样就无法满足大多数的要求。例如，一个5节点的组，如果3个节点突然沉默(不心跳)，这样剩下2个节点就不可能达到5的大多数(3)。实际上，3节点突然离组或者网络分裂将2个节点隔离出去，这两个节点是无法通告任何消息的，因此也不能做任何自动配置、协调操作(译注：失去大多数节点后，剩余的少数节点永远无法自动调整，组的大小永远都是5，这样2个节点永远也满足不了大多数)。

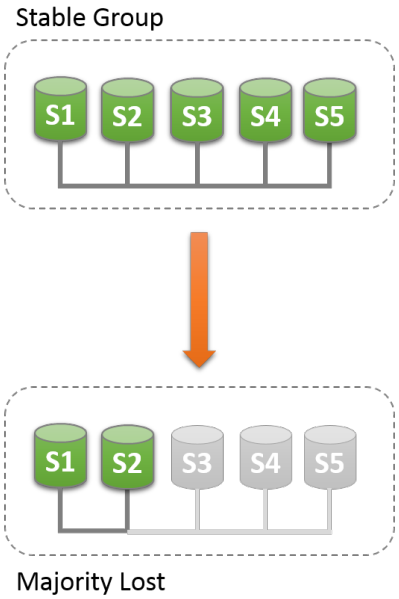
另一方面，如果节点自愿离开组，它们会告诉组让它们重新配置组成员。实际上这意味着节点告诉组，我要离开了，你们重组吧。例如上面的5节点组，突然3个要自愿离开，在它们相继离开后，剩余的2个节点重新配置后，它们两独自组成一个组。这时的组成员总数是2而不是5，所以组中的2是组的全部，意味着达到大多数的要求。(译注：自愿离开时，会自动配置组大小，配置好之后才会成功离开组，这样5节点的组离开3个自愿退出的节点后 ，组的大小为2，而不是5)

下面将介绍出现网络分区而无法满足大多数要求时如何解决。

18.4.3.1 探测分区

performance schema 中的 `replication_group_members` 表可以显示当前视图中每个节点的状态(以本节点为观察角度)。大多数时候，复制组中不会出现分区现象，所以该表显示的组中成员信息是完全一致的。也就是说，每个节点的状态都是经过当前视图中所有节点协商后同意的。但是，如果出现了网络分裂，那些无法联系到的主机的法定票数会丢失，它们在表中显示的状态将是 `UNREACHABLE`。该信息会被内置在组复制中的本地故障探测器向外通告。

Figure 18.7 Losing Quorum



为了理解这种类型的网络分裂，下面将解释上图的场景，初始时5节点正常工作，然后突然做出改变，使得组中只剩2个在线节点。

因此，先假定有一个包含5个节点的组：

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

初始时，所有节点都正常工作，它们也都能正常地通信。你可以连接到 s1 上通过它 performance schema 中的 `replication_group_members` 表来验证之。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       |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
```

但随后在 s3、s4 和 s5 上发生了意料之外的灾难性事故。几秒之后，再去查看 s1 实例上的表 `replication_group_members` ，结果发现 s1 自身是 ONLINE 的，但 s3、s4 和 s5 却 `UNREACHABLE`，如下。此时组将无法自动重新配置它们自身来调整组中的成员信息，因为无法满足组的大多数要求。

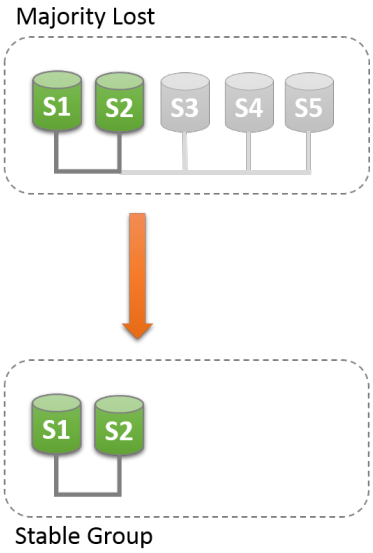
```
mysql> SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+--------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
| 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  |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
```

该表显示 s1 现在处于一个没有外部干预就无法继续处理的组中，因为大多数节点都是 `unreachable` 状态。在这种特殊的情况下，组成员列表需要进行重置以便让祖复制继续工作。或者，你也可以停止实例 s1 和 s2 上的组复制(或直接停止它们的MySQL服务)，然后查出 s3、s4 和 s5 上发生了什么问题，最后重启组复制(或MySQL服务)。

18.4.3.2 疏通网络分裂导致的阻塞

组复制允许你通过强制配置的方式重置组成员列表。例如上面的情况， s1 和 s2 是组中仅剩的在线成员，你可以强制配置组使其只由 s1 和 s2组成。这需要检查 s1 和 s2 上的一些信息，然后再使用变量 `group_replication_force_members` 。

Figure 18.8 Forcing a New Membership



现在回到**s1**和**s2**是组中仅剩节点的情况。服务器**s3**、**s4**和**s5**意外地离开了组，要使服务器**s1**和**s2**继续运行，你需要强制配置一个只包含**s1**和**s2**的组。

警告

此处使用的 `group_replication_force_members` 应当将其认为是最后的补救手段。你必须非常小心地使用它，并只有在无法仲裁的情况下使用。如果使用不当，则可能会人为地造成脑裂或阻塞整个复制系统。

回想一下，目前复制系统被阻塞，当前的配置如下(由 s1 节点的本地故障探测器感知):

```
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  |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
```

第一件事就是获取s1和s2对端的地址(组通信的标识)。连接上s1和s2，如下是s1上获取的信息：

```
mysql> SELECT @@group_replication_local_address;
+-----------------------------------+
| @@group_replication_local_address |
+-----------------------------------+
| 127.0.0.1:10000                   |
+-----------------------------------+
1 row in set (0,00 sec)
```

连上 s2 做相同操作：

```
mysql> SELECT @@group_replication_local_address;
+-----------------------------------+
| @@group_replication_local_address |
+-----------------------------------+
| 127.0.0.1:10001                   |
+-----------------------------------+
1 row in set (0,00 sec)
```

当你知道了它们用于成员间通信的地址s1 (`127.0.0.1:10000`) 、s2 (`127.0.0.1:10001`)后，你可以使用它们中之一来注入一个新的组成员配置，然后覆盖已存在且失去仲裁能力的那个组成员配置。例如，在 s1 上操作：

```
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)
```

这表示强制配置新的组成员配置。然后，在 s1 和 s2 上查看表 `replication_group_members` 来验证组成员信息是否更改。首先在s1上执行：

```
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       |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
```

然后在 s2 上执行：

```
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       |
+---------------------------+--------------------------------------+-------------+-------------+--------------+
```

当强制配置新的组成员配置时，需要确保那些被排除在外的节点确实是已经停止工作的。在上面的场景中，如果 s3, s4 和 s5 并非真的是 unreachable 而是 ONLINE，这3个节点将自己组成一个分区，且这个分区还正好满足组的大多数要求。这种情况下，强制创建只包含 s1 和 s2 的组将会人为造成脑裂的问题。因此，在重新配置 s1 和 s2 为新的组成员之前，先确认其他节点是否确实故障是非常重要的，如果它们之中还有非故障节点，但你还想继续创建新的组成员配置，请手动停止这些节点。

18.5 组复制的安全问题

This section explains how to secure a group, securing the connections between members of a group, or by establishing a security perimeter using address whitelisting.

18.5.1 IP Address Whitelisting

The Group Replication plugin has a configuration option to determine from which hosts an incoming Group Communication connection can be accepted. This option is called `group_replication_ip_whitelist`. If you set this option on a server s1, then when server s2 is establishing a connection to s1 for the purpose of engaging group communication, then s1 first checks the whitelist before accepting the connection from s2. If s2 is in the whitelist, then s1 accepts the connection, otherwise s1 rejects the connection attempt by s2.

Note

By default, if not specified explicitly, the whitelist is automatically set to the private network addresses that the server has network interfaces on.

If you do not configure any whitelist, the server automatically sets the whitelist to the private networks that the server has an interface on. This means that a server, even if it has interfaces on public IPs, does not by default allow connections from external hosts.

Whenever the IP whitelist is set to AUTOMATIC, an entry in the error log can be emitted in such case, similar to:

```
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'
```

You can improve the security of the group further by manually setting the list of IP addresses permitted for group communication connections to come from. The list can be specified in CIDR notation or as simple IP addresses. A comma must separate each entry. For example:

```
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

The localhost IP address (127.0.0.1) is always added to the whitelist. If not explicitly, it is implicitly and automatically added.

18.5.2 Secure Socket Layer Support (SSL)

MySQL Group Replication supports both OpenSSL and YaSSL builds of MySQL Server.

Group communication connections as well as recovery connections, are secured using SSL. The following sections explain how to configure connections.

Configuring SSL for Recovery

Recovery is performed through a regular asynchronous replication connection. Once the donor is selected, the joiner establishes an asynchronous replication connection. This is all automatic.

However, a user that requires an SSL connection must have been created before the joiner connects to the donor. Typically, this is set up at the time one is provisioning a server to join the group.

```
donor> SET SQL_LOG_BIN=0;
donor> CREATE USER 'rec_ssl_user'@'%' REQUIRE SSL;
donor> GRANT replication slave ON *.* TO 'rec_ssl_user'@'%';
donor> SET SQL_LOG_BIN=1;
```

Assuming that all servers already in the group have a replication user set up to use SSL, you configure the joiner to use those credentials when connecting to the donor. That is done according to the values of the SSL options provided for the Group Replication plugin.

```
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';
```

And by configuring the recovery channel to use the credentials of the user that requires an SSL connection.

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

Configuring SSL for Group Communication

Secure sockets can be used to establish communication between members in a group. The configuration for this depends on the server's SSL configuration. As such, if the server has SSL configured, the Group Replication plugin also has SSL configured. For more information on the options for configuring the server SSL, see Section 7.4.5, “Command Options for Secure Connections”. The options which configure Group Replication are shown in the following table.

Table 18.6 SSL Options

Server Configuration	Plugin Configuration Description
ssl_key	Path of key file. To be used as client and server certificate.
ssl_cert	Path of certificate file. To be used as client and server certificate.
ssl_ca	Path of file with SSL CAs that are trusted.
ssl_capath	Path of directory containing certificates for SSL CAs that are trusted.
ssl_crl	Path of file containing the certificate revocation lists.
ssl_crlpath	Path of directory containing revoked certificate lists files.
ssl_cipher	Permitted ciphers to use while encrypting data over the connection.
tls_version	Secure communication will use this version and its protocols.

These options are MySQL Server configuration options which Group Replication relies on for its configuration. In addition there is the following Group Replication specific option to configure SSL on the plugin itself.

- `group_replication_ssl_mode` - specifies the security state of the connection between Group Replication members.

Table 18.7 group_replication_ssl_mode configuration values

Value	Description
DISABLED	Establish an unencrypted connection (*default*).

Value	Description
REQUIRED	Establish a secure connection if the server supports secure connections.
VERIFY_CA	Like REQUIRED, but additionally verify the server TLS certificate against the configured Certificate Authority (CA) certificates.
VERIFY_IDENTITY	Like VERIFY_CA, but additionally verify that the server certificate matches the host to which the connection is attempted.

The following example shows an example my.cnf file section used to configure SSL on a server and how activate it for Group Replication.

```
[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
```

The only plugin specific configuration option that is listed is `group_replication_ssl_mode`. This option activates the SSL communication between members of the group, by configuring the SSL framework with the `ssl_*` parameters that are provided to the server.

18.5.3 Virtual Private Networks (VPN)

There is nothing preventing Group Replication from operating over a virtual private network. At its core, it just relies on an IPv4 socket to establish connections between servers for the purpose of propagating messages between them.

18.6 组复制相关系统变量

以下都是和组复制插件相关的系统变量。每个配置选项都使用前缀 "group_replication"。

重要

尽管大多数变量都描述为可在实例运行过状态下动态修改，但大多数修改都只在重启组复制插件后才生效。那些不需重启组复制插件就能生效的变量都特地做了说明。

- `group_replication_group_name`

Introduced	5.7.17	
Command-Line Format	`--group-replication-group-name=value`	
System Variable	**Name**	`group_replication_group_name`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	string

该节点所属复制组的名称。值必须是一个有效的 UUID。

- `group_replication_start_on_boot`

Introduced	5.7.17	
Command-Line Format	`--group-replication-start-on-boot=value`	
System Variable	Name	`group_replication_start_on_boot`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	boolean
	Default	`ON`

是否在启动 `mysql` 实例时也启动组复制插件。

- `group_replication_local_address`

Introduced	5.7.17	
Command-Line Format	`--group-replication-local-address=value`	
System Variable	Name	`group_replication_local_address`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	string

以 `host:port` 格式显示的地址，该地址用于组成员间通信(特指其他成员连接本节点时的连接地址)，它必须能和组中所有节点通信，因为该地址会被组复制系统内部的XCOM消息中间件使用。

- `group_replication_group_seeds`

Introduced	5.7.17	
Command-Line Format	`--group-replication-group-seeds=value`	
System Variable	Name	`group_replication_group_seeds`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	string

A list of peer addresses as a comma separated list such as `host1:port1`,`host2:port2`. Each address is validated when starting Group Replication and the list must contain at least one valid address. Any invalid host names could cause `START GROUP_REPLICATION` to fail.

- `group_replication_force_members`

Introduced	5.7.17	
Command-Line Format	`--group-replication-force-members=value`	
System Variable	Name	`group_replication_force_members`
	Variable Scope	Global
	Dynamic Variable	Yes

Permitted Values	Type	string

A list of peer addresses as a comma separated list such as `host1:port1`,`host2:port2`. This option is used to force a new group membership, in which the excluded members do not receive a new view and are blocked. You need to manually kill the excluded servers. Any invalid host names in the list could cause subsequent `START GROUP_REPLICATION` statements to fail because they could block group membership.

- `group_replication_bootstrap_group`

Introduced	5.7.17	
Command-Line Format	`--group-replication-bootstrap-group=value`	
System Variable	Name	`group_replication_bootstrap_group`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	boolean
	Default	`OFF`

Configure this server to bootstrap the group. This option must only be set on one server and only when starting the group for the first time or restarting the entire group. After the group has been bootstrapped, set this option to `OFF`. It should be set to `OFF` both dynamically and in the configuration files. Starting two servers or restarting one server with this option set while the group is running may lead to an artificial split brain situation, where two independent groups with the same name are bootstrapped.

- `group_replication_poll_spin_loops`

Introduced	5.7.17	
Command-Line Format	`--group-replication-poll-spin-loops=value`	
System Variable	Name	`group_replication_poll_spin_loops`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	Type	integer
	Default	`0`
	Min Value	`0`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	Type	integer
	Default	`0`
	Min Value	`0`
	Max Value	`18446744073709547520`

The number of times the group communication thread waits for the communication engine mutex to be released before the thread waits for more incoming network messages.

- `group_replication_recovery_retry_count`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-retry-count=value`	
System Variable	Name	`group_replication_recovery_retry_count`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	`10`
	Min Value	`0`
	Max Value	`31536000`

The number of times that the member that is joining tries to connect to the available donors before giving up.

- `group_replication_recovery_reconnect_interval`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-reconnect-interval=value`	
System Variable	Name	`group_replication_recovery_reconnect_interval`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	`60`
	Min Value	`0`
	Max Value	`0`

The sleep time, in seconds, between reconnection attempts when no donor was found in the group.

- `group_replication_recovery_use_ssl`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-use-ssl=value`	
System Variable	Name	`group_replication_recovery_use_ssl`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	boolean
	Default	`OFF`

Whether Group Replication recovery connection should use SSL or not.

- `group_replication_recovery_ssl_ca`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-ssl-ca=value`	
System Variable	**Name**	`group_replication_recovery_ssl_ca`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	string

The path to a file that contains a list of trusted SSL certificate authorities.

- `group_replication_recovery_ssl_capath`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-ssl-capath=value`	
System Variable	**Name**	`group_replication_recovery_ssl_capath`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	string

The path to a directory that contains trusted SSL certificate authority certificates.

- `group_replication_recovery_ssl_cert`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-ssl-cert=value`	
System Variable	**Name**	`group_replication_recovery_ssl_cert`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	string

The name of the SSL certificate file to use for establishing a secure connection.

- `group_replication_recovery_ssl_key`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-ssl-key=value`	
System Variable	**Name**	`group_replication_recovery_ssl_key`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	string

The name of the SSL key file to use for establishing a secure connection.

- `group_replication_recovery_ssl_cipher`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-ssl-cipher=value`	
System Variable	Name	`group_replication_recovery_ssl_cipher`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	string

A list of permissible ciphers to use for SSL encryption.

- `group_replication_recovery_ssl_crl`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-ssl-crl=value`	
System Variable	Name	`group_replication_recovery_ssl_crl`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	string

The path to a directory that contains files containing certificate revocation lists.

- `group_replication_recovery_ssl_crlpath`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-ssl-crlpath=value`	
System Variable	Name	`group_replication_recovery_ssl_crlpath`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	string

The path to a directory that contains files containing certificate revocation lists.

- `group_replication_recovery_ssl_verify_server_cert`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-ssl-verify-server-cert=value`	
System Variable	Name	`group_replication_recovery_ssl_verify_server_cert`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	boolean

	Default	`OFF`

Make the recovery process check the server's Common Name value in the donor sent certificate.

- `group_replication_recovery_complete_at`

Introduced	5.7.17	
Command-Line Format	`--group-replication-recovery-complete-at=value`	
System Variable	**Name**	`group_replication_recovery_complete_at`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	enumeration
	Default	`TRANSACTIONS_APPLIED`
	Valid Values	`TRANSACTIONS_CERTIFIED`
		`TRANSACTIONS_APPLIED`

Recovery policies when handling cached transactions after state transfer. This option specifies whether a member is marked online after it has received all transactions that it missed before it joined the group (`TRANSACTIONS_CERTIFIED`) or after it has received and applied them (`TRANSACTIONS_APPLIED`).

- `group_replication_components_stop_timeout`

Introduced	5.7.17	
Command-Line Format	`--group-replication-components-stop-timeout=value`	
System Variable	**Name**	`group_replication_components_stop_timeout`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	`31536000`

Timeout, in seconds, that Group Replication waits for each of the components when shutting down.

- `group_replication_allow_local_lower_version_join`

Introduced	5.7.17	
Command-Line Format	`--group-replication-allow-local-lower-version-join=value`	
System Variable	**Name**	`group_replication_allow_local_lower_version_join`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`OFF`

Allow the current server to join the group even if it has a lower plugin version than the group.

- `group_replication_allow_local_disjoint_gtids_join`

Introduced	5.7.17	
Command-Line Format	`--group-replication-allow-local-disjoint-gtids-join=value`	
System Variable	**Name**	`group_replication_allow_local_disjoint_gtids_join`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`OFF`

Allow the current server to join the group even if it has transactions not present in the group.

Warning

Use caution when enabling this option as incorrect usage could lead to inconsistencies in the group.

- `group_replication_auto_increment_increment`

Introduced	5.7.17	
Command-Line Format	`--group-replication-auto-increment-increment=value`	
System Variable	**Name**	`group_replication_auto_increment_increment`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	`7`
	Min Value	`1`
	Max Value	`65535`

Determines the interval between successive column values for transactions that execute on this server instance.

- `group_replication_compression_threshold`

Introduced	5.7.17	
Command-Line Format	`--group-replication-compression-threshold=value`	
System Variable	**Name**	`group_replication_compression_threshold`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	integer
	Default	`1000000`

	Min Value	`0`
	Max Value	`4294967296`

The value in bytes above which (LZ4) compression is enforced. When set to zero, deactivates compression.

- `group_replication_gtid_assignment_block_size`

Introduced	5.7.17	
Command-Line Format	`--group-replication-gtid-assignment-block-size=value`	
System Variable	**Name**	`group_replication_gtid_assignment_block_size`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values (32-bit platforms)	**Type**	integer
	Default	`1000000`
	Min Value	`1`
	Max Value	`4294967295`
Permitted Values (64-bit platforms)	**Type**	integer
	Default	`1000000`
	Min Value	`1`
	Max Value	`18446744073709547520`

The number of consecutive GTIDs that are reserved for each member. Each member consumes its blocks and reserves more when needed.

- `group_replication_ssl_mode`

Introduced	5.7.17	
Command-Line Format	`--group-replication-ssl-mode=value`	
System Variable	**Name**	`group_replication_ssl_mode`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	enumeration
	Default	`DISABLED`
	Valid Values	`DISABLED`
		`REQUIRED`
		`VERIFY_CA`
		`VERIFY_IDENTITY`

Specifies the security state of the connection between Group Replication members.

- `group_replication_single_primary_mode`

Introduced	5.7.17	
Command-Line Format	`--group-replication-single-primary-mode=value`	
System Variable	**Name**	`group_replication_single_primary_mode`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`ON`

Instructs the group to automatically pick a single server to be the one that handles read/write workload. This server is the PRIMARY and all others are SECONDARIES.

- `group_replication_enforce_update_everywhere_checks`

Introduced	5.7.17	
Command-Line Format	`--group-replication-enforce-update-everywhere-checks=value`	
System Variable	**Name**	`group_replication_enforce_update_everywhere_checks`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	boolean
	Default	`OFF`

Enable or disable strict consistency checks for multi-master update everywhere.

- `group_replication_flow_control_mode`

Introduced	5.7.17	
Command-Line Format	`--group-replication-flow-control-mode=value`	
System Variable	**Name**	`group_replication_flow_control_mode`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	**Type**	enumeration
	Default	`QUOTA`
	Valid Values	`DISABLED`
		`QUOTA`

Specifies the mode used for flow control. This variable can be changed without resetting Group Replication.

- `group_replication_flow_control_certifier_threshold`

Introduced	5.7.17	
Command-Line Format	`--group-replication-flow-control-certifier-threshold=value`	
System Variable	Name	`group_replication_flow_control_certifier_threshold`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	`25000`
	Min Value	`0`
	Max Value	`2147483648`

Specifies the number of waiting transactions in the certifier queue that trigger flow control. This variable can be changed without resetting Group Replication.

- `group_replication_flow_control_applier_threshold`

Introduced	5.7.17	
Command-Line Format	`--group-replication-flow-control-applier-threshold=value`	
System Variable	Name	`group_replication_flow_control_applier_threshold`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	integer
	Default	`25000`
	Min Value	`0`
	Max Value	`2147483648`

Specifies the number of waiting transactions in the applier queue that trigger flow control. This variable can be changed without resetting Group Replication.

- `group_replication_ip_whitelist`

Introduced	5.7.17	
Command-Line Format	`--group-replication-ip-whitelist=value`	
System Variable	Name	`group_replication_recovery_ip_whitelist`
	Variable Scope	Global
	Dynamic Variable	Yes
Permitted Values	Type	string
	Default	`AUTOMATIC`

Specifies which hosts are permitted to connect to the group. By default set to `AUTOMATIC`, which permits connections from private subnetworks active on the host. Active interfaces on the host are scanned and those with addresses on private subnetworks are automatically added to the permitted list. Alternatively you can specify the permitted hosts using a comma separated list of IPv4 addresses or subnet CIDR notation. For example `192.168.1.0/24,10.0.0.1`.

Note

Address 127.0.0.1 is always permitted to connect, even when not specified using `group_replication_ip_whitelist`.

18.7 组复制的要求和限制

本节列出和解释了组复制相关的要求和限制。

18.7.1 组复制的要求

要使用组复制，你的MySQL节点必须满足以下条件：

基本要求

- **InnoDB存储引擎**：数据必须存储在事务型的 `InnoDB` 存储引擎中。事务以乐观形式执行，然后在提交前会检测冲突问题。如果有冲突，为了维护组中一致性，有些事务必须回滚。这意味着需要事务型的存储引擎。此外，`InnoDB` 存储引擎提供了一些额外的功能，它们结合组复制时能更好地管理和处理冲突。
- **Primary Keys**：每张需要被组复制的表都必须显式定义一个主键。主键在判断事务是否冲突扮演极其重要的角色：通过主键来准确识别每个事务中修改了表中的哪些行。
- **使用IPv4 地址**：MySQL组复制使用的组通信引擎组件只支持 IPv4。因此，必须使用IPv4的网络。
- **良好的网络性能**：组复制设计的初衷是部署在集群环境中，集群中的节点相互之间都非常近，因此除了网络延迟，网络带宽也会影响组复制。

节点配置上的要求

组中的每个成员都必须配置以下选项：

- **必须开启二进制日志**：设置 `--log-bin[=log_file_name]`。MySQL组复制会复制二进制日志的内容，因此必须开启二进制日志。
- **Slave Updates Logged**：设置 `--log-slave-updates`。节点需要记录`applier`已应用的日志。组中的每个节点都需要记录它们所接收到并应用的所有事务，这是必须的，因为恢复过程是依赖于组中参与者的二进制日志来进行的。因此，组中每个成员都必须保留每个事务的副本，即使某事务不是在该节点上开始的。
- **Row Format的二进制日志**：设置 `--binlog-format=row`。组复制依赖于基于行格式的二进制日志，以便在组中传播所发生的更改能保持一致性。而且，在探测组中不同节点间发生的并发事务是否冲突时，需要从行格式的日志中提取一些内容来做比较。
- **开启GTID复制**：设置 `--gtid-mode=ON`。组复制使用GTID(全局事务ID)来精确跟踪每个节点上已经提交了哪些事务。也因此可以推断出某节点上要执行的事务是否和已执行的事务(每个节点上都有副本)冲突。换句话说，GTID是整个组复制判断事务是否冲突的基础。
- **Replication Information Repositories**：设置 `--master-info-repository=TABLE` 和 `--relay-log-info-repository=TABLE`。applier需要将 master 和 relay log 的元数据信息写入到系统表 `mysql.slave_master_info` 和 `mysql.slave_relay_log_info` 中。这保证了组复制插件具有一致性恢复的能力和复制的元数据事务管理能力。

- **Transaction Write Set Extraction**：设置 `--transaction-write-set-extraction=XXHASH64` ，以便将行写入到二进制日志中时，节点也收集写集。写集基于每行的主键，是唯一标识被更改行的标签的简化形式，该标签后续会用于探测事务冲突性。

- **Multithreaded Appliers**：(某些旧版本没有该要求)可以将组复制成员配置为多线程appliers，使得可以并行应用事务。需要设置 `--slave-parallel-workers=N` (**N**是applier线程数量)、`--slave-preserve-commit-order=1` 以及 `--slave-parallel-type=LOGICAL_CLOCK`。`--slave-parallel-workers=N` 表示启用多applier线程，组复制依赖于建立在所有参与节点都以相同顺序接收和应用、提交事务的一致性机制，因此还必须设置 `--slave-preserve-commit-order=1` 以保证并行事务的最终提交是和原事务所在顺序位置一致的。最后，为了决定哪些事务可以并行执行，relay log 必须包含由 `--slave-parallel-type=LOGICAL_CLOCK` 生成的事务父信息(transaction parent information)。当尝试加入一个只设置了 `--slave-parallel-workers` 大于0，却没有设置其他两项的新成员，将会报错并阻止它的加入。

18.7.2 组复制的限制(局限性)

下面是使用组复制已知的限制：

- **Replication Event Checksums**：由于对复制事件校验的设计缺陷，目前组复制不能使用它们。因此，需要设置 `--binlog-checksum=NONE`。

- **Gap Locks**：在验证阶段(certification process)中，不会考虑 Gap Locks，因此在 InnoDB 的外部无法获取任何关于Gap 锁的信息。

注意

除非你的应用程序或业务需求依赖于 `REPEATABLE READ` ，否则我们建议在组复制中使用 `READ COMMITTED` 隔离级别。在 `READ COMMITTED` 隔离级别中，InnoDB不会使用Gap Locks，这将使得InnoDB自带的冲突探测能和组复制的冲突探测相互对齐从而保持一致。

- **Table Locks and Named Locks**：验证阶段(certification process)中不考虑表锁和命名锁(见 `get_lock()`)。

- **不支持 SERIALIZABLE 隔离级别**：在多主模型下，默认不支持该隔离级别。如果在多主模型下设置了该隔离级别，将拒绝提交事务。

- **不支持并发的 DDL 和DML 操作**：不支持在多主模型下不同节点上同时执行DDL和DML修改同一对象。在某节点上对某对象执行DDL语句时，如果在其他节点同时执行DML修改该对象，将有一定风险探测到冲突。(译注：是 DDL+DML 的并发，DDL+DDL 的并发也不允许。)

- **不支持级联的外键约束**：多主模型的组(所有节点都配置了 `group_replication_single_primary_mode=OFF`)不支持多级外键依赖，特别是表上定义了级联的外键约束(`CASCADING` foreign key constraints)。这是因为多主模型下执行外键约束的级联操作可能会出现未检测到的冲突，从而导致组内成员间数据不一致。因此，我们推荐在使用多主模型时，在每个节点上都设置 `group_replication_enforce_update_everywhere_checks=ON` 以避免出现未检测到的冲突。
  在单主模型下没有这种问题，因为没有并发写操作，从而不可能会出现未被探测到的冲突。

- **大事务可能会错误**：如果一个事务非常大，导致GTID的内容非常多，以至于无法在 5 秒内通过网络传输完成，这时组成员间的通信将失败。要避免该问题，可以尽可能地限制事务的大小。例如，将 `LOAD DATA INFILE` 的文件切割为多个小块。

- **多主模型可能出现死锁**：在多主模型下， `SELECT ... FOR UPDATE` 语句可能会导致死锁。这是因为组内成员之间不会共享锁资源(译注：share nothing)，因此这样的语句可能达不到预期的结果。

18.8 组复制常见问题FAQ

本节列出了关于组复制的常见问题。

复制组允许的最大成员数量？

一个复制组最多只允许9个成员，当想要添加第10个成员时，将会拒绝加入。

组内成员如何连接？

组内的成员会建立端对端(peer-to-peer)的 TCP 连接。这些连接仅用于组内通信和消息传递。

选项group_replication_bootstrap_group是干什么用的？

bootstrap标记指示这个成员去创建一个组，并且扮演组内第一个种子节点。第二个成员要加入组时，需要询问这个引导者，让其动态更新组配置以便让新成员加入组。

有两种场景下需要一个成员来引导组。当组第一次被创建时，或者当停止整个组后再启动组时。

我要如何设置恢复过程(阶段)的凭据？

使用 CHANGE MASTER TO 语句先配置好组复制恢复通道凭据即可。

使用组复制是否可以负载写操作？(scale-out write-load)

无法直接实现这样的目的，但是MySQL组复制是一种 **share-nothing** 的复制模式，组内所有节点都有完全相同的数据副本。因此，如果组内某节点上通过事务提交操作向存储引擎写入了 N 字节的数据，那么其他所有节点上也都会向存储引擎写入 N 字节数据，因为在组中，事务会到处复制。

虽然其他节点上也写入 N 字节数据，但是它们不像发起事务的那个节点一样会执行事务，它们完成这个事务的速度非常快。事务以一种格式(组复制强制限制为 **row-based** 格式)在组内复制，因此其他节点只需转换行数据即可，无需去重新执行事务。

此外，"以 **row-based** 的方式传播已发生的更改"意味着它们以一种优化的、紧凑的方式接收事务，相比于事务发起节点，这会减少IO的操作数量。

总而言之，通过在组内传播无冲突的事务可以实现写操作的负载。而且，还可以负载一小部分的IO操作，因为远程节点接收事务后会经过"读--改--写"的过程，从而只写入一些必要的更改到存储中。

在相同的工作负载下，组复制是否比常规复制需要更多的网络带宽和CPU？

确实如此。因为处于组内同步的原因，每个节点都需要不断地和其他成员交互，所以在预期上它需要更多资源。难以量化到底多需要多少资源的数据。它还取决于组的大小(组中只有3个节点肯定比9个节点需要的带宽压力小)。

此外，内存和CPU的需求也更大，因为组内节点同步和消息传递需要完成更复杂的工作。

能否在广域网(WAN)上部署组复制？

可以。但是组内成员之间的连接必须要保证可靠性以及良好的网络传输性能。低延迟、高带宽的网络连接是最优性能的基本要求。

如果网络带宽是个问题，可以使用消息压缩功能(Message Compression)来降低带宽的要求。但是，如果出现网络丢包问题，从而导致数据重传和高延迟，那么组复制的吞吐量和延迟性都将受到影响。

警告

当组内任何成员之间的网络往返时间(RTT)为2秒或以上时，您可能会遇到一些问题，因为内置的故障检测机制可能被不正确触发。

当出现临时的连接问题，成员是否可以自动重新加入组？

这取决于出现连接问题的原因。如果连接问题是短暂的且再次连接成功足够快速，那么故障探测器就无法意识到这个问题，这个成员就不会从组中踢出。如果"短"期内无法再次连接成功，那么故障探测器就会怀疑这个节点是否故障，于是节点会从组中踢出。

当节点从组中踢出去后，你需要重新将它加入到组中。换句话说，当节点离开组后，你只能手动(或使用脚本)将它重新加入回组。

成员何时会被排除在组外？

如果节点沉默了，其他成员将其从组配置中移除。这可能是发生了节点宕机或网络断开的情况。在一段超时时间之后，这个故障会被探测到，之后会重新配置组，使其不包含这个故障节点。

当有一个节点明显落后时，会发生什么？

没有任何可以制定的策略用来决定何时自动从组中驱逐一个节点。你需要查找出这个成员为什么会明显落后，然后修复它或者将它从组中移除。否则，如果节点很慢，慢到触发流程控制(`flow control`)，这将会让整个组都放慢脚步。可以根据你自己的需要来定制 `flow control`。

当怀疑组中某节点故障时，是否有专门的成员负责触发组重新配置的操作？

没有，组中没有专门的成员复制触发组重新配置。

任何成员都可能怀疑出现了问题。所有成员都需要(这是自动的)就某成员出现故障的问题达成一致。然后会有一个(某)成员通过触发组重新配置，负责将故障节点驱逐出组，但这个成员无法控制和设置的。

我可以用组复制来进行分片(sharding)吗？

组复制旨在提供高可用的副本集：组中每个节点上的数据都是重复的。为了扩展单系统所能提供的能力，你需要一个围绕着一系列组复制集的编排(orchestration)和分片框架(译注：编排的意思是在复杂的环境中让杂乱、复杂的工作简化、流程化)，使得每个副本集都负责维护和管理整个数据集中的一个给定切片或分区。这种类型的配置通常称为"分片集群(sharded cluster)"，可以让你无限地线性扩展读、写。(译注：此处的扩展请理解为"负载分散"、"压力分散"的意思，也就是让架构具有负载伸缩性)

使用组复制时如何配置iptables？

如果启用了iptables，你需要放行用于成员间通信的端口。使用 `iptables -L` 查看当前规则。假定你配置的用于成员间通信的端口为6606，执行 `iptables -A INPUT -p tcp --dport 6606 -j ACCEPT` 可放行该端口上的通信数据。(译注：节点上用于 成员间通信的端口是给其他节点联系本节点用的，本节点联系其他节点时自身分配一个随机端口)

如何恢复成员所使用的复制通道的relay log？

组复制所使用的复制通道和常规主从复制使用的通道行为是一样的，因此依赖于relay log。当修改了 `relay_log`变量，或者未设置该变量而主机名发生了改变，则可能会出现错误。这类问题的解决方法见Section 17.2.4.1, “The Slave Relay Log” 。你也可以执行`STOP GROUP_REPLICATION`和`START GROUP_REPLICATION`语句来重启组复制，此时组复制插件会重新创建group_replication_applier通道，这也能修复该问题。

为什么组复制要使用两个地址？

使用两个地址是为了将SQL客户端的请求和组内成员通信的网络流量分开，其中组内其他成员连接本节点时使用的地址由选项 `group_replication_local_address` 设置。例如，某节点有两个网络接口，地址分别为 203.0.113.1 和 198.51.100.179，group_replication_local_address=203.0.113.1:33061表示`203.0.113.1:33061`用于组内通信，然后你可以使用`198.51.100.179`作为主机名。以后SQL客户端程序可以通过连接`198.51.100.179:3306`来获取MySQL提供的数据库服务。

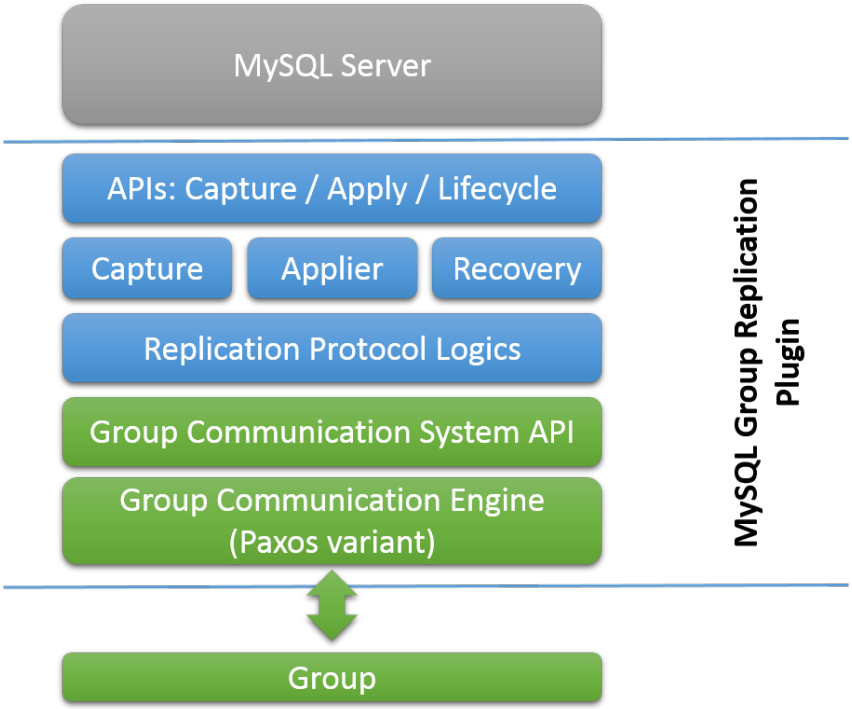
18.9 组复制技术的细节

本节提供MySQL组复制的更多技术细节。

18.9.1 组复制插件的架构

MySQL组复制是一个MySQL插件，它基于现有的MySQL复制基础，利用了基于行格式的二进制日志和GTID等特性。下图是MySQL组复制的整体框架图。

Figure 18.9 Group Replication Plugin Block Diagram



从上图的最顶端开始，有一系列的API控制组复制插件如何和MySQL Server进行交互(图中灰色方框)。中间有一些接口可以使得信息从MySQL Server流向组复制插件，反之亦然。这些接口将MySQL Server核心部分和插件隔离开来，并且大多数钩子(hook)都位于事务执行管道中(pipeline)。在Server到插件的方向上，传递一些通知信息，例如**server**正在启动，server正在恢复，server已准备好接收连接，server将要提交事务等等。另一方向，即插件到server的方向上，插件会通知 server对事务进行提交，终止正在进行的事务，将事务放进relay-log中排队等等。

从API往下，是一些响应组件，当通知信息路由到它们身上时就响应。capture组件负责跟踪正在执行的事务的上下文信息。applier组件负责在本节点上执行远程传输来的事务。recovery组件负责管理分布式恢复过程，还负责在节点加入组时选择donor，编排追赶过程以及对选择donor失败时的反应(译注：新节点加入组之前先要选择donor，并从donor上获取缺失的数据以便赶上组中已有的数据。即先和组中数据同步，然后才能加入到组)。

继续向下，replication协议模块包含了特定的复制协议逻辑。它负责探测冲突，在组中接收和传播事务。

最后两层是组内通信系统(GCS)的API(第一个绿色方框)，以及一个基于Paxos组通信引擎的实现(implementation)(第二个绿色方框)。GCS API是组内通信API，它是一个上层API，它抽象了构建一个复制状态机(状态机在本章最前面的背景知识中解释过)所需的属性，它从插件的更上层解耦了消息传递层。组通信引擎负责处理组内成员的通信。

18.9.2 组复制中的(复制)组

在MySQL组复制中，一系列节点组成一个复制组。复制组有名称，名称形式为UUID。复制组是动态的，任何时候，组内成员都可以离开(既可以是自愿离开，也可以非自愿离开)，组外成员也可以加入。当出现离组、加组的情况，组会自动调整它自身，以接受新的组配置。

如果一个节点加入组，它会自动从组内某个成员身上抓取它所缺失的那部分数据以便和组数据保持同步。这个状态是通过MySQL异步复制来实现的。如果节点离开组，例如出于维护的目的而停止mysql实例，剩余的节点会意识到它的离开，然后会自动重新配置组。Section 18.1.3.2, “组成员服务” 对此做了更详细的说明。

18.9.3 DML语句

由于对于任何特定的数据集都没有主节点(master)的概念，所以组中的每个节点都允许在任意时间执行事务，甚至可以执行更改状态的事务(RW事务)。

任何节点都可以以一种无需由因及果的方式执行事务(译注：事务发起节点执行事务是一句一句执行按逻辑执行下去，非事务发起节点可以跳过这些逻辑，直接修改行数据为结果数据)。但是，在提交的时候，必须和组中其他节点进行协调以便对该事务命运的决定达成一致。协调提供两种目的：(1)检查事务是否应该提交；(2)在组内传播更改，以便让其他节点也应用该事务。

由于事务是通过一个原子性的广播发送的，所以组内所有节点要么全部接收到该事务，要么全都未接收到该事务。如果它们都接收到了，它们接收到的事务所在顺序位置是完全一致的。冲突检测是通过检查和比较事务的写集来进行的，所以它们是在行级别上进行探测。探测到了冲突的解决方案是先提交者获胜规则。如果在不同节点上并发执行 t1 和 t2 事务，它们都修改同一行，如果t2先提交，则t2获胜，t1终止。换句话说，此时的t1是在尝试修改t2的过期数据。

注意

如果两个事务经常冲突，更好的方法是在同一个节点上开始这两个事务。这样，它们有机会在本地锁管理器上同步，而不是在稍后的复制协议中终止。

18.9.4 DDL语句

在执行DDL语句时需要小心。考虑到MySQL不支持原子或事务的DDL，因此不能乐观地执行DDL语句以及回滚需求。因此，由于DDL缺乏原子性的能力，它并不能直接适应与组复制基于的乐观复制范式。

所以，在复制DDL语句时需要多加小心。schema的更改以及对对象中包含的数据更改需要在同一个节点上进行处理，防止schema的操作还没有完成，却复制到了其他节点上。如果不这么做，可能会导致数据不一致性。

注意

如果部署的是单主模型的组复制，则没有此问题，因为所有的更改都只在主节点上执行。

警告

MySQL的DDL语句不具备原子性，也没有DDL类型的事务。组不会在执行和提交DDL操作之前做一致性协商。因此，你必须将对同一对象操作的DDL语句和DML语句路由到同一个节点上。

18.9.5 分布式恢复

本节描述正在加组的成员赶上组内已有节点数据的过程，这个过程称为分布式恢复阶段。

18.9.5.1 分布式恢复基础

组复制的分布式恢复过程可以总结为：新节点从组中在线节点获取它所缺失的那部分数据，同时监听组中正在发生的事件。在恢复过程中，新节点会监听组中发生的事件，也会监听正在发生的事务。这是一句高层次的总结(译注：浓缩就是精华，这个总结浓缩的太精华了)。下面几小节将提供关于这两个阶段详细的说明。

Phase 1

在第一阶段，新节点会从组中选择一个在线节点作为donor来获取它缺失的数据。donor负责传输新节点开始加组那一刻之前的所有数据需求(译注：这个加组指的是新节点请求加入组的那一刻，而不是成功加入组的那一刻)。这是通过在donor和新节点之间建立一个标准的异步复制通道来实现的，二进制日志日志会通过这个通道一直向上流动，直到成员视图更新时，这意味着新节点成功成为组中成员。当新节点从donor处接收到二进制日志时，它会去应用它们。

此外，在二进制日志传输进行时，新节点也会缓存组内发生的每一个事务。即，新节点正从donor处获取缺失数据的同时，也会监听从它建立异步复制通道后组中发生的新事务。当第一阶段完成后，和donor建立的复制通道就会关闭，之后进入第二阶段：追赶。

Phase 2

在这个阶段，新节点继续执行缓存中的事务，当队列中待执行的事务数量最终达到0时，该成员将标记为ONLINE。

高弹性(可容错)

在新节点从donor获取二进制日志的时候，恢复过程可以允许donor出现故障。这种情况下，当donor在第一阶段故障时，新节点将自动重新选择一个新的donor，并从新donor那里恢复到旧donor故障的那一刻。这意味着，新节点会自动地关闭和旧donor的通道，并和新donor建立通道。

18.9.5.2 从一个特定时间点处恢复

为了让新节点和donor的某个特定时间点保持同步，新节点和donor使用GTID机制。但这还不够，因为这仅提供了一种让新节点意识到它所缺失的是哪些事务的方法。它没有为新节点必须追赶到哪个特定时间点做任何标记，同时也没有传输认证(certification)有关的信息。这正是二进制日志视图标记登台的地方。它们标记二进制日志流中的视图更改，还包含一些额外的元数据信息，以及给新节点提供关于它缺失的认证相关数据。

视图和视图更改

理解视图和视图更改是什么对解释视图更改标记很重要。

*视图*对应的是当前配置组内在线成员的列表。换句话说，在任意特定的时间点，在线且一切正常的节点。

*视图更改*发生在对组配置进行修改时，例如有节点要加组或离组。任何一次组成员变更都会导致一次独立的视图更改，在同一逻辑时间点传输给组内所有成员。

*视图标识符*惟一地标识一个视图。在视图更改时会更改它。

在组通信层，视图更改以及它们关联的视图id是加入组之前和之后的区分边界。这个概念是通过一个新的二进制事件实现的："视图更改事件"。因此视图id也作为组成员配置变更之前、之后传输的事务的标记。

视图标识符由两部分组成：第一部分是创建组时随机生成的，在组停止(关闭)之前一直保持不变；第二部分是一个单调递增的整数，每次视图发生更改都递增一次。

使用这种混合视图id的原因是，可以明确地标记当成员加入或离开时发生的组成员配置更改，也能标记所有成员离开组后没有任何信息保留在视图中。实际上，单纯使用单调递增的整数作为标识符会导致在组重启后重用视图ID，显然这会破坏恢复过程所依赖的二进制日志标记的唯一性。总而言之，第一部分标识这个组是从什么时候启动的，第二部分标识组在什么时间点发生了更改。

18.9.5.3 视图更改

本节解释如何控制视图更改标识符合并到二进制日志事件并写入到日志的过程。有以下几个步骤：

Begin: 稳定的组

所有节点都是在线的，并且处理组内事务。某些节点可能会因为复制的原因而稍由落后，但最终它们会达到一致。组在这里充当一个分布式的、复制的数据库。

Figure 18.10 Stable Group

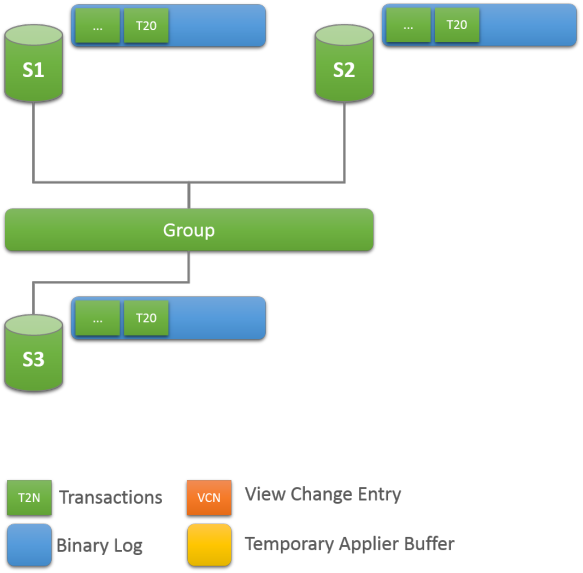
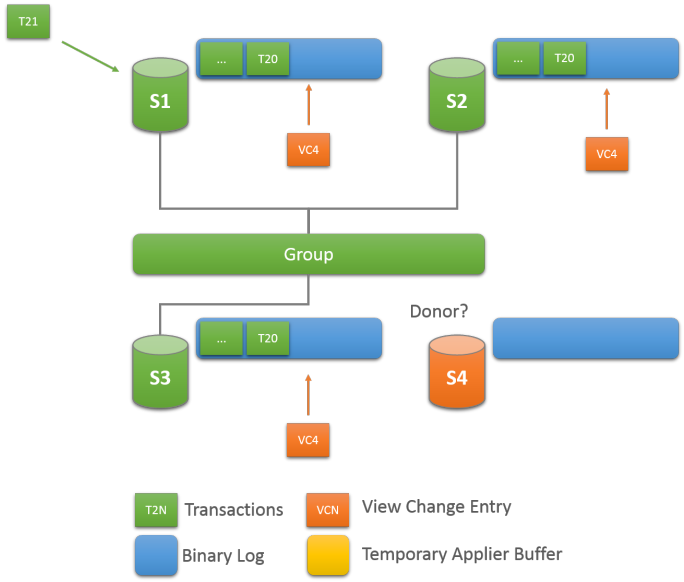


Figure 18.11 A Member Joins



View Change: 新成员要加入

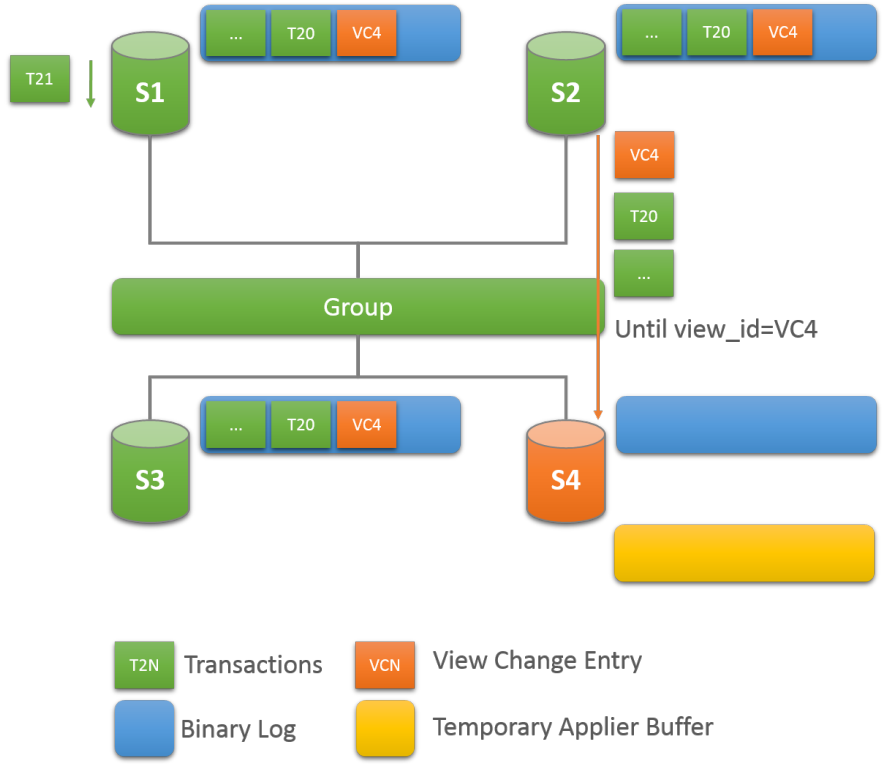
每当一个新成员要加入组时，都会更改视图，每个在线节点都会将视图更改的日志事件排队以等待执行。之所以要排队，是因为在更改视图之前可能已经有一些事务在排队等待应用，这些事务属于旧的视图。将视图更改事件进行排队，保证了这种情况下能正确对其标记。

同时，待加入成员会从组内在线成员列表中选择一个donor。上面的右图中，这个新成员对应第4个视图(译注：组创建时视图id的第二部分为1，第二个成员加入后第二部分为2，依此类推，这是第四个成员，所以第二部分为4)，目前在线的(3个)成员会向二进制日志中写入视图更改事件。

State Transfer: 追赶

当待加组成员选择好donor之后，它们之间会建立新的异步复制连接，并且开始状态传输(第一阶段)。该成员和donor的交互会一直持续到该成员的applier线程开始处理视图更改日志事件，因为这个事件对应的是该成员请求加入组的那一刻。换句话说，待加组成员会一直从donor上复制二进制日志，直到发现了此次加组时的视图更改标识符。

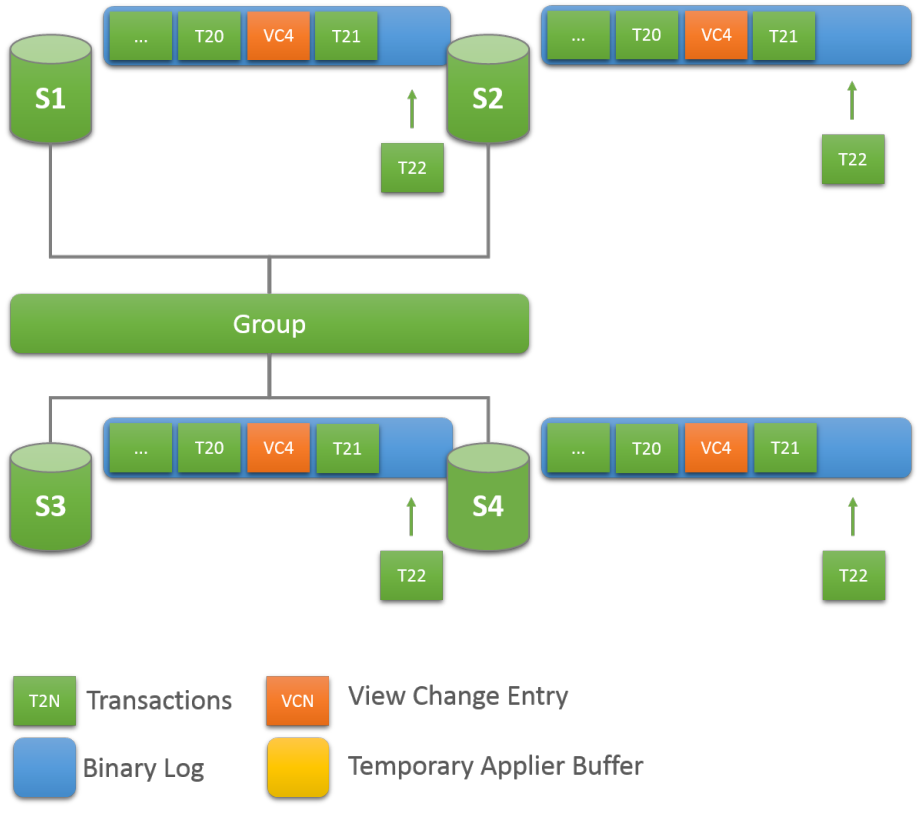
Figure 18.12 State Transfer: Catching Up



由于视图标识符会在同一逻辑时间点传输到组中所有成员，待加组成员知道在哪个视图标识符处停止复制二进制日志。这避免了复杂的GTID集计算，因为视图ID明确标记了哪部分数据属于哪个视图。

当待加组成员从donor上复制时，它也会缓存新进入组中的事务。最终，它停止从donor处复制，并转而去应用那些已缓存的事务。

Figure 18.13 Queued Transactions



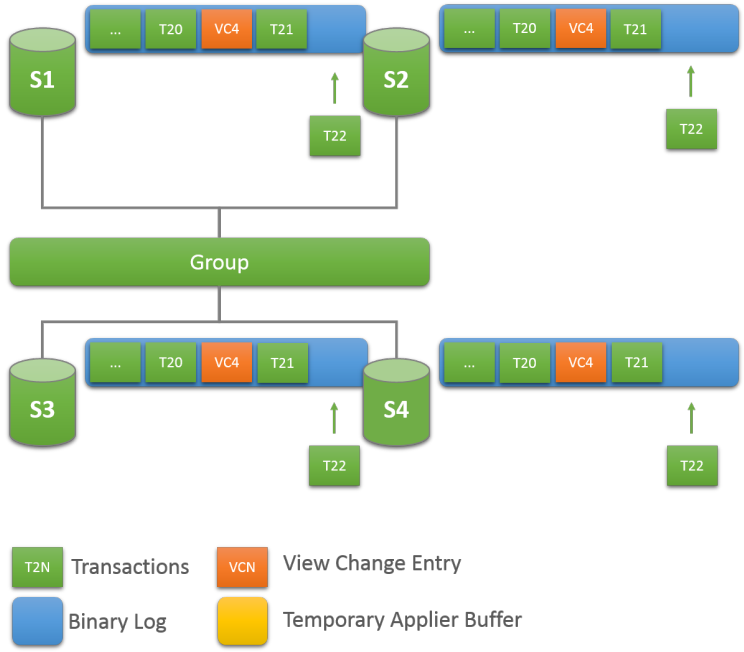
Finish: 追赶完成

当待加组成员根据视图标识符识别了一个视图更改的日志事件后，它和donor之间的连接就会断开，然后它开始应用已缓存的事务。需要理解的一个关键之处在于这是恢复过程的最终步骤。尽管视图更改事件在二进制日志中充当了一个标记，分隔了视图更改，但它还扮演另一个角色。它传达了在它加组时其他所有节点感知到的认证信息(certification information)，也就是最后一个节点(即这个新节点)的视图也更改了。如果没有它，待加组成员就不具备验证(探测冲突)后续事务的必须信息。

追赶的持续时间(第二阶段)是不确定的，因为它依赖于负荷程度以及组内事务传输的速率。追赶的过程是完全在线的，待加组成员不会阻塞组内其他任何节点。因此，当恢复过程前进到第二阶段后，待加组成员落后的事务数量因这个原因而变化，它根据负荷程度而增加或减少。

当待加组成员的排队事务数量达到 0 后，它所存储的数据和组内其他节点就保持了一致，这时它在组内的状态会改变为ONLINE。

Figure 18.14 Instance Online



18.9.5.4 分布式恢复的使用建议和局限性

分布式恢复有一些局限性。它基于传统的异步复制，因此如果它在复制之前完全没有数据或者备份的数据集很老，那么恢复的过程会比较慢。这意味着如果在第一阶段要传输的数据非常大，会需要一长段时间来恢复。因此，在新成员加入组之前，建议从组中在线节点备份最新的数据并恢复到该新成员上，这会将第一阶段的持续时间缩短，且会降低对donor的影响，因为它需要保存和传输更少的二进制日志。

18.9.6 可观察性

在组复制中内置了很多自动化功能。尽管如此，某些时候你可能想要了解幕后所发生的事情。这就是组复制的仪表(译注：即状态显示)和Performance Schema重要的原因。通过performance_schema表可以查询到系统的整个状态(包括视图、冲突统计和各种服务的状态)。

复制协议的分布式特性，以及所有节点对决定达成一致，事务和元数据因此而保持同步这一事实，使得组的状态检查变得更简单。例如，你可以连接到组中的某节点上，通过 select语句查询performance_schema中关于组复制的表，可以获取本地和全局信息。更多信息见"18.3 监控组复制"。

18.9.7 组复制的性能

本节解释如果使用可配置选项以便让你的组复制达到最佳性能。

18.9.7.1 微调组通信线程

当组复制插件加载后，组通信线程(GCT)工作在一个循环内。GCT从组中和组复制插件中接收消息，负责处理仲裁、故障探测相关任务，以及向外发送存活消息、处理MySQL Server和组复制插件之间流入/流出的事务。GCT排队等待流入的消息，当没有消息时，GCT会等待。将这个等待时长配置久一点(会做一个主动等待, active wait)，可以让GCT晚一点进入睡眠态，有些情况下这样配置是有益的。这是因为切换到sleep态时，操作系统会将CPU从GCT切换出来，这会导致上下文切换。

为了强制GCT做active wait，配置 `group_replication_poll_spin_loops` 选项，这使得在实际轮询队列中的下一条消息之前，GCT循环不会做任何和已配置的循环数量相关的工作。(译注：这个变量的作用是：GCT在等待下一条消息之前，等待消息引擎互斥锁释放的时长，设置长一点时间，可以多等一段时间，收到下一条消息的中间不用临时进入睡眠态)

例如：

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

18.9.7.2 压缩消息

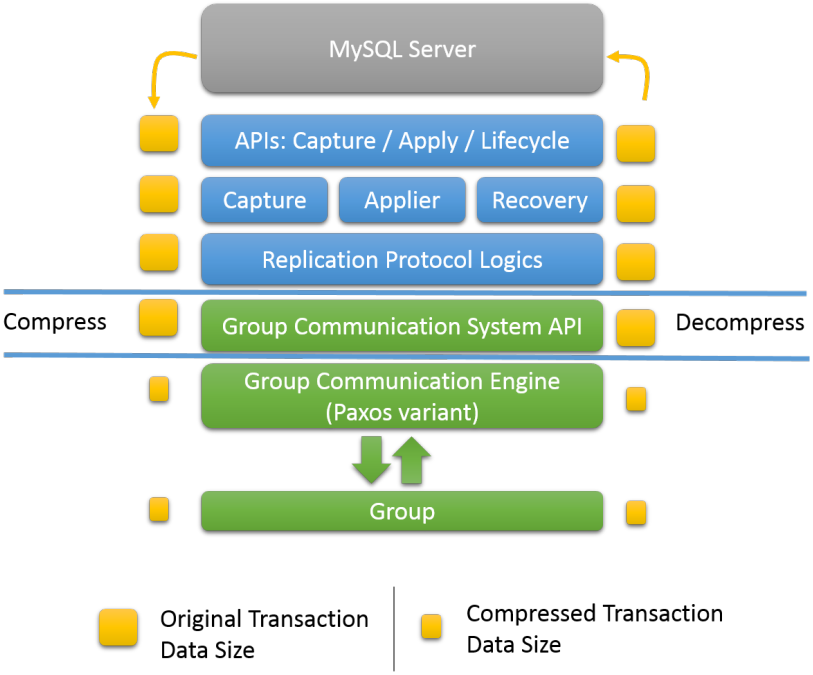
当网络带宽是一个瓶颈时，压缩消息可以提升组通信层的30%-40%的吞吐量。对于负载较重的大型复制组，这非常有用。

Table 18.8 LZ4 对不同二进制日志格式的压缩率

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

由于组内N个节点为了内部通信需要和组内其他所有节点都建立TCP端对端的连接，这使得同一份数据要向外发送N次(译注：实际是N-1次，需要减去本节点自身)。更重要的是，二进制日志的压缩比较高(见上表)。这使得在包含大事务高负载的环境下，消息压缩成为一个引人瞩目的功能。

Figure 18.15 Compression Support



消息压缩的位置在将数据交给组通信线程之前的组通信引擎层，因此它发生在mysql用户会话线程的上下文中。事务在被发送到组之前被压缩，在被接收时被解压缩。 压缩是有条件的，取决于配置的阈值。 默认情况下压缩已启用。

此外，不要求组内所有节点都开启压缩功能才能一起工作。收到消息后，该成员会检查消息是否已被压缩，如果被压缩，则在消息传输到上层之前先解压缩。

使用的压缩算法是LZ4。默认启用压缩的阈值为1000000字节(译注：即1MB左右)。这个阈值可以设置为比默认值更大，这样只有超出阈值的大事务才会进行压缩。下面是设置压缩阈值的示例：

```
STOP GROUP_REPLICATION;
SET GLOBAL group_replication_compression_threshold= 2097152;
START GROUP_REPLICATION;
```

上面设置了压缩阈值为2MB。如果事务生成的复制消息大于2MB，例如这个事务对应生成的二进制日志大于2MB，将会压缩这段日志。如果要禁用压缩功能，将其设置为0。

18.9.7.3 流程控制(flow control)

组复制只有在组内所有节点都收到了事务，且大多数成员对该事务所在顺序及其他一些相关内容都达成一致时才会进行事务提交。

如果写入组的总数量都在组中任何成员的写入能力范围之内，则此方法运行良好。如果某节点或某些节点的写吞吐量较其他节点更差，则这些节点可能会落后。

当组中有一些成员落后了，可能会带来一些问题，典型的是读取这些落后节点上的数据都是比较旧的。根据节点为什么落后的原因，组内其他成员可能会保存更多或更少的上下文，以便满足落后成员潜在的数据传输请求。

好在复制协议中有一种机制，可以避免在应用事务时，快速成员和慢速成员之间拉开的距离太大。这就是所谓的***流程控制(flow control)***机制。该机制试图实现以下几个目标：

1. 保证成员足够接近，使得成员之间的缓冲和非同步只是一个小问题；
2. 快速适应不断变化的情况，例如不同的工作量、更多的写节点；
3. 给每个成员分配公平的可用写容量；(译注：其实就是取某种类型的平均写能力值)
4. 不要因为避免浪费资源而过多地减少吞吐量。

考虑到组复制的设计，可以考虑两个工作队列来决定是否节流：*(1)*认证队列(certification)；*(2)*二进制日志上的应用队列(applier)。当这两个队列中的任何一个超出队列最大值(阈值)时，将会触发节流机制。只需配置：*(1)*是否要对ceritier或applier或两者做flow control；*(2)*每个队列的阈值是多少。

flow control依赖于两个基本的机制：

1. 监控组内每个成员收集到的有关于吞吐量和队列大小的一些统计数据，以便对每个成员可以承受的最大写入压力做有根据的猜测；
2. 每当有成员的写入量试图超出可用容量公平份额，就对其节流限制。

探针和统计数据

flow control依赖的监控机制是通过在组中每个成员上部署一系列的探针来实现的，这些探针收集它们的工作队列和吞吐量信息，然后将这些信息每隔一段时间就传播给组内其他成员进行共享。

这些探针分散在整个插件栈中，并且允许对探测指标进行设置，例如：

- certifier队列的大小；
- applier队列的大小；
- 已被认证(certified)的事务总数量；
- 本节点上已应用的远程事务总数量；
- 本地事务总数量。

当某成员接收到其他成员发送的统计信息时，它会计算额外的指标：自上次监控期以来有多少个事务被验证、被应用以及在本地执行了。

监控数据会每隔一段时间就和其他成员共享。监控时间间隔必须足够长，以便允许其他节点能够决定当前的写请求(译注：即对写请求达成一致)，但也要足够短，使其对组的带宽影响最小。这些信息每秒钟都会共享一次，这段时间足以解决这两个问题。

组复制节流

基于收集到的组中所有成员的指标，会启动一个节流机制，用来决定是否限制某成员执行/提交新事务的速率。

因此，从其他所有成员处获取的指标是计算每个成员能力的基础：如果某成员的队列中事务很多(certification或applier队列)，则应当让该成员执行新事务的能力接近上一周期认证和应用的能力。

组中能力最低的成员决定了组真正的处理能力，而本地事务数量决定了正有多少成员在写入它。因此，这些正在写入的成员之间应该共享这个最低处理的能力。

这意味着每个成员都有一个建立在可用容量基础上的写配额，换句话说，它可以在下一周期安全地写这些事务。如果certifier或applier队列大小超出了阈值，节流机制将强制设置写配额。

上一周期中延迟的事务数量会降低本期配额的大小，在此基础上再减少10%，以允许触发问题的度列可以减小它的大小。为了避免队列大小超出阈值时吞吐量大幅增加，此后的吞吐量只允许每周期增长10%。

当前节流机制不会惩罚低于配额的事务，但会延迟完成那些超过监控期限的事务。 因此，如果发出的写请求的配额非常小，则某些事务的延迟可能接近整个监控周期。

(译注：关于监控和节流限制，大概如下：
1.监控有周期，每个周期都会向外共享统计信息。收到统计信息后，会计算一些指标。
2.基于这些指标，会有一个节流机制决定执行事务的速率和能力。
3.为了避免拖后腿的问题，节点之间会共享最慢的写能力作为这个周期的写配额。
4.如果配额很小，执行事务的能力很低，就会积压事务，这些积压的事务会跨过监控周期进入下一个监控周期。
5.积压到下一个周期的事务越多，说明在下一个周期应该允许接收的新事务数量越少，否则会越压越多。
)