DRBD9用户指南

目录

请先看这个

本指南旨在为分布式复制块设备Distributed Repliated Block Device 版本9(DRBD-9)的用户提供最终参考指南和手册。

该项目的发起公司 linbit，正在向DRBD社区免费提供，希望它能造福业界。指南不断更新。我们尝试在相应的DRBD版本中同时添加有关新DRBD特性的信息。本指南的在线HTML版本始终在 https://links.linbit.com/DRBD9-Users-guide上提供。

本指南始终假设您正在使用最新版本的DRBD和相关工具。如果您使用的是8.4版本的DRBD，请使用 https://links.linbit.com/DRBD84-Users-guide提供的本指南的匹配版本。

请使用the drbd-user mailing list提交评论。

本指南的组织如下：

- DRBD简介介绍DRBD的基本功能。本文简要概述了DRBD在Linux I/O堆栈中的位置，以及基本的DRBD概念。它还详细研究了DRBD最重要的特性。
- 建立和安装DRBD软件讨论从源代码构建DRBD，安装预构建的DRBD包，并包含在集群系统上运行DRBD的概述。
- 使用DRBD讲述如何使用资源配置文件管理DRBD，以及常见的故障排除方案。
- 支持DRBD的应用程序利用DRBD为应用程序添加存储复制和高可用性。它不仅涵盖了Pacemaker cluster manager中的DRBD集成，还包括高级LVM配置、DRBD与GFS的集成，以及为Xen虚拟化环境添加高可用性。
- 优化DRBD性能包含从DRBD配置中获得最佳性能的要点。
- 了解更多深入了解DRBD的内部，还包含指向其他资源的指引，本指南的读者可能会发现这些指引很有用。
- [P-Appendices]:
  - 最近的变化是DRBD 9.0与早期DRBD版本相比所做更改的概述。

欢迎对DRBD培训或支持服务感兴趣的用户通过 sales@linbit.com 或 sales_us@linbit.com 与我们联系。

DRBD简介

1. DRBD基础

DRBD是一个基于软件的、无共享、复制存储解决方案，它在主机之间镜像块设备（硬盘、分区、逻辑卷等）的内容。

DRBD镜像数据

- **实时**: 当应用程序修改设备上的数据时，数据的副本更改是连续进行的。
- **透明**: 应用程序不会意识到数据实际上是存储在多台主机上的。
- **同步** or **异步**: 当使用同步镜像数据时，只有在所有(连接上的)主机上都完成写操作后，才会通知应用程序写完成。当使用异步镜像数据时，在本地完成写入时（通常在镜像数据传输到其他节点前），就会通知应用程序写入完成。

1.1. 内核模块

DRBD的核心功能是通过Linux内核模块实现的。具体来说，DRBD构成虚拟块设备的驱动程序，因此DRBD位于系统的I/O堆栈的底部附近。因此，DRBD非常灵活和通用，这使得它成为一个复制解决方案，适合为几乎任何应用程序添加高可用性。

根据定义和Linux内核体系结构的要求，DRBD与上面各层无关。因此，DRBD不可能奇迹般地向上层添加它们不具备的特性。例如，DRBD无法自动检测文件系统损坏，也无法向ext3或XFS等文件系统添加活动群集功能。

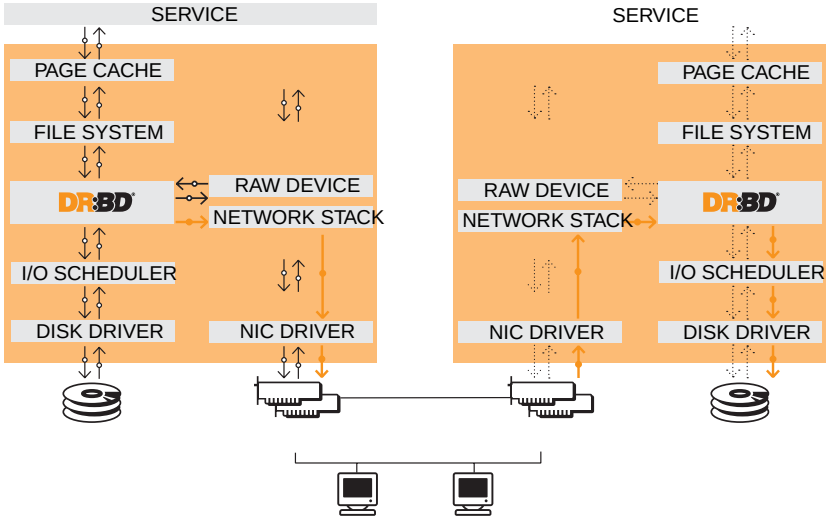


插图 1. DRBD在Linux I/O堆栈中的位置

1.2. 用户空间管理工具

DRBD附带了一组管理工具，这些工具与内核模块通信，以便配置和管理DRBD资源。从上到下列举如下：

drbdadm

DRBD-utils程序套件的高级管理工具。从配置文件`/etc/DRBD.conf`获取所有DRBD配置参数，并充当 drbdsetup 和 drbdmeta 的前端。drbdadm 有一个用 -d 选项调用的 *dry-run* 模式，它显示哪些 drbdsetup 和 drbdmeta 调用 drbdadm ，而不实际调用这些命令。

drbdsetup

配置加载到内核中的DRBD模块。drbdsetup 的所有参数都必须在命令行上传递。drbdadm 和 drbdsetup 之间的分离带来了最大程度上的灵活性。在绝大多数情况下，大多数用户几乎不需要直接使用 drbdsetup。

drbdmeta

允许创建、转储、还原和修改DRBD元数据结构。与 drbdsetup 一样，大多数用户很少直接使用 drbdmeta。

1.3. 资源

在DRBD中，*资源* 是指特定复制数据集的所有方面的集合术语。其中包括：

资源名称

This can be any arbitrary, US-ASCII name not containing white space by which the resource is referred to.

卷

任何资源都是一个复制组，由共享同一复制流的多个 *卷* 之一组成。DRBD确保了资源中所有卷的写保真度。卷的编号以 0 开头，一个资源中最多可以有65535个卷。卷包含复制的数据集和一组供DRBD内部使用的元数据。

在 drbdadm 级别，可以通过资源名称和卷号将资源中的卷寻址为 *resource*/*volume*。

DRBD设备

This is a virtual block device managed by DRBD. It has a device major number of 147, and its minor numbers are numbered from 0 onwards, as is customary. Each DRBD device corresponds to a volume in a resource. The associated block device is usually named /dev/drbd*X*, where *X* is the device minor number. udev will typically also create symlinks containing the resource name and volume number, as in /dev/drbd/by-res/*resource*/*vol-nr*. Please note that depending on how you installed drbd, you might need to install the drbd-udev package on RPM based systems.

早期的DRBD版本劫持了NBD的设备主要编号43。这已经过时很久了；147是 LANANA-registered 中定义的DRBD 主设备号。

连接

连接 是共享复制数据集的两台主机之间的通信链路。在DRBD 9中，每个资源可以在多个主机上定义；在当前版本中，这需要在这些主机之间建立一个完整的网状连接（即，每个主机都为该资源彼此连接）

在 drbdadm 级别，连接由资源和连接名（后者默认为对等主机名）寻址，如 *resource*:*connection*。

1.4. 资源角色

在DRBD中，每个resource都有一个角色，该角色可以是 *Primary* 或 *Secondary* 。

这里的术语选择不是任意的。这些角色被DRBD的创造者特意命名为"Active"和"Passive"。*Primary* 与 *secondary* 是指与 **存储** 的可用性相关的概念，而 *active* 与 *passive* 是指 **应用程序** 的可用性。在高可用性环境中，通常情况下 *primary* 节点也是 *active* 节点，但这并不是必需的。

- 处于 *primary* 角色的DRBD设备可以不受限制地用于读写操作。它可用于创建和装载文件系统、原始或直接I/O到块设备等。
- 处于 *secondary* 角色的DRBD设备接收来自对等节点设备的所有更新，但在其他情况下完全不允许访问。它不能被应用程序使用，也不能用于读写访问。甚至不允许对设备进行只读访问的原因是必须保持缓存一致性，如果以任何方式访问辅助资源，这是不可能的。

当然，资源的角色可以通过manual intervention，调用集群管理应用程序的一些自动算法，或者automatically来更改。将资源角色从次要角色更改为主要角色称为 *promotion* (升级)，而反向操作称为 *demotion* (降级)。

2. DRBD特征

本章讨论了各种有用的DRBD特性，并给出了它们的一些背景信息。这些特性中的一些对大多数用户都很重要，有些只在非常特定的部署场景中才相关。使用DRBD和故障排除和错误恢复包含如何在日常操作中启用和使用这些功能的说明。

2.1. 单一主(Single-primary)模式

在单一主模式下，资源在任何给定的时间仅在一个群集成员上处于主角色。由于可以保证在任何时候只有一个集群节点操作数据，因此这种模式可以用于任何传统的文件系统（ext3、ext4、XFS等）。

在单一主模式下部署DRBD是高可用性（支持故障转移）集群的规范方法。

2.2. 双主(Dual-primary)模式

在双主模式下，资源在任何给定时间都处于两个群集节点上的主角色[1]。由于可以并发访问数据，因此此模式需要依赖使用分布式锁管理器的共享群集文件系统。示例包括GFS和OCFS2。

在双主模式下部署DRBD是负载均衡集群的首选方法，这种集群需要从两个节点并发访问数据，例如需要实时迁移的虚拟化环境。此模式在默认情况下是禁用的，并且必须在DRBD的配置文件中显式启用。

有关为特定资源启用双主模式的信息，请参见启用双主模式。

2.3. 复制(Replication)模式

DRBD支持三种不同的复制模式，允许三种程度的复制同步性。

Protocol A

异步复制协议。一旦本地磁盘写入完成，并且复制数据包已放置在本地TCP发送缓冲区中，则认为主节点上的本地写入操作已完成。在强制故障转移的情况下，可能会发生数据丢失。备用节点上的数据在故障转移后是一致的；但是，在崩溃之前执行的最新更新可能会丢失。Protocol A最常用于远程复制场景。当与DRBD Proxy结合使用时，它是一个有效的灾难恢复解决方案。有关详细信息，请参见通过DRBD代理进行远程复制。

Protocol B

内存同步（半同步）复制协议。一旦本地磁盘写入发生，并且复制数据包已到达对等节点，则认为主节点上的本地写入操作已完成。通常，在强制故障转移的情况下不会丢失任何写操作。但是，如果同时在主备节点 **同时** 发生了电源故障，则主节点的数据存储可能会丢失最新完成的写入操作。

Protocol C

同步复制协议。只有在确认本地和远程磁盘写入之后，主节点上的本地写入操作才被视为完成。因此，单个节点的丢失不会导致任何数据丢失。数据丢失当然是不可避免的，即使采用这个复制协议，如果所有节点（例如:它们的存储子系统）同时遭到不可逆转的破坏, 数据也可能丢失。

到目前为止，DRBD设置中最常用的复制协议是protocol C。

复制协议的选择影响部署的两个因素： *保护* 和 *延迟* 。相比之下， *吞吐量* 在很大程度上独立于所选的复制协议。

有关演示复制协议配置的资源配置示例，请参见配置资源。

2.4. 不只双向冗余(2-way redundancy)

使用DRBD 9，可以将数据同时存储在两个以上的集群节点上。

在以前通过stacking来实现不只双向冗余是可能的，但在DRBD 9中，目前直接就可以支持多达16个节点。（实际上，通过DRBD使用3、4乃至5路冗余将使其他事情成为停机的主要原因。）

与stacking解决方案的主要区别在于性能损失更小，因为只使用了一个级别的数据复制。

2.5. 自动提升资源

在DRBD 9之前，可以使用 drbdadm primary 命令提升资源。在DRBD 9下，当启用 auto promote 选项时，DRBD将自动将资源提升为主角色，并装入或打开其中一个卷进行写入。一旦卸载或关闭所有卷，资源的角色就变回次要角色。

只有在群集状态允许时（即，如果显示 drbdadm primary 命令成功），自动提升才会成功。否则，安装或打开设备将失败，就像在DRBD 9之前一样。

2.6. 多种复制传输

DRBD支持多种网络传输。到目前为止，有两种传输实现可用：TCP和RDMA。每个传输都由对应的内核模块来实现。

2.6.1. TCP传输

drbd_transport_tcp.ko 传输实现包含在drbd本身的分发文件中。顾名思义，这个传输实现使用TCP/IP协议在机器之间移动数据。

DRBD的复制和同步框架套接字层支持多个低级传输：

IPv4上的TCP

这是规范实现，也是DRBD的默认实现。它可以在任何启用了IPv4的系统上使用。

IPv6上的TCP

当配置为使用标准的TCP套接字进行复制和同步时，DRBD还可以使用IPv6作为其网络协议。这在语义和性能上等同于IPv4，尽管它使用不同的寻址方案。

SDP

SDP是BSD风格的套接字的实现，用于支持RDMA的传输，如InfiniBand。SDP是大多数发行版OFED堆栈的一部分，但现在 **被视为已弃用** 。SDP使用IPv4风格的寻址方案。SDP通过InfiniBand互连使用，为DRBD提供了一个高吞吐量、低延迟的复制网络。

SuperSockets

SuperSockets用一个单一的、单片的、高效的、支持RDMA的套接字实现来替换堆栈的TCP/IP部分。DRBD可以使用这种套接字类型进行非常低延迟的复制。SuperSockets必须运行在特定的硬件上，而这些硬件目前只能从一家供应商Dolphin Interconnect Solutions获得。

2.6.2. RDMA传输

此外，LINBIT提供 drbd_transport_rdma.ko 内核模块。此传输使用verbs/RDMA API在InfiniBand HCAs、支持iWARP的NICs或支持RoCE的NICs上移动数据。与BSD sockets API（由TCP/IP使用）相比，verbs/rdmaapi允许在很少的CPU参与下移动数据。

2.6.3. 结论

在高传输速率下，tcp传输的CPU负载/内存带宽可能成为限制因素。使用rdma传输和适当的硬件，您可能可以获得更高的传输速率。

可以为资源的每个连接配置传输层实现。有关详细信息，请参见配置传输实现。

2.7. 高效同步

（重新）同步不同于设备复制。当对主角色中的资源的任何写入事件发生复制时，同步与传入的写操作是分离的。否则，它会影响到设备整体。

如果复制链路由于任何原因中断（无论是由于主节点故障、辅助节点故障还是复制链路中断）时，则必须进行同步。同步是有效的，因为DRBD不会按照修改后的块最初写入的顺序，而是按照线性顺序同步，这会产生以下后果：

- 同步速度很快，因为发生多个连续写操作的块只同步一次。
- 同步还与很少的磁盘查找相关，因为块是根据磁盘上块的自然布局进行同步的。
- 在同步过程中，备用节点上的数据集部分过时，部分已更新。这种数据状态称为 *不一致* 。

后台同步正在进行时，服务继续在活动节点上不间断地运行。

数据不一致的节点通常不能投入运行，因此希望尽可能缩短节点不一致的时间段。基于此，DRBD附带了LVM集成工具，用于在同步之前自动创建LVM快照。这可以确保对等机上始终可以使用数据的一致性副本，即使同步正在运行。有关使用此工具的详细信息，请参见在DRBD同步期间使用自动LVM快照。

2.7.1. 可变速率同步

在可变速率同步（8.4版本后被设为默认值）中，DRBD检测同步网络上的可用带宽，将其与传入的前台应用程序I/O进行比较，并基于全自动控制回路选择适当的同步速率。

有关可变速率同步的配置建议，请参见可变同步速率配置。

2.7.2. 固定速率同步

在固定速率同步中，每秒传送到同步对等机的数据量（同步速率）有一个可配置的静态上限。基于此限制，您可以根据以下简单公式估计预期的同步时间：

$$t_{resync} = \frac{D}{R}$$

插图 2. 同步时间

t_{sync} 是预期的同步时间。D 是要同步的数据量，这两者很难人为干预（这是复制链接断开时应用程序修改的数据量）。R 是可配置的同步速率，受限于复制网络和I/O子系统的吞吐量限制。

有关固定速率同步的配置建议，请参见配置同步速率。

2.7.3. 基于校验和的同步

通过使用数据摘要（也称为校验和）可以进一步提高DRBD的同步算法的效率。当使用基于校验和的同步时，DRBD在同步前先 *读取* 块，并计算当前在磁盘上找到的内容的哈希，而不是对标记为不同步的块执行强制覆盖。然后，它将此哈希与从对等机上的同一扇区计算的哈希进行比较，如果哈希匹配，则省略重新写入此块。如果文件系统在DRBD处于断开模式时用相同的内容重新写入了扇区，这可以显著缩短同步时间。

有关同步的配置建议，请参见配置基于校验和的同步。

2.8. 已挂起的复制

如果配置正确，DRBD可以检测复制网络是否拥塞，在这种情况下可以暂停复制。在这种模式下，主节点会"超前"于次节点 即 暂时不同步，但仍在次节点上保留一致的副本。当有更多带宽可用时，复制将自动恢复并进行后台同步。

挂起复制通常在具有可变带宽的链接上启用，例如在数据中心或云实例之间的共享连接上启用广域复制。

有关拥塞策略和挂起复制的详细信息，请参见配置拥塞策略和挂起复制。

2.9. 在线设备验证

在线设备验证使用户能够以非常有效的方式在节点之间执行逐块数据完整性检查。

注意 *efficient* 是指的是网络带宽的有效使用，以及验证不会以任何方式破坏冗余的事实。在线验证仍然是一个资源密集型操作，对CPU利用率和平均负载有显著影响。

在线验证由一个节点（ *verification source* ）按顺序计算存储在特定资源的低级存储设备上的每个块的密码摘要。然后，DRBD将该摘要发送到对等节点（即 *verification targets(s)*），在对等节点中，根据受影响块的本地副本摘要对其进行检查。如果摘要不匹配，则块被标记为不同步，以后可能会同步。因为DRBD只传输摘要，而不是完整的块，所以在线验证能非常有效地使用网络带宽。

该过程被称为在线验证， *on-line* 验证，因为它不要求被验证的DRBD资源在验证时未被使用。因此，尽管在线验证在运行时确实会带来轻微的性能损失，但它不会导致服务中断或系统停机—无论是在验证运行期间还是在随后的同步期间。

由本地cron守护进程管理 on-line 验证是一个常见的用例，例如，每周或每月运行一次。有关如何启用、调用和自动执行联机验证的信息，请参见使用在线设备验证。

2.10. 复制流量完整性检查

DRBD可以选择使用加密消息摘要算法（如MD5、SHA-1或CRC-32C）执行端到端消息完整性检查。

这些消息摘要算法 **不是** 由DRBD提供的，而是由Linux内核加密API提供的；DRBD只是使用它们。因此，DRBD能够利用特定系统内核配置中可用的任何消息摘要算法。

启用此功能后，DRBD生成复制到对等方的每个数据块的消息摘要，然后对等方使用该摘要来验证复制包的完整性。如果无法根据摘要验证复制的块，则会断开连接并立即重新建立；由于位图的存在，典型的结果是重新传输。因此，DRBD复制可以避免一些错误的源资源，如果对这些错误源执行检查，则在复制过程中可能会导致数据损坏：

- 在主存储器和发送节点上的网络接口之间传输的数据上会发生按位错误（"位翻转"）（如果将其卸载到网卡，则TCP校验和无法检测到该错误，这在最近的实现中很常见）；
- 在从网络接口传输到接收节点的主存储器的数据上发生的位翻转（同样的考虑适用于TCP校验和卸载）；
- 由于网络接口固件或驱动程序中的竞争条件或错误而导致的任何形式的损坏；
- 通过在节点之间重新组装网络组件（如果不使用直接的背对背连接）注入的位翻转或随机损坏。

有关如何启用复制通信完整性检查的信息，请参见配置复制通信完整性检查。

2.11. 裂脑通知和自动恢复

裂脑是指由于群集节点之间的所有网络链路的暂时故障，以及可能由于群集管理软件的干预或人为错误，两个节点在断开连接时都切换到主要角色的情况。这是一种潜在的有害状态，因为它意味着对数据的修改可能是在任一节点上进行的，而没有复制到对等节点。因此，在这种情况下，很可能已经创建了两个不同的数据集，这些数据集不能简单地合并。

DRBD split brain is distinct from cluster split brain, which is the loss of all connectivity between hosts managed by a distributed cluster management application such as Pacemaker. To avoid confusion, this guide uses the following convention:

- *裂脑* 是指如上文所述的DRBD裂脑。
- 所有集群连接的丧失被称为 *集群分片*，是集群裂脑的另一个术语。

DRBD允许在检测到脑裂时自动通知管理员（通过电子邮件或其他方式）。有关如何配置此功能的详细信息，请参见裂脑通知。

虽然在这种情况下，建议的操作过程是manually resolve处理脑裂，然后消除其根本原因，但在某些情况下，可能需要自动化该过程。DRBD有几种可用的解决方案列举如下：

- **放弃较新主节点上的修改。** 在这种模式下，当重新建立网络连接并发现裂脑时， DRBD在_最后_切换到主模式的节点上，会放弃在该节点上的更改。
- **放弃较旧主节点上的修改。** 在这种模式下，DRBD会放弃先前切换到 主模式的节点上所做的修改。
- **放弃改动小的主节点上的修改。** 在这种模式下， DRBD将检查两个节点中的哪个节点记录了较少的修改，然后将丢弃在该主机上进行的所有修改。
- **如果任一节点没有中间变化，则可以从裂脑中恢复。** 在这种模式下， 如果其中一个主机在裂脑期间根本没有进行任何修改，则DRBD会简单地恢复正常并宣布裂脑已解决。 请注意，这是一个不太可能发生的情况。即使两个主机都仅将文件系统安装在DRBD块设备上（甚至是只读的）， 设备的内容通常也会被修改（例如，通过文件系统日志重播），从而排除了自动恢复的可能性。

自动裂脑恢复是否可以接受在很大程度上取决于运行的应用。以DRBD托管数据库为例。 对于web应用程序使用的点击型数据库，使用放弃改动小的主节点上的修改的方法可能很好。 相比之下，自动放弃对财务数据库所做的任何修改可能是完全不可接受的，在任何脑分裂事件中都需要手动恢复。 在启用自动裂脑恢复之前，请仔细考虑应用程序的要求。

请参阅自动裂脑恢复策略以了解有关配置DRBD的自动裂脑恢复策略的详细信息。

2.12. 支持磁盘刷新

当本地块设备（如硬盘驱动器或RAID逻辑磁盘）启用了写缓存时，对这些设备的写入在到达易失性缓存后即被视为完成。控制器制造商通常将此称为回写模式，反之则为直写模式。如果控制器在回写模式下断电，则最后一次写入永远不会提交到磁盘，可能会导致数据丢失。

为了解决这个问题，DRBD使用了磁盘刷新。磁盘刷新是一种写入操作，仅当关联的数据已提交到稳定（非易失性）存储时才完成，也就是说，它已有效地写入磁盘，而不是缓存。DRBD使用磁盘刷新对其复制的数据集和元数据进行写操作。实际上，DRBD在其认为必要的情况下绕过写缓存，如在activity log更新或强制隐式写后依赖项中。这意味着即使在停电的情况下，也会有额外的可靠性。

需要了解的是，DRBD只能在落地于支持磁盘刷新的备份设备上时才能使用磁盘刷新，这一点很重要。最新的内核支持大多数SCSI和SATA设备的磁盘刷新。Linux软件RAID（md）支持RAID-1的磁盘刷新，前提是所有组件设备也支持磁盘刷新。设备映射器(device-mapper)设备（LVM2、dm raid、多路径）也是如此。

带电池备份的写缓存（BBWC）的控制器使用电池备份其易失性存储。在这些设备上，当断电后恢复供电时，控制器会从电池支持的高速缓存中清除所有挂起的写入到磁盘的操作，确保提交到易失性高速缓存的所有写入实际上都转移到稳定存储中。在这些设备上运行DRBD时，可以禁用磁盘刷新，从而提高DRBD的写入性能。有关详细信息，请参见禁用备份设备刷新。

2.13. Trim/Discard支持

Trim/*Discard* 是同一功能的两个名称：对存储系统的请求，告诉它某些数据区域不再使用 [2]，并且可以被回收。+ 此调用源于基于闪存的存储（固态硬盘、FusionIO卡等），这些存储不能轻易重写 *rewrite* 扇区，而是必须擦除后并再次写入（新）数据（产生一定的延迟成本）。有关更多详细信息，请参见 [[https://en.wikipedia.org/wiki/Trim_%28computing%29,wikipedia page]].

从8.4.3版开始， DRBD包含了对 *Trim*/*Discard* 的支持。您不需要配置或启用任何东西；如果DRBD检测到本地（底层）存储系统允许使用这些命令，它将透明地启用这些命令并传递这些请求。

其效果是，例如，在一个有很多TB的卷上使用一个新版本的 mkfs.ext4 时，可以将初始同步时间缩短到几秒到几分钟，只要告诉DRBD（它将把该信息中继到所有连接的节点），大多数/所有存储现在都将被视为无效。

稍后连接到该资源的节点将不会看到 *Trim*/*Discard* 请求，因此将启动完全重新同步；不过，依赖于内核版本和文件系统，对`fstrim`的调用可能会给出所需的结果。

即使您没有支持 *Trim*/*Discard* 的存储，一些虚拟块设备也会为您提供相同的功能，例如精简LVM。

2.14. 磁盘错误处理策略

如果被用作其中一个节点上DRBD的备份块设备的硬盘发生故障，DRBD可以将I/O错误传递到上层（通常是文件系统），也可以屏蔽上层的I/O错误。

传递I/O错误

如果DRBD被配置为传递I/O错误，那么在低级设备上发生的任何此类错误都将透明地传递到上层I/O层。因此，由上层处理这些错误（例如，这可能导致文件系统以只读方式重新装载）。此策略无法确保服务的连续性，因此不建议大多数用户使用。

掩蔽I/O错误

如果DRBD被配置为在低层I/O错误时分离，即 *detach* ，DRBD将在第一个低层I/O错误发生时自动这样做。当DRBD通过网络从对等节点透明地获取受影响的块时，I/O错误从上层被屏蔽。从那时起，DRBD被称为在无盘模式下操作，并且仅在对等节点上执行所有后续的读写I/O操作。此模式下的性能将降低，但服务不会中断，并且可以在方便的时候经过验证后再移动到对等节点。

有关为DRBD配置I/O错误处理策略的信息，请参见配置I/O错误处理策略。

2.15. 处理过时数据的策略

DRBD能区分不一致数据和过时数据。不一致数据是指不能期望以任何方式访问和使用的数据。举例说明：举某节点上的数据为例，该节点当前是正在进行的同步的目标。这样一个节点上的数据部分是过时的，部分是最新的，不可能被识别为二者之一。因此，例如，如果设备持有一个文件系统（通常情况下是这样），那么该文件系统将无法挂载，甚至无法通过自动文件系统检查。

相比之下，过时的数据是次节点(sencondary node)上的一致数据，但不再与主节点同步。这将发生在复制链接的任何中断中，无论是临时的还是永久的。过时的、断开连接的辅助节点上的数据应该是干净的，但它反映了过去某个时间对等节点的状态。为了避免服务使用过时的数据，DRBD不允许promoting a resource。

DRBD的接口允许外部应用程序在网络中断发生时立即使辅助节点失效。DRBD将拒绝将节点切换到主角色，从而防止应用程序使用过时的数据。此功能的完整实现可参阅Pacemaker cluster management framework（它使用与DRBD复制链接分离的通信通道）。但是，这些接口是通用的，并且很容易被任何其他集群管理应用程序使用。

每当过时的资源重新建立其复制链接时，其过时标志将自动清除。参阅background synchronization。

2.16. 三路叠加复制

在DRBD版本8.3.0及更高版本中可用；在DRBD版本9.x中不推荐使用，因为更多节点可以在单个级别上实现。有关详细信息，请参见定义网络连接。

当使用三向复制时，DRBD会将第三个节点添加到现有的2节点群集，并将数据复制到该节点，在该节点上，数据可用于备份和灾难恢复目的。这种类型的配置通常涉及通过DRBD代理进行远程复制。

三向复制通过在保存生产数据的现有资源上添加另一个堆叠 *stacked* 的DRBD资源来工作，如下图所示：

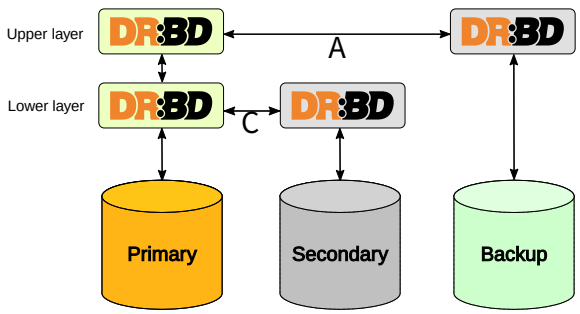


插图 3. DRBD资源堆叠

堆叠(stack)资源使用异步复制（DRBD Protocol A）进行复制，而生产数据通常使用同步复制（DRBD Protocol C）。

三路复制可以永久使用，其中第三个节点将使用生产集群中的数据不断更新。或者，它也可以按需使用，其中生产集群通常与备份站点断开连接，并且定期执行站点到站点的同步，例如通过运行夜间cron作业。

2.17. 通过DRBD代理进行远程复制

DRBD的protocol A是异步的，但是一旦套接字输出缓冲区满了，写入应用程序就会阻塞（请参见DRBD.conf 手册页中的 sndbuf size 选项）。在这种情况下，写入应用程序必须等到某些写入的数据, 这些数据可能要通过可能很小的带宽网络，而使得链路耗尽。

平均写入带宽受网络链路可用带宽的限制。只有配置好受限的套接字输出缓冲区时，突发写入才能被优雅的处理。

您可以通过DRBD Proxy的缓冲机制来缓解这种情况。DRBD代理将从主节点上的DRBD设备将更改的数据放入其缓冲区。DRBD代理的缓冲区大小是可以自由配置的，仅受地址空间大小和可用物理RAM的限制。

可选地，可以配置DRBD代理来压缩和解压缩它转发的数据。压缩和解压缩DRBD的数据包可能会稍微增加延迟。然而，当网络链路的带宽是限制因素时，缩短传输时间的增益大于压缩和解压缩开销。

压缩和解压缩是在考虑多核SMP系统的情况下实现的，可以利用多个CPU核。

事实上，大多数块I/O数据压缩得非常好，因此对带宽的使用更加有效,这证明了即使使用DRBD Protocol B和C也可以使用DRBD代理。

有关配置drbd proxy的信息，请参见使用DRBD代理。

DRBD Proxy是DRBD产品系列中少数未在开源许可下发布的部分之一。请联系sales@linbit.com 或 sales_us@linbit.com 获取评估许可证。

2.18. 基于卡车的复制

基于卡车的复制，也称为磁盘传送，是通过物理方式将存储介质传送到远程站点，为远程站点提供要复制的数据的一种方式。这特别适用于

- 要复制的数据总量相当大（超过几百GB）；
- 要复制的数据的预期变化率不太大；
- 站点之间的可用网络带宽是有限的。

在这种情况下，如果没有基于卡车的复制，DRBD将需要非常长的初始设备同步（按周、月或年的顺序）。基于卡车的复制允许将数据种子传送到远程站点，因此大大缩短了初始同步时间。有关此用例的详细信息，请参见使用基于卡车的复制。

2.19. 浮动节点

此功能在DRBD 8.3.2及以上版本中可用。

DRBD的一个有点特殊的用例是 *浮动节点* 。在浮动节点设置中，DRBD节点不绑定到特定的命名主机（如在传统配置中），而是能够在多个主机之间浮动。在这种配置中，DRBD通过IP地址而不是主机名来标识对方。

有关管理浮动对等配置的更多信息，请参见配置DRBD以在两个支持SAN的Pacemaker集群之间进行复制。

2.20. 数据重新平衡（水平存储扩展）

如果公司的政策规定需要3路冗余，则安装时至少需要3台服务器。

现在，随着存储需求的增长，您将需要额外的服务器。比起同时购买3台以上的服务器的方案，您也可以在一个额外的节点上重新平衡数据。

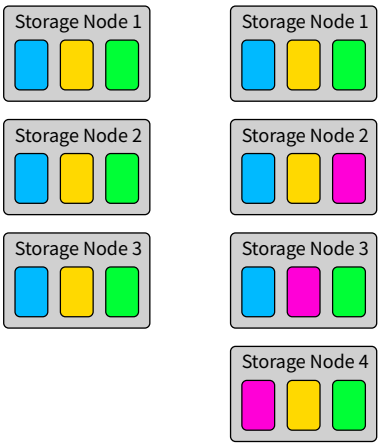


插图 4. DRBD数据重新平衡

在上图中，您可以看到 *before* 和 *after* 状态：从3个节点（每个节点有3个25TiB卷）(净75TiB）到4个节点（净100TiB）。

DRBD 9使在线实时迁移数据成为可能；请参见数据再平衡了解所需的确切步骤。

2.21. DRBD客户端

通过DRBD的多节点特性，添加了许多有趣的用例，例如 *DRBD 客户端*。

基本思想是DRBD *后端* 可以由3、4或更多节点组成（取决于所需冗余的策略）；但是，由于DRBD 9可以连接更多的节点。DRBD除了作为存储复制之外，还作为存储访问协议工作。

在主 *DRBD客户端* 上执行的所有写请求都会被发送到配备了存储设备的所有节点。读取请求仅传送到其中一个服务器节点。*DRBD客户端* 将在所有可用的服务器节点之间平均分配读取请求。

2.22. Quorum(仲裁)

为了避免脑裂或数据分散，必须配置围栏(fencing)。事实证明，在实际部署中，节点围栏并不流行，因为在规划或部署时经常会出错。

当一个数据集有3个副本时，可以依赖DRBD中的仲裁实现，而不是依赖Pacemake级别的围栏。Pacemaker通过资源的master分数获得有关Quorum(仲裁)或Quorum(仲裁)丢失的通知。

DRBD的quorum(仲裁)可以用于任何基于Linux的服务。如果一个服务在出现IO-错误的时候终止，quorum(仲裁）丢失的行为会非常优雅。如果服务没有在IO-错误时出现终止，则需要将系统设置为丢失Quorum(仲裁)后重启主节点。

有关详细信息，请参见配置仲裁。

2.22.1. 断路器(Tiebreaker)

quorum tiebreaker功能在DRBD版本9.0.18及更高版本中可用。

两节点集群的根本问题是，当它们失去连接时，我们有两个分区，而它们都没有仲裁，这导致集群停止服务。这个问题可以通过在集群中添加第三个无磁盘节点来缓解，该节点将充当仲裁层断路器。

有关详细信息，请参见使用无盘节点作为分层断路器。

2.23. VCS的DRBD集成

Veritas Cluster Server (or Veritas Infoscale Availability) is a commercial alternative to the Pacemaker open source software. In case you need to integrate DRBD resources into a VCS setup please see the README in drbd-utils/scripts/VCS on github.

[1] 也许该功能稍后会在实际工作时重命名为"Multi Primary" ;)
[2] 例如，已删除文件的数据

建立和安装DRBD软件

3. 安装预先构建的DRBD二进制包

3.1. LINBIT提供的包

LINBIT, the DRBD project’s sponsor company, provides binary packages to its commercial support customers. These packages are available via repositories (e.g., apt, yum), and when reasonable via LINBIT’s docker registry. Packages/images from these sources are considered "official" builds.

这些版本可用于以下发行版：

- Red Hat Enterprise Linux（RHEL），版本6、7和8
- SUSE Linux企业服务器（SLES），11SP4、12和15版
- Debian GNU/Linux，9（stretch）和10（buster）
- Ubuntu Server Edition LTS 16.04 (Xenial Xerus), LTS 18.04 (Bionic Beaver), and LTS 20.04 (Focal Fossa).

一些其他发行版的软件包也已经构建好，但是未经充分测试。

LINBIT与DRBD在source release中也同时发布了二进制版本。

在基于RPM的系统（SLES，RHEL）上安装包的方法是简单地调用 yum install （用于新安装）或 yum update （用于升级）。

对于基于Debian的系统（Debian GNU/Linux，Ubuntu）系统，drbd-utils 和 drbd-module-`uname -r` 包可以使用 apt, 或类似的工具，如 aptitude 或 synaptic 安装。

3.1.1. Secure Boot / Kernel module signing

Starting from DRBD version 8.4.12/9.0.25 LINBIT signs its kernel module object files. For DRBD 8.4 this is available for RHEL 7, for DRBD9 this is available for RHEL 7/8.

The public signing key is shipped in the rpm package and gets installed to /etc/pki/linbit/SECURE-BOOT-KEY-linbit.com.der. It can be enrolled via:

```
# mokutil --import /etc/pki/linbit/SECURE-BOOT-KEY-linbit.com.der
input password:
input password again:
```

A password can be chosen freely. It will be used when the key is actually enrolled to the MOK list after the required reboot.

3.2. LINBIT提供的Docker镜像

LINBIT为其商业支持客户提供Docker registry。registry可通过主机名 drbd.io 访问。在拉取镜像之前，必须登录该registry：

```
# docker login drbd.io
```

成功登录后，可以拉取镜像。要测试登录名和registry，请首先输入以下命令：

```
# docker pull drbd.io/alpine
# docker run -it --rm drbd.io/alpine # 按 CTRL-D 退出
```

3.3. 发行版提供的包

许多发行版提供DRBD，包括预构建的二进制包。由发行版提供商提供对这些包的支持（如果有的话）。它们的发布周期可能落后于DRBD源代码发布。

3.3.1. SUSE Linux企业服务器

SLES高可用性扩展（HAE）包括DRBD。

在SLES上，DRBD通常通过YaST2的软件安装组件安装。它与高可用性包选项捆绑在一起。

喜欢命令行安装的用户只需输入：

```
# yast -i drbd
```

或

```
# zypper install drbd
```

3.3.2. CentOS

CentOS从第5版开始就有了DRBD 8；对于DRBD 9，您需要查看EPEL和类似的源代码。

可以使用 yum 安装DRBD（请注意，要配置好正确的repository才能正常工作）：

```
# yum install drbd kmod-drbd
```

3.3.3. Ubuntu Linux

对于Ubuntu LTS，LINBIT在https://launchpad.net/~LINBIT/+archive/Ubuntu/LINBIT-drbd9-stack上提供了一个PPA存储库。有关更多信息，请参见 Adding Launchpad PPA Repositories

```
# apt install drbd-utils drbd-dkms
```

3.4. 从源代码编译包

由 github上 的git标记生成的版本是git存储库在给定时间的快照。你很可能不想用这些。因为它们可能缺少生成的手册页、configure 脚本和其他生成的文件。如果您想从tarball构建，请使用 https://www.linbit.com/en/drbd-community/drbd download/[由我们提供的档案]。

我们所有的项目都包含标准的构建脚本（例如，Makefile、configure）。维护每个发行版的特定信息（例如，编译文档时损坏的宏等）太麻烦了，而且从历史上看，本节中提供的信息很快就过时了。如果你不知道如何以标准的方式构建软件，请考虑使用LINBIT提供的软件包。

使用DRBD

4. 常见管理任务

本章概述了日常运营中遇到的典型管理任务。它不包括故障排除任务，故障排除任务在故障排除和错误恢复中有详细介绍。

4.1. 配置DRBD

4.1.1. 准备底层存储

安装完DRBD之后，必须在两个集群节点上留出大致相同大小的存储区域。这将成为DRBD资源的底层设备。为此，您可以使用系统上找到的任何类型的块设备。典型示例包括：

- 硬盘分区（或完整的物理硬盘驱动器），
- 一个软件RAID设备，
- LVM逻辑卷或由Linux device-mapper配置出的任何其他块设备，
- 在系统上找到的任何其他块设备类型。

您也可以使用 *资源堆叠* ，这意味着您可以将一个DRBD设备用作另一个底层设备。使用堆叠资源有一些特定注意事项；它们的配置在创建堆叠的三节点设置中有详细介绍。

虽然可以使用环路设备(loop devices)作为DRBD的底层设备，但由于死锁问题，不建议这样做。

在某个存储区域中创建DRBD资源之前，*不* 需要将其设置为空。事实上，通常利用DRBD从以前的非冗余单服务器系统中创建出双节点集群。（如果您计划这样做，请参考DRBD元数据）。

在本指南中，我们假设一个非常简单的设置：

- 两台主机都有一个名为 /dev/sda7 的可用（当前未使用）分区。
- 我们选择使用internal meta data。

4.1.2. 准备网络配置

虽然不是严格要求，但建议您通过专用连接运行DRBD复制。在撰写本文时，最合理的选择是直接、背对背(back-to-back)、千兆以太网连接。当DRBD在交换机网络上运行时，建议使用冗余组件和 *bonding* 驱动程序（在 *active-backup* 模式下）。

通常不建议通过路由器网络运行DRBD复制，因为存在相当明显的性能缺陷（对吞吐量和延迟都有不利影响）。

就本地防火墙的考虑而言，重要的是要理解DRBD（按惯例）使用7788以上的TCP端口，每个资源都在一个单独的端口上监听。DRBD对每个配置的资源使用两个TCP连接。为了实现正确的DRBD功能，防火墙配置必须允许这些连接。

如果启用了诸如SELinux或AppArmor之类的强制访问控制（MAC）方案，则除防火墙之外的安全考虑也可能适用。您可能需要调整本地安全策略，使其无法使DRBD正常工作。

当然，您还必须确保DRBD的TCP端口尚未被其他应用程序使用。

目前还不能配置DRBD资源来支持多个TCP连接对。如果您想提供DRBD连接负载平衡或冗余，那么您可以在以太网级别（再次提醒，使用 *bonding* 设备）轻松地做到这一点。

在本指南中，我们假设一个非常简单的设置：

- 我们的两台DRBD主机都有一个当前未使用的网络接口eth1，其IP地址分别为 10.1.1.31 和 10.1.1.32 。
- 无其他服务在任一主机上使用TCP端口7788到7799。

· 本地防火墙配置允许主机之间通过这些端口进行入站和出站TCP连接。

4.1.3. 配置资源

DRBD的所有选项都在其配置文件 /etc/DRBD.conf 中进行控制。通常，此配置文件只是具有以下内容的框架：

```
include "/etc/drbd.d/global_common.conf";
include "/etc/drbd.d/*.res";
```

按照惯例，/etc/drbd.d/global_common.conf 包含drbd配置的global和common部分，而 .res 文件包含每个resource部分。

也可以使用 drbd.conf 作为一个单一配置文件，而不使用任何 include 语句。然而，这样的配置很快就会变得杂乱无章，难以管理，这就是为什么多文件方法是首选的方法。

无论采用哪种方法，都应该始终确保所有参与集群节点上的 drbd.conf 及其包含的任何其他文件 *完全相同* 。

DRBD 源码压缩包解压后， 在 scripts 子目录中包含一个示例配置文件。二进制安装包将直接在 /etc ，或在 /usr/share/doc/packages/drbd 等由安装包定义出的文档目录中安装此示例配置。

本节仅描述配置文件的几个部分，这些部分对于使DRBD启动和运行是绝对必要的。配置文件的语法和内容在 drbd.conf 的手册页中有详细的说明。

示例配置

在本指南中，我们假设按照前面章节中给出的示例进行最小设置：

Listing 1. 简单的DRBD配置（ /etc/DRBD.d/global_common.conf）

```
global {
 usage-count yes;
}
common {
 net {
  protocol C;
 }
}
```

Listing 2. 简单的DRBD资源配置(/etc/drbd.d/rc0.res)

```
resource r0 {
 on alice {
  device    /dev/drbd1;
  disk      /dev/sda7;
  address   10.1.1.31:7789;
  meta-disk internal;
 }
 on bob {
  device    /dev/drbd1;
  disk      /dev/sda7;
  address   10.1.1.32:7789;
  meta-disk internal;
 }
}
```

此示例按以下方式配置DRBD：

- 您 "决定参加" 将包含在DRBD的使用统计信息中 (参见 usage-count)。
- 除非另有明确规定，否则资源配置为使用完全同步复制（Protocol C）。
- 我们的集群由两个节点 alice 和 bob 组成。
- 我们有一个命名为 r0 的资源，它使用 /dev/sda7 作为低级设备，并配置了internal meta data。
- 资源使用TCP端口7789进行网络连接，并分别绑定到IP地址10.1.1.31和10.1.1.32。（这将隐式定义所使用的网络连接。）

上面的配置隐式地在资源中创建一个卷，编号为0（ 0 ）。对于一个资源中的多个卷，请按如下所示修改语法（假设两个节点上使用相同的底层存储块设备）：

Listing 3. 多卷DRBD资源配置（ /etc/drbd.d/rc0.res）

```
resource r0 {
  volume 0 {
    device    /dev/drbd1;
    disk      /dev/sda7;
    meta-disk internal;
  }
  volume 1 {
    device    /dev/drbd2;
    disk      /dev/sda8;
    meta-disk internal;
  }
  on alice {
    address   10.1.1.31:7789;
  }
  on bob {
    address   10.1.1.32:7789;
  }
}
```

卷也可以动态添加到现有资源中。有关示例，请参见将新的DRBD卷添加到现有卷组。

global 部分

此部分在配置中只允许使用一次。它通常位于 /etc/drbd.d/global_common.conf 文件中。在单个文件配置中，它应该位于配置文件的最顶端。在本节提供的少数选项中，只有一个与大多数用户相关：

usage-count

DRBD项目保存关于各种DRBD版本使用情况的统计信息。这是通过每次在系统上安装新的DRBD版本时联系官方HTTP服务器来完成的。这可以通过设置 usage count no； 禁用。默认值是 usage count ask； ，每次升级DRBD时都会提示您。

当然，DRBD的使用统计数据是公开的：参见http://usage.DRBD.org。

common 部分

本节提供了一个速记方法来定义每个资源继承的配置设置。它通常位于 /etc/drbd.d/global_common.conf 。您可以定义任何选项，也可以基于每个资源定义。

严格来说，不需要包含 common 部分，但如果您使用多个资源，则强烈建议您这样做。否则，重复使用的选项会使配置很快变得复杂。

在上面的示例中，我们在 common 部分中包含了 net {protocol C; } ”，因此每个配置的资源（包括 r0 ）都

继承此选项，除非它显式配置了另一个 protocol 选项。有关可用的其他同步协议，请参见复制(Replication)模式。

resource 部分

每个资源配置文件通常命名为 /etc/drbd.d/*resource*.res 您定义的任何DRBD资源都必须通过在配置中指定资源名称来命名。惯例是只使用字母、数字和下划线；虽然在技术上也可以使用其他字符，但如果碰巧例如 *peer@resource/volume* 格式的命名时，可能会让您困惑不已。

每个资源配置还必须至少有两个 on *host* 子字节，每个群集节点一个。所有其他配置设置要么继承自 common 部分（如果存在），要么派生自DRBD的默认设置。

此外，可以在 resource 部分直接指定在所有主机上具有相同值的选项。因此，我们可以进一步压缩示例配置如下：

```
resource r0 {
  device    /dev/drbd1;
  disk      /dev/sda7;
  meta-disk internal;
  on alice {
    address   10.1.1.31:7789;
  }
  on bob {
    address   10.1.1.32:7789;
  }
}
```

4.1.4. 定义网络连接

目前，DRBD 9中的通信链路必须建立一个完整的网格，即在每个资源中，每个节点都必须与每个其他节点（当然，不包括自身）有直接连接。

对于两台主机的简单情况， drbdadm 将自行插入（单个）网络连接，以便于使用和向后兼容。

其净效果是主机上网络连接的二次方数量。对于"传统"的两个节点，需要一个连接；对于三个主机，有三个节点对；对于四个主机， 有六个节点对；对于五个主机，有十个连接，依此类推。对于（当前的）最多16个节点，将有120个主机对需要连接。

$$C = \frac{N(N-1)}{2}$$

插图 5. N 个主机的连接数

三台主机的配置文件示例如下：

```
resource r0 {
  device    /dev/drbd1;
  disk      /dev/sda7;
  meta-disk internal;
  on alice {
    address   10.1.1.31:7000;
    node-id   0;
  }
  on bob {
    address   10.1.1.32:7000;
    node-id   1;
  }
  on charlie {
    address   10.1.1.33:7000;
    node-id   2;
  }
  connection-mesh {
    hosts alice bob charlie;
  }
}
```

如果服务器中有足够的网卡，则可以在服务器对之间创建直接交叉链接。一个四端口以太网卡允许有一个单一的管理接口，并连接3个其他服务器，以获得4个群集节点的完整网格。

在这种情况下，可以指定其他节点的IP地址以使用直接链接：

```
resource r0 {
  ...
  connection {
    host alice   address 10.1.2.1:7010;
    host bob     address 10.1.2.2:7001;
  }
  connection {
    host alice   address 10.1.3.1:7020;
    host charlie address 10.1.3.2:7002;
  }
  connection {
    host bob     address 10.1.4.1:7021;
    host charlie address 10.1.4.2:7012;
  }
}
```

为了便于维护和调试，建议为每个端点使用不同的端口 - 查看 tcpdump 跟踪数据包的结果，从而很容易地将其关联起来。

下面的示例仍将仅使用两台服务器；请参见四个节点的示例配置了解四节点示例。

4.1.5. 配置传输实现

DRBD支持多种网络传输。可以为资源的每个连接配置传输实现。

TCP/IP协议

```
resource <resource> {
 net {
  transport "tcp";
 }
 ...
}
```

tcp 是默认传输协议。即：每个缺少配置传输选项的连接默认使用 tcp 传输。

tcp 传输可以使用以下网络选项进行配置： sndbuf size 、rcvbuf size 、connect int 、 sock check timeo 、ping timeo 、timeout 。

RDMA

```
resource <resource> {
 net {
  transport "rdma";
 }
 ...
}
```

rdma 传输可以使用以下网络选项配置：sndbuf size 、rcvbuf size 、max_buffers 、connect int 、sock check timeo 、ping timeo 、timeout 。

rdma 传输是零拷贝接收传输。这意味着 max_buffers 配置选项必须设置为足以容纳所有 rcvbuf size 的值。

rcvbuf size 以字节为单位配置，max_buffers 以页为单位配置。为了获得最佳性能，max_buffers 应该足够大，可以容纳所有 rcvbuf size 和在任何时间点可能传输到后端设备的数据量。

如果您将InfiniBand HCAs与 rdma 传输一起使用，则还需要配置IPoIB。IP地址不用于数据传输，但用于在建立连接时找到正确的适配器和端口。

只有在建立连接时才考虑配置选项 sndbuf size 、rcvbuf size 。即: 如果你在连接已建立时更改它们。它们将在重新建立连接时生效。

RDMA的性能考虑

通过查看pseudo文件 */sys/kernel/debug/drbd/<resource>/connections/<peer>/transport* ，可以监视可用接收描述符（rx_desc）和传输描述符（tx_desc）的计数。如果某个描述符类型耗尽，则应增加 sndbuf size ”或 rcvbuf size 。

4.1.6. 首次启用资源

在您完成了前面章节中概述的初始资源配置之后，您可以启用您定义的资源。

必须在两个节点上完成以下每个步骤。

请注意，使用我们的示例配置片段（ resource r0 { … })时， <resource> 将是 r0 。

创建设备元数据
此步骤只能在初始设备创建时完成。它初始化DRBD的元数据：

```
# drbdadm create-md <resource>
v09 Magic number not found
Writing meta data...
initialising activity log
NOT initializing bitmap
New drbd meta data block successfully created.
```

请注意，元数据中分配的位图插槽(bitmap slots)数量取决于此资源的主机数量；默认情况下，资源配置中的主机将被计算在内。如果在创建元数据 *之前* 指定了所有主机，这将 "正常工作"；以后可以为更多节点添加位图插槽(bitmap slots)，但需要一些手动操作。

启用资源

此步骤将资源与其备份设备（如果是多卷资源，则为多个设备）关联，设置复制参数，并将资源连接到其对等方：

```
# drbdadm up <resource>
```

运行 drbdadm status *观察状态变化*

drbdsetup 的状态输出现在应该包含类似于以下内容的信息：

```
# drbdadm status r0
r0 role:Secondary
  disk:Inconsistent
  bob role:Secondary
    disk:Inconsistent
```

此时磁盘状态应该是 *Inconsistent/Inconsistent* 。

到目前为止，DRBD已经成功地分配了磁盘和网络资源，并准备就绪。然而它还不知道应该使用哪个节点作为初始设备同步的源。

4.1.7. 初始设备同步

要使DRBD完全运行，还需要两个步骤：

选择初始同步源

如果处理的是新初始化的空磁盘，则此选择完全是任意的。但是，如果您的某个节点已经有需要保留的有价值的数据，则选择该节点作为同步源至关重要。如果在错误的方向上执行初始设备同步，则会丢失该数据。这点要非常小心。

启动初始化全量同步

此步骤只能在一个节点上执行，只能在初始资源配置上执行，并且只能在您选择作为同步源的节点上执行。要执行此步骤，请输入以下命令：

```
# drbdadm primary --force <resource>
```

发出此命令后，将启动初始化全量同步。您将能够通过 drbdadm status 监视其进度。根据设备的大小，可能需要一些时间。

现在，您的DRBD设备已经完全运行，甚至在初始化同步完成之前（尽管性能略有降低）。如果从空磁盘开始，现在可能已经在设备上创建了一个文件系统，将其用作原始块设备，挂载它，并对可访问的块设备执行

任何其他操作。

您现在可能需要继续执行使用DRBD，它描述了要在资源上执行的常见管理任务。

4.1.8. Skipping initial resynchronization

If (and only if) you are starting DRBD resources from scratch (with no valueable data on them) you can use following command sequence to skip initial resync (don’ t do that with data you want to keep on the devices):

On all nodes:

```
# drbdadm create-md <res>
# drbdadm up <res>
```

The command drbdadm status should now show all disks as *Inconsistent*.

Then, on one node execute the following command:

```
# drbdadm new-current-uuid --clear-bitmap <resource>/<volume>
```

或

```
# drbdsetup new-current-uuid --clear-bitmap <minor>
```

Running drbdadm status now shows the disks as *UpToDate* (even tough the backing devices might be out of sync). You can now create a file system on the disk and start using it.

Don’ t do the above with data you want to keep or it gets corrupted.

4.1.9. 使用基于卡车的复制

为了向远程节点预先设定数据，然后保持同步，并跳过初始的全量设备同步，请执行以下步骤。

这假设您的本地节点在主角色中有一个已配置好的但断开连接的DRBD资源。也就是说，设备配置是已经就绪的，两个节点上都存在相同的 drbd.conf 副本，并且您已经在本地节点上发出了用于initial resource promotion的命令⍰—⍰但远程节点尚未连接。

- 在本地节点上，输入以下命令：

  ```
  # drbdadm new-current-uuid --clear-bitmap <resource>/<volume>
  ```

  或

  ```
  # drbdsetup new-current-uuid --clear-bitmap <minor>
  ```

- 创建资源数据及其元数据的一致的、逐字的复本。您可以这样做，例如，从RAID-1镜像中删除热插拔驱动器。当然，您可以用新的驱动器替换它，并重建RAID集，以确保持续的冗余。但删除的驱动器是逐字复制的，现在可以运离现场。如果本地块设备支持快照副本（例如在LVM上使用DRBD），也可以使用 dd 创建该快照的逐位副本。
- 在本地节点上，输入：

```
# drbdadm new-current-uuid <resource>
```

或者匹配的 drbdsetup 命令。

注意，在第二次调用中没有 --clear bitmap 选项。

- 将副本物理传输到远程对等位置。
- 将副本添加到远程节点。这可能再次是插入物理磁盘的问题，或者将已传送数据的按位副本移植到远程节点上的现有存储上。请确保不仅还原或复制了复制的数据副本，而且还原或复制了与之关联的DRBD元数据。如果不这样做，磁盘传送过程就没有意义。
- 在新节点上，我们需要修复元数据中的节点ID，并为这两个节点交换对等节点信息。请参阅以下几行，作为将资源 r0 卷 0 上的节点id从2更改为1的示例。

  必须在卷未使用时执行此操作。

```
V=r0/0
NODE_FROM=2
NODE_TO=1

drbdadm -- --force dump-md $V > /tmp/md_orig.txt
sed -e "s/node-id $NODE_FROM/node-id $NODE_TO/" \
   -e "s/^peer.$NODE_FROM. /peer-NEW /" \
   -e "s/^peer.$NODE_TO. /peer[$NODE_FROM] /" \
   -e "s/^peer-NEW /peer[$NODE_TO] /" \
   < /tmp/md_orig.txt > /tmp/md.txt

drbdmeta --force $(drbdadm sh-minor $V) v09 $(drbdadm sh-ll-dev $V) internal restore-
md /tmp/md.txt
```

NOTE
8.9.7之前的 drbdmeta 无法处理无序的 peer 部分；您需要通过编辑器手动更换配置部分。

- 在远程节点上启动该资源：

```
# drbdadm up <resource>
```

两个节点连接后，它们将不会启动完整的设备同步。相反，现在开始的自动同步只覆盖自调用 drbdadm --clear bitmap new-current-uuid 命令后更改的块。

即使此后 *没有* 任何更改，由于在新的副节点上有Activity Log中提到的因回滚导致的区域覆盖问题，仍可能有一个短暂的同步周期。这可以通过使用checksum-based synchronization来降低影响。

无论资源是常规DRBD资源还是堆栈资源，步骤都是相同的。对于堆叠资源，只需将 -S 或 --stacked 选项添加到 drbdadm 。

4.1.10. 四个节点的示例配置

下面是一个四节点集群的示例。

```
resource r0 {
  device     /dev/drbd0;
  disk       /dev/vg/r0;
  meta-disk   internal;

  on store1 {
    address   10.1.10.1:7100;
    node-id   1;
  }
  on store2 {
    address   10.1.10.2:7100;
    node-id   2;
  }
  on store3 {
    address   10.1.10.3:7100;
    node-id   3;
  }
  on store4 {
    address   10.1.10.4:7100;
    node-id   4;
  }

  connection-mesh {
    hosts     store1 store2 store3 store4;
  }
}
```

如果您想查看 connection mesh 配置的扩展，请尝试 drbdadm dump *<resource>* -v 。

另一个例子是，如果四个节点有足够的接口，可以通过直接链接提供完整的网格。注意：[即，三个交叉点和至少一个传出/管理接口]，可以指定接口的IP地址：

```
resource r0 {
  ...

  # store1 has crossover links like 10.99.1x.y
  connection {
    host store1  address 10.99.12.1 port 7012;
    host store2  address 10.99.12.2 port 7021;
  }
  connection {
    host store1  address 10.99.13.1  port 7013;
    host store3  address 10.99.13.3  port 7031;
  }
  connection {
    host store1  address 10.99.14.1  port 7014;
    host store4  address 10.99.14.4  port 7041;
  }

  # store2 has crossover links like 10.99.2x.y
  connection {
    host store2  address 10.99.23.2  port 7023;
    host store3  address 10.99.23.3  port 7032;
  }
  connection {
    host store2  address 10.99.24.2  port 7024;
    host store4  address 10.99.24.4  port 7042;
  }

  # store3 has crossover links like 10.99.3x.y
  connection {
    host store3  address 10.99.34.3  port 7034;
    host store4  address 10.99.34.4  port 7043;
  }
}
```

请注意用于IP地址和端口的编号方案。另一个资源可以使用相同的IP地址，但端口需要命名为 71*xy* 和下一个端口 72*xy* , 等等。

4.2. 检查DRBD状态

4.2.1. 使用drbdmon检索状态

One convenient way to look at DRBD’s status is the drbdmon utility. It updates the state of DRBD resources in real time.

4.2.2. 通过drbdtop检索状态并与DRBD交互

顾名思义，drbdtop 与 htop 等工具有相似之处。一方面，它允许监控DRBD资源以及交互（例如，将它们切换到 *Primary* ，甚至解决裂脑问题）。完整的概述可以在这里找到[https://linbit.github.io/drbdtop/]。

4.2.3. /proc/drbd中的状态信息

/proc/drbd 已弃用。虽然它不会在8.4系列中删除，但我们建议切换到其他方式，如通过 drbdadm 获取状态信息；或者，为了更方便地监视:One-shot or real time monitoring via drbdsetup events2。

/proc/drbd 是一个虚拟文件，显示有关drbd模块的基本信息。在DRBD 8.4之前，它被广泛使用，但无法跟上DRBD 9提供的信息量。

```
$ cat /proc/drbd
version: 9.0.0 (api:1/proto:86-110) FIXME
GIT-hash: XXX build by linbit@buildsystem.linbit, 2011-10-12 09:07:35
```

第一行以 version: 为前缀，显示系统上使用的DRBD版本。第二行包含有关此特定生成的信息。

4.2.4. 通过 drbdadm 获取状态信息

在其最简单的调用中，我们只需请求单个资源的状态。

```
# drbdadm status home
home role:Secondary
 disk:UpToDate
 nina role:Secondary
  disk:UpToDate
 nino role:Secondary
  disk:UpToDate
 nono connection:Connecting
```

这里表明资源_home_是本地的，在 nina 上以及在 nino 上，是 *UpToDate* 的和 *Secondary* 的，所以这三个节点在他们的存储设备上有相同的数据，并且目前没有人在使用这个设备。

节点 nono 是未连接的，其状态报告为 Connecting_；有关详细信息，请参见下面的连接状态。

您可以通过将 --verbose 和/或 --statistics 参数传递给 drbdsetup 来获得更多信息（为了可读性，行输出将被打断）：

```
# drbdsetup status home --verbose --statistics
home node-id:1 role:Secondary suspended:no
  write-ordering:none
 volume:0 minor:0 disk:UpToDate
    size:1048412 read:0 written:1048412 al-writes:0 bm-writes:48 upper-pending:0
                          lower-pending:0 al-suspended:no blocked:no
 nina local:ipv4:10.9.9.111:7001 peer:ipv4:10.9.9.103:7010 node-id:0
                              connection:Connected role:Secondary
   congested:no
  volume:0 replication:Connected disk:UpToDate resync-suspended:no
     received:1048412 sent:0 out-of-sync:0 pending:0 unacked:0
 nino local:ipv4:10.9.9.111:7021 peer:ipv4:10.9.9.129:7012 node-id:2
                              connection:Connected role:Secondary
   congested:no
  volume:0 replication:Connected disk:UpToDate resync-suspended:no
     received:0 sent:0 out-of-sync:0 pending:0 unacked:0
 nono local:ipv4:10.9.9.111:7013 peer:ipv4:10.9.9.138:7031 node-id:3
                                 connection:Connecting
```

此示例中的每几行组成一个块，对于此资源中使用的每个节点重复此块，对于本地节点则有小的格式例外 - 有关详细信息，请参见下面的内容。

每个块中的第一行显示 node id（对于当前资源；主机在不同资源中可以有不同的 node id ）。此外，还显示了 role （请参见资源角色）。

下一个重要的行以 volume 规范开始；这些规范通常以零开始编号，但配置也可能指定其他id。此行显示 replication 项中的 连接状态。（有关详细信息，请参见连接状态）和 disk 磁盘中的远程 磁盘状态（请参见磁盘状态）。还有一行是这个卷的，它提供了一些统计信息 - 数据 已接收 、已发送 、未同步 等；有关详细信息，请参见性能指标和连接信息数据。

对于本地节点，在我们的示例中，第一行显示资源名 home 。由于第一个块始终描述本地节点，因此没有连接或地址信息。

有关详细信息，请参阅 drbd.conf 手册页。

本例中的其他四行组成一个块，对每个配置的DRBD设备重复该块，前缀为设备次要编号。在本例中，这是 0 ，对应于设备 /dev/drbd0 。

特定于资源的输出包含有关资源的各种信息：

4.2.5. One-shot or real time monitoring via drbdsetup events2

这仅适用于8.9.3及更高版本的用户空间。

这是一种从DRBD中获取信息的底层机制，适用于自动化工具，如监视。

在最简单的调用中，仅显示当前状态，输出如下所示（但是，在终端上运行时，将包括颜色）：

```
# drbdsetup events2 --now r0
exists resource name:r0 role:Secondary suspended:no
exists connection name:r0 peer-node-id:1 conn-name:remote-host connection:Connected
role:Secondary
exists device name:r0 volume:0 minor:7 disk:UpToDate
exists device name:r0 volume:1 minor:8 disk:UpToDate
exists peer-device name:r0 peer-node-id:1 conn-name:remote-host volume:0
  replication:Established peer-disk:UpToDate resync-suspended:no
exists peer-device name:r0 peer-node-id:1 conn-name:remote-host volume:1
  replication:Established peer-disk:UpToDate resync-suspended:no
exists -
```

如果没有 ''--now'' ，进程将继续运行，并发送连续更新，如下所示：

```
# drbdsetup events2 r0
...
change connection name:r0 peer-node-id:1 conn-name:remote-host connection:StandAlone
change connection name:r0 peer-node-id:1 conn-name:remote-host
connection:Unconnected
change connection name:r0 peer-node-id:1 conn-name:remote-host connection:Connecting
```

然后，出于监控的目的，还有另一个参数 ''--statistics''，它将生成一些性能计数器和其他指标：

drbdsetup **verbose output**（为可读性而换行输出）：

```
# drbdsetup events2 --statistics --now r0
exists resource name:r0 role:Secondary suspended:no write-ordering:drain
exists connection name:r0 peer-node-id:1 conn-name:remote-host connection:Connected
                                    role:Secondary congested:no
exists device name:r0 volume:0 minor:7 disk:UpToDate size:6291228 read:6397188
          written:131844 al-writes:34 bm-writes:0 upper-pending:0 lower-pending:0
                                    al-suspended:no blocked:no
exists device name:r0 volume:1 minor:8 disk:UpToDate size:104854364 read:5910680
         written:6634548 al-writes:417 bm-writes:0 upper-pending:0 lower-pending:0
                                    al-suspended:no blocked:no
exists peer-device name:r0 peer-node-id:1 conn-name:remote-host volume:0
         replication:Established peer-disk:UpToDate resync-suspended:no received:0
                                sent:131844 out-of-sync:0 pending:0 unacked:0
exists peer-device name:r0 peer-node-id:1 conn-name:remote-host volume:1
         replication:Established peer-disk:UpToDate resync-suspended:no received:0
                                sent:6634548 out-of-sync:0 pending:0 unacked:0
exists -
```

您可能还喜欢 ''--timestamp'' 参数。

4.2.6. 连接状态

可以通过调用 drbdadm cstate 命令来观察资源的连接状态：

```
# drbdadm cstate <resource>
Connected
Connected
StandAlone
```

如果只对资源的单个连接感兴趣，请指定连接名称：

默认值是配置文件中给定的节点主机名。

```
# drbdadm cstate <peer>:<resource>
Connected
```

资源可能具有以下连接状态之一：

StandAlone
没有可用的网络配置。资源尚未连接，或者已被管理性断开（使用 drbdadm disconnect ），或者由于身份验证失败或脑裂而断开其连接。

Disconnecting
断开连接期间的临时状态。下一个状态是 *StandAlone*。

Unconnected
临时状态，在尝试连接之前。可能的下一个状态： *Connecting*。

Timeout
与对等方通信超时后的临时状态。下一个状态：*Unconnected*。

BrokenPipe

与对等方的连接丢失后的临时状态。下一个状态： *Unconnected* 。

NetworkFailure
与伙伴的连接丢失后的临时状态。下一个状态： *Unconnected*。

ProtocolError
与伙伴的连接丢失后的临时状态。下一个状态：*Unconnected*。

TearDown
临时状态。对等方正在关闭连接。下一个状态： *Unconnected*。

Connecting
此节点正在等待，直到对等节点在网络上变为可见。

Connected
已建立DRBD连接，数据镜像现在处于活动状态。这是正常状态。

4.2.7. 复制状态

每个卷在每个连接上都有一个复制状态。可能的复制状态是：

Off
由于连接未连接，因此卷未通过此连接进行复制。

Established
对该卷的所有写入都将在线复制。这是正常状态。

StartingSyncS
由管理员启动的完全同步正在启动。下一个可能的状态是： *SyncSource* 或 *PausedSyncS* 。

StartingSyncT
由管理员启动的完全同步正在启动。下一个状态：*WFSyncUUID*。

WFBitMapS
部分同步刚刚开始。下一个可能的状态：*SyncSource_或_PausedSyncS*。

WFBitMapT
部分同步刚刚开始。下一个可能的状态：*WFSyncUUID*。

WFSyncUUID
同步即将开始。下一个可能的状态：*SyncTarget_或_PausedSyncT*。

SyncSource
同步当前正在运行，本地节点是同步源。

SyncTarget
同步当前正在运行，本地节点是同步的目标。

PausedSyncS
本地节点是正在进行的同步的源，但同步当前已暂停。这可能是由于依赖于另一个同步进程的完成，或者是由于同步已被 drbdadm pause sync 手动中断。

PausedSyncT
本地节点是正在进行的同步的目标，但同步当前已暂停。这可能是由于依赖于另一个同步进程的完成，或者是由于同步已被 drbdadm pause sync 手动中断。

VerifyS
联机设备验证当前正在运行，本地节点是验证源。

VerifyT
联机设备验证当前正在运行，本地节点是验证的目标。

Ahead
Data replication was suspended, since the link can not cope with the load. This state is enabled by the configuration on-congestion option (see 配置拥塞策略和挂起复制).

Behind
数据复制被对等方挂起，因为链接无法处理负载。此状态由对等节点上的配置 on-congestion 选项启用（请参见配置拥塞策略和挂起复制）。

4.2.8. 资源角色

可以通过调用 drbdadm role 命令来观察资源的角色：

```
# drbdadm role <resource>
Primary
```

您可以看到以下资源角色之一：

Primary
资源当前处于主角色中，可以读取和写入。此角色仅在两个节点中的一个节点上发生，除非启用了dual-primary mode。

Secondary
资源当前处于辅助角色中。它通常从其对等方接收更新（除非在断开连接模式下运行），但既不能读取也不能写入。此角色可能出现在一个或两个节点上。

Unknown
资源的角色当前未知。本地资源角色从未具有此状态。它仅为对等方的资源角色显示，并且仅在断开连接模式下显示。

4.2.9. 磁盘状态

可以通过调用 drbdadm dstate 命令来观察资源的磁盘状态：

```
# drbdadm dstate <resource>
UpToDate
```

磁盘状态可以是以下之一：

Diskless
没有为DRBD驱动程序分配本地块设备。这可能意味着资源从未连接到其备份设备，它已使用 drbdadm detach 手动分离，或者在发生较低级别的I/O错误后自动分离。

Attaching
读取元数据时的临时状态。

Detaching
在分离并等待正在进行的IOs完成时的临时状态。

Failed
本地块设备报告I/O失败后的瞬态。下一个状态：*Diskless*。

Negotiating

在 已经*Connected* 的DRBD设备上执行附加操作时的瞬态。

Inconsistent
数据不一致。在两个节点上（在初始完全同步之前）创建新资源时立即出现此状态。此外，在同步期间，在一个节点（同步目标）中可以找到此状态。

Outdated
资源数据一致，但outdated。

DUnknown
如果没有可用的网络连接，则此状态用于对等磁盘。

Consistent
没有连接的节点的一致数据。建立连接后，决定数据是 *UpToDate* 还是 *Outdated*。

UpToDate
数据的一致、最新状态。这是正常状态。

4.2.10. 连接信息数据

local
显示网络协议栈、用于接受来自对等方的连接的本地地址和端口。

peer
显示网络协议栈、对等方节点地址和用于连接的端口。

congested
此标志指示数据连接的TCP发送缓冲区是否已填充80%以上。

4.2.11. 性能指标

The command drbdsetup status --verbose --statistics can be used to show performance statistics. These are also available in drbdsetup events2 --statistics, although there will not be a *changed* event for every change. The statistics include the following counters and gauges:

Per volume/device:

read (disk read)
Net data read from local disk; in KiB.

written (disk written)
Net data written to local disk; in KiB.

al-writes (activity log)
元数据活动日志区域的更新次数。

bm-writes (bitmap)
元数据位图区域的更新次数。

upper-pending (application pending)
Number of block I/O requests forwarded to DRBD, but not yet answered (completed) by DRBD.

lower-pending (local count)
DRBD向本地I/O子系统发出的打开请求数。

blocked
显示本地I/O拥塞。

- *no*: No congestion.
- *上面的* I/O设备被阻塞，即文件系统。典型的原因是
  - 由管理员暂停I/O，请参阅 drbdadm 中的 suspend-io 命令。
  - transient blocks, e.g. during attach/detach
  - 缓冲区耗尽，请参见优化DRBD性能
  - Waiting for bitmap IO
- *lower*: Backing device is congested.
- *upper*,*lower*: Both *upper* and *lower* are blocked.

Per connection:

ap-in-flight (application in-flight)
Application data that is being written by the peer. That is, DRBD has sent it to the peer and is waiting for the acknowledgement that it has been written. In sectors (512 bytes).

rs-in-flight (resync in-flight)
Resync data that is being written by the peer. That is, DRBD is *SyncSource*, has sent data to the peer as part of a resync and is waiting for the acknowledgement that it has been written. In sectors (512 bytes).

Per connection and volume ("peer device"):

received (network receive)
Net data received from the peer; in KiB.

sent (network send)
Net data sent to the peer; in KiB.

out-of-sync
Amount of data currently out of sync with this peer, according to the bitmap that DRBD has for it; in KiB.

pending
Number of requests sent to the peer, but that have not yet been acknowledged by the peer.

unacked (unacknowledged)
Number of requests received from the peer, but that have not yet been acknowledged by DRBD on this node.

resync-suspended
Whether the resynchronization is currently suspended or not. Possible values are *no*, *user*, *peer*, *dependency*. Comma separated.

4.3. 启用和禁用资源

4.3.1. 使能资源

通常，所有配置的DRBD资源都会自动启用

- 由群集资源管理应用程序根据您的群集配置自行决定，或
- 在系统启动时通过 /etc/init.d/drbd init脚本。

但是，如果出于任何原因需要手动启用资源，可以通过调用命令

```
# drbdadm up <resource>
```

与往常一样，如果要同时启用在 /etc/drbd.conf 中配置的所有资源，可以使用关键字 all 而不是特定的资源名。

4.3.2. 禁用资源

您可以通过调用命令暂时禁用特定资源

```
# drbdadm down <resource>
```

在这里，如果您希望一次临时禁用 /etc/drbd.conf 中列出的所有资源，也可以使用关键字 all 代替资源名。

4.4. 重新配置资源

DRBD允许您在资源运行时重新配置它们。包括，

- 对 /etc/drbd.conf 中的资源配置需要进行任何必要的更改，
- 在两个节点之间同步 /etc/drbd.conf 文件，
- 在两个节点上发出 drbdadm adjust<resource> 命令。

drbdadm adjust 然后切换到 drbdsetup 对配置进行必要的调整。与往常一样，您可以通过使用 -d （dry-run）选项运行 drbdadm 来查看挂起的 drbdsetup 调用。

对 /etc/drbd.conf 中的 common 部分进行更改时，可以通过发出 drbdadm adjust all 来调整一次运行中所有资源的配置。

4.5. 提升和降级资源

使用以下命令之一手动将aresource’ s role从次要角色切换到主要角色（提升）或反之亦然（降级）：

```
# drbdadm primary <resource>
# drbdadm secondary <resource>
```

在single-primary mode（DRBD的默认设置）中，任何资源在任何给定时间只能在一个节点上处于主要角色，而connection state是连接的。因此，在一个节点上发出 drbdadm primary<resource> ，而该指定的资源是在另一个节点上的primary角色, 将导致错误。

配置为允许dual-primary mode的资源可以切换到两个节点上的主要角色；这是虚拟机在线迁移所必需的。

4.6. 基本手动故障切换

如果不使用Pacemaker并希望在被动/主动配置中手动处理故障转移，则过程如下。

在当前主节点上，停止使用DRBD设备的任何应用程序或服务，卸载DRBD设备，并将资源降级为次要资源。

```
# umount /dev/drbd/by-res/<resource>/<vol-nr>
# drbdadm secondary <resource>
```

现在登录到想提升为primary的节点上，升级资源并装载设备。

```
# drbdadm primary <resource>
# mount /dev/drbd/by-res/<resource>/<vol-nr> <mountpoint>
```

If you’re using the auto-promote feature, you don’t need to change the roles (*Primary*/*Secondary*) manually; only stopping of the services and unmounting, respectively mounting, is necessary.

4.7. 升级DRBD

升级DRBD是一个相当简单的过程。本节将详细介绍从8.4.x升级到9.0.x的过程；对于9版本内升级，它变得更加容易，请参见下面的简短版本从DRBD 9到DRBD 9。

4.7.1. 概述

将8.4升级到9.0的一般过程如下：

- 配置新仓库（如果使用来自LINBIT的包）
- 确保当前情况检查良好
- Pause任何群集管理器
- 安装新版本
- 如果要移动到2个以上的节点，则需要调整较低级别的存储空间，以便为其他元数据提供空间；本主题将在LVM Chapter中讨论。
- 取消配置资源，卸载DRBD 8.4，然后load the v9 kernel module
- Convert DRBD meta-data格式为 v09 ，可能在同一步骤中需要更改位图的数量
- 启动新版本DRBDresources up

4.7.2. 更新仓库

由于8.4和9.0分支之间的更改众多，我们为每个分支创建了单独的仓库。在两台服务器上执行此仓库更新。

RHEL/CentOS系统

编辑 /etc/yum.repos.d/linbit.repo 文件。

```
[drbd-9.0]
name=DRBD 9.0
baseurl=http://packages.linbit.com/<hash>/yum/rhel7/drbd-9.0/<arch>
gpgcheck=0
```

必须填充<hash>和<arch>变量。<hash>由LINBIT支持服务提供。

Debian/Ubuntu系统

编辑 /etc/apt/sources.list （或 /etc/apt/sources.d/ 中的文件）加入以下更改。

```
deb http://packages.linbit.com/<hash>/ stretch drbd-9.0
```

如果您不使用 stretch 版本，而是使用其他版本，则需要更改到对应的版本名称。

您必须填充 <hash> 变量。<hash> 由LINBIT支持服务提供。

接下来，您需要将DRBD签名密钥添加到您的受信任密钥中。

```
# gpg --keyserver subkeys.pgp.net --recv-keys  0x282B6E23
# gpg --export -a 282B6E23 | apt-key add -
```

最后执行 apt update ，以便Debian识别更新的存储库。

```
# apt update
```

4.7.3. 检查DRBD状态

在开始之前，请确保资源同步。cat/proc/drbd 的输出（仅在9.0之前可用）应显示 *UpToDate/UpToDate* 。

```
bob# cat /proc/drbd

version: 8.4.9-1 (api:1/proto:86-101)
GIT-hash: e081fb0570183db40caa29b26cb8ee907e9a7db3 build by linbit@buildsystem, 2016-
11-18 14:49:21

 0: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r-----
    ns:0 nr:211852 dw:211852 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:d oos:0
```

4.7.4. 暂停群集

既然您知道资源是同步的，就从升级辅助节点开始。这可以手动完成，或者如果您正在使用Pacemaker，请将节点置于待机模式。下面将介绍这两个过程。如果你正在运行Pacemaker，不要使用手动方法。

- 手动方法

```
bob# /etc/init.d/drbd stop
```

- Pacemaker

将辅助节点置于待机模式。在本例中，bob 是辅助节点。

```
bob# crm node standby bob
```

您可以使用 crm-mon-rf 或 cat/proc/drbd 监视群集的状态，直到它显示您的资源为 未配置 。

4.7.5. 升级包

现在用yum或apt更新包。

```
bob# yum upgrade
```

```
bob# apt upgrade
```

升级完成后，将在次节点 bob 上安装最新的DRBD 9.0内核模块和DRBD实用程序。

但是内核模块还没有激活。

4.7.6. 加载新的内核模块

现在旧的DRBD内核模块不应该再使用了，所以我们通过

```
bob# rmmod drbd
```

如果出现类似 ERROR:Module drbd is in use （错误：模块drbd正在使用）的消息，则并非所有资源都已正确停止。+ 重试升级DRBD，和/或运行命令 drbdadm down all 以确定哪些资源仍处于活动状态。

可能阻止卸载的典型问题有：

- 在DRBD支持的文件系统上有导出NFS的操作（参见 exportfs -v 输出）
- 文件系统仍在安装-检查 grep drbd/proc/mounts
- Loopback 设备仍然处于活动状态（ losetup -l ）
- 直接或间接使用DRBD的device mapper（ dmsetup ls --tree ）
- 有带DRBD-PV的LVM（pvs）

请注意，这个列表并不完整 - 这些只是最常见的例子。

现在我们可以加载新的DRBD内核模块：

```
bob# modprobe drbd
```

现在您应该检查 /proc/drbd 的内容，并验证是否加载了正确的（新的）版本；如果安装的软件包的内核版本不正确，modprobe 也会成功，但旧版本将再次处于活动状态。

cat /proc/drbd 的输出现在应该显示为9.0.x，看起来与此类似。

```
version: 9.0.0 (api:2/proto:86-110)
GIT-hash: 768965a7f158d966bd3bd4ff1014af7b3d9ff10c build by root@bob, 2015-09-03
13:58:02
Transports (api:10): tcp (1.0.0)
```

在主节点，alice上, cat /proc/drbd 将仍然显示先前的版本，直到您升级它。

4.7.7. 迁移配置文件

DRBD 9.0与8.4配置文件向后兼容；但是，某些语法已更改。有关更改的完整列表，请参见配置语法的更改。同时，您可以使用`drbdadm dump all` 命令轻松移植旧配置。这将输出一个新的全局配置和一个新的资源配置文件。获取此输出并相应地进行更改。

4.7.8. 更改元数据

现在您需要将磁盘上的元数据转换为新版本；这非常简单，只需运行一个命令并确认两个问题。

如果要更改节点数，您应该已经增加了底层设备的大小，以便有足够的空间存储增加的位图；在这种情况下，您将使用附加参数 --max peers=<*N*> 。当需要确定(可能的)节点个数时，请参考 <<s-drbd-client> 做相应的配置。

升级DRBD元数据只需运行一个命令并确认两个问题：

```
# drbdadm create-md <resource>
You want me to create a v09 style flexible-size internal meta data block.
There appears to be a v08 flexible-size internal meta data block
already in place on <disk> at byte offset <offset>

Valid v08 meta-data found, convert to v09?
[need to type 'yes' to confirm] yes

md_offset <offsets...>
al_offset <offsets...>
bm_offset <offsets...>

Found some data

 ==> This might destroy existing data! <==

Do you want to proceed?
[need to type 'yes' to confirm] yes

Writing meta data...
New drbd meta data block successfully created.
success
```

Of course, you can pass all for the resource names, too; and if you feel really lucky, you can avoid the questions via a command line like this here, too. (Yes, the order is important.)

```
drbdadm -v --max-peers=<N>  -- --force create-md <resources>
```

4.7.9. 重新启动DRBD

现在，唯一要做的就是让DRBD设备重新启动并再次运行- 一个简单的命令 drbdadm up all 应该能做到这一点。

现在，根据您是否有集群管理器或手动跟踪资源，又有两种不同的方法。

- 手动

  ```
  bob# /etc/init.d/drbd start
  ```

- Pacemaker

  ```
  # crm node online bob
  ```

  这将使DRBD连接到另一个节点，并且重新同步过程将启动。

当所有资源上的两个节点都是 *UpToDate* 时，您可以将应用程序移动到已升级的节点（此处为 bob ），然后在仍在运行8.4的群集节点上执行相同的步骤。

4.7.10. 从DRBD 9到DRBD 9

如果您已经在运行9.0，则只需install new package versions，使群集节点*standby*，unload/reload内核模块，start the resources，并使集群节点重新联机 [3].

上面已经详细介绍了这些步骤，因此我们在这里不再重复。

4.8. 启用双主模式

双主模式允许资源在多个节点上同时承担主角色。这样做可以是永久性的，也可以是暂时性的。

双主模式要求将资源配置为同步复制（protocol C）。因此，它对延迟敏感，不适合广域网环境。

另外，由于这两种资源都是主要的，节点之间网络的任何中断都会导致脑裂。

在DRBD 9.0.x中，双主模式仅限于2个主节点，通常用于实时迁移。

4.8.1. 永久双主模式

要启用双主模式，请在资源配置的 net 部分将 allow two primaries 选项设置为 yes：

```
resource <resource>
 net {
  protocol C;
  allow-two-primaries yes;
  fencing resource-and-stonith;
 }
 handlers {
  fence-peer "...";
  unfence-peer "...";
 }
 ...
}
```

之后，不要忘记同步节点之间的配置。在两个节点上都运行 drbdadm adjust 。

现在可以使用 drbdadm primary 将两个节点同时更改为role primary。

您应该始终执行适当的围栏策略。使用 allow-two-primaries 而没有围栏策略是个坏主意，比在无围栏使用单主节点更糟糕。

4.8.2. 临时双主模式

要临时为通常在单个主配置中运行的资源启用双主模式，请使用以下命令：

```
# drbdadm net-options --protocol=C --allow-two-primaries <resource>
```

要结束临时双主模式，请运行与上面相同的命令，但使用 --allow two primaries=no （以及所需的复制协议，如果适用）。

4.9. 使用在线设备验证

4.9.1. 启用在线验证

默认情况下，资源的On-line device verification未启用。要启用它，请将以下行添加到 /etc/drbd.conf 中的资源配置中：

```
resource <resource>
  net {
    verify-alg <algorithm>;
  }
  ...
}
```

<algorithm> 可能是系统内核配置中内核加密API支持的任何消息摘要算法。通常，您至少可以从 sha1 ，md5 和 crc32c 中进行选择。

如果对现有资源进行此更改，请一如既往地将 drbd.conf 同步到对等节点，并在两个节点上运行 drbdadm adjust *<resource>* 。

4.9.2. 调用联机验证

启用联机验证后，可以使用以下命令启动验证运行：

```
# drbdadm verify <resource>
```

输入上述命令后，DRBD将针对 *<resource>* 启动联机验证运行，如果它检测到任何不同步的块，则会将这些块标记为此类并向内核日志中写入一条消息。所有使用该设备的应用程序那时可以继续不受阻碍地执行此操作， 并且您也根据[s-switch-resource-roles，switch resource roles]进行定制化。

如果在验证运行期间检测到不同步块，则可以在验证完成后使用以下命令重新同步它们：

```
# drbdadm disconnect <resource>
# drbdadm connect <resource>
```

4.9.3. 自动在线验证

大多数用户都希望自动化联机设备验证。这很容易实现。在任一节点上创建一个包含以下内容的文件，名为 /etc/cron.d/drbd-verify :

```
42 0 * * 0    root    /sbin/drbdadm verify <resource>
```

这将使 cron 在每周日午夜后42分钟调用一个设备验证；因此，假设您在周一上午进入办公室，快速查看资源的状态将显示结果。如果您的设备很大，而且32小时还不够，那么您将注意到 *VerifyS* 或 *VerifyT* 作为连接状态，这意味着 verify 仍在进行中。

如果您已经为所有资源启用了联机验证（例如，通过在 /etc/drbd.d/global_common.conf 中的 common 部分添加 verify-alg *<algorithm>*，您就可以使用：

```
42 0 * * 0    root    /sbin/drbdadm verify all
```

4.10. 配置同步速率

通常，会尝试确保后台同步（这会使同步目标上的数据暂时不一致）尽快完成。但是，还必须防止后台同步占用前台复制所需的所有带宽，这将对应用程序性能造成损害。因此，您必须配置同步带宽以匹配您的硬件 - 您可以永久地或动态地这样做。

设置高于辅助节点上最大写入吞吐量的同步速率是没有意义的。您不能期望辅助节点奇迹般地能够比其I/O子系统所允许的写入速度快，因为它恰好是正在进行的设备同步的目标。

同样，出于同样的原因，设置高于复制网络上可用带宽的同步速率是没有意义的。

4.10.1. 估计同步速度

对于这个值，一个很好的经验法则是使用大约30%的可用复制带宽。因此，如果有一个I/O子系统能够维持400MB/s的写吞吐量，而一个千兆以太网网络能够维持110MB/s的网络吞吐量（网络是瓶颈），您可以如下计算：

$$110\ MB/s * 0.3 = 33\ MB/s$$

插图 6. 同步速率示例，110MB/s有效可用带宽

因此，rate 选项的建议值为 33M 。

相比之下，如果您有一个最大吞吐量为80MB/s的I/O子系统和一个千兆以太网连接（I/O子系统是瓶颈），您将计算：

$$80\ MB/s * 0.3 = 24\ MB/s$$

插图 7. 同步速率示例，80MB/s有效可用带宽

在这种情况下，rate 选项的建议值为 24M 。

类似地，对于800MB/s的存储速度和10Gbe的网络连接，您将获得大约\~240MB/s的同步速率。

4.10.2. 可变同步速率配置

当多个DRBD资源共享一个复制/同步网络时，以固定速率同步可能不是最佳方法。因此，在DRBD 8.4.0中，默认情况下启用了可变速率同步。在这种模式下，DRBD使用自动控制环路算法来确定和调整同步速率。该算法保证了前台复制始终有足够的带宽，大大减轻了后台同步对前台I/O的影响。

可变速率同步的最佳配置可能因可用的网络带宽、应用程序I/O模式和链路拥塞而有很大差异。理想的配置设置还取决于DRBD Proxy是否正在使用。为了优化配置这个DRBD特性，最好聘请专业顾问。下面提供了一个配置示例（假设与DRBD代理一起部署）：

```
resource <resource> {
 disk {
  c-plan-ahead 5;
  c-max-rate 10M;
  c-fill-target 2M;
 }
}
```

c-fill-target 的一个很好的初始值是 *BDP * 2*, 其中 *BDP* 是复制链接上的带宽延迟产品。

例如，当使用1GBit/s交叉连接时，您将得到大约200微秒的延迟 [4]。
1GBit/s意味着大约120MB/s；乘以200*10^{-6}秒得到24000字节。把这个值四舍五入到下一个MB，你就得到了值。

另一个例子：一个100兆位的广域网连接，200毫秒的延迟意味着12兆字节/秒乘以0.2秒，或者说大约2.5兆字节 "on the wire""在线"。在这里， c-fill-target 的初始值可预设为是3MB。

有关其他配置项的详细信息，请参见 drbd.conf 手册页。

4.10.3. 永久固定同步速率配置

在一些非常受限的情况下，[5]，使用一些固定的同步速率可能是有意义的。在这种情况下，首先需要使用 c-plan-ahead 0； 关闭动态同步速率控制器。

然后，资源用于后台重新同步的最大带宽由资源的 resync-rate 选项确定。这必须包含在 /etc/drbd.conf 的资源定义的 disk 条目中：

```
resource <resource>
 disk {
  resync-rate 40M;
  ...
 }
 ...
}
```

请注意，速率设置以字节为单位，而不是以比特/秒为单位；默认单位是字节，因此值 4096 将被解释为 4MiB 。

这只是定义了DRBD试图达到的速率。如果存在吞吐量较低的瓶颈（网络、存储速度），则无法达到定义的速度（也称为"期望"性能；）。

4.10.4. 关于同步的更多提示

当一些要同步的数据不再实际使用时（例如，因为文件在一个节点未连接时被删除），您可能会从Trim/Discard支持中受益。

此外， c-min-rate 很容易被误解 - 它没有定义*最小*同步速度，而是一个限制，低于这个限制，DRBD就不会进一步 **故意** 减速。
您是否能够达到同步速率取决于您的网络和存储速度、网络延迟（对于共享链接来说，这可能是高度可变的）和应用程序IO（对此可能无能为力）。

4.11. 配置基于校验和的同步

Checksum-based synchronization默认情况下不为资源启用。要启用它，请将以下行添加到 /etc/drbd.conf 中的资源配置中：

```
resource <resource>
 net {
  csums-alg <algorithm>;
 }
 ...
}
```

<algorithm> 可能是系统内核配置中内核加密API支持的任何消息摘要算法。通常，您至少可以从 sha1 ， md5 和 crc32c 中进行选择。

如果对现有资源进行此更改，请一如既往地将 drbd.conf 同步到对等节点，并在两个节点上运行 drbdadm adjust *<resource>* 。

4.12. 配置拥塞策略和挂起复制

在复制带宽高度可变的环境中（这在广域网复制设置中是典型的），复制链路有时可能会变得拥挤。在默认配置中，这将导致主节点上的I/O阻塞，这有时是不可取的。

相反，在这种情况下，您可以将DRBD配置为挂起正在进行的复制，从而使主数据集位于次数据集的 *前拉 pull ahead* 位置。在这种模式下，DRBD保持复制通道打开 - 它从不切换到断开连接的模式 - 但直到有足够的带宽再次可用时才真正进行复制。

以下示例适用于DRBD代理配置：

```
resource <resource> {
  net {
    on-congestion pull-ahead;
    congestion-fill 2G;
    congestion-extents 2000;
    ...
  }
  ...
}
```

通常明智的做法是将 congestion-fill 和 congestion-extents 与 pull-ahead 选项一起设置。

congestion-fill 的理想值是90%

- 当通过DRBD proxy进行复制时，分配的DRBD proxy缓冲存储器，或
- 在非DRBD代理设置中的TCP网络发送缓冲区。

congestion-extents 的理想值是受影响资源配置的 al-extents 的90%。

4.13. 配置I/O错误处理策略

DRBD的strategy for handling lower-level I/O errors由 /etc/drbd.conf 文件中resource下的 disk 配置中的 on-io-error 选项确定:

```
resource <resource> {
  disk {
    on-io-error <strategy>;
    ...
  }
  ...
}
```

当然，如果要为所有资源定义全局I/O错误处理策略，也可以在 common 部分中设置此值。

<strategy> 可能是以下选项之一：

detach
这是默认和推荐的选项。在发生较低级别的I/O错误时，节点将丢弃其备份设备，并继续以无盘模式运行。

pass-on
这导致DRBD向上层报告I/O错误。在主节点上，它将报告给已装入的文件系统。在次节点上，它被忽略（因为次节点没有要报告的上层）。

call-local-io-error

调用定义为本地I/O错误处理程序的命令。这要求在资源的 handlers 部分中定义相应的 local-io-error 命令调用。完全由管理员自行决定使用 local-io-error 调用的命令（或脚本）来实现I/O错误处理。

早期的DRBD版本（8.0之前）包含另一个选项 panic ，每当发生本地I/O错误时，该选项将通过内核panic从集群中强制删除节点。虽然该选项不再可用，但可以通过 local-io-error / call-local-io-error 接口来模拟相同的行为。只有当你完全理解这种行为的含义时，你才应该这样做。

您可以按照此过程重新配置正在运行的资源的I/O错误处理策略：

- 在 /etc/drbd.d/*<resource>*.res 中编辑资源配置。
- 将配置复制到对等节点。
- 在两个节点上都运行 drbdadm adjust 命令。

4.14. 配置复制通信完整性检查

Replication traffic integrity checking默认情况下不为资源启用。要启用它，请将以下行添加到 /etc/drbd.conf 中的资源配置中：

```
resource <resource>
 net {
  data-integrity-alg <algorithm>;
 }
 ...
}
```

<algorithm> 可能是系统内核配置中内核加密API支持的任何消息摘要算法。通常，您至少可以从 sha1 ， md5 和 crc32c 中进行选择。

如果对现有资源进行此更改，请一如既往地将 drbd.conf 同步到对等节点，并在两个节点上运行 drbdadm adjust *<resource>* 。

此功能不用于生产用途。仅当需要诊断数据损坏问题并希望查看传输路径（网络硬件、驱动程序、交换机）是否有故障时才启用！

4.15. 调整资源大小

When growing (extending) DRBD volumes, you need to grow from bottom to top. You need to extend the backing block devices on all nodes first. Then you can tell DRBD to use the new space.

Once the DRBD volume is extended, you need to still propagate that into whatever is using DRBD: extend the file system, or make a VM running with this volume attached aware of the new "disk size".

That all typically boils down to

```
# # on all nodes, resize the backing LV:
# lvextend -L +${additional_gb}g VG/LV
# # on one node:
# drbdadm resize ${resource_name}/${volume_number}
# # on the Primary only:
# # resize the file system using the file system specific tool, see below
```

See also the next sections 在线扩容 and following.

Note that different file systems have different capabilities and different sets of management tools. For example XFS can only grow. You point its tool to the active mount point: xfs_growfs /where/you/have/it/mounted.

While the EXT family can both grow (even online), and also shrink (only offline; you have to unmount it first). To resize an ext3 or ext4, you would point the tool not to the mount point, but to the (mounted) block device: resize2fs /dev/drbd#

Obviously use the correct DRBD (as displayed by mount or df -T, while mounted), and **not** the backing block device. As long as DRBD is up, that’s not supposed to work anyways (resize2fs: Device or resource busy while trying to open /dev/mapper/VG-LV Couldn’t find valid filesystem superblock.). If you tried to do that offline (with DRBD stopped), you may corrupt DRBD meta data if you ran the file system tools directly against the backing LV or partition. So don’t.

You do the file system resize only **once** on the Primary, against the active DRBD device. DRBD replicates the changes to the file system structure. That is what you have it for.

Also, don’t use resize2fs on XFS volumes, or XFS tools on EXT, or … but the right tool for the file system in use.

resize2fs: Bad magic number in super-block while trying to open /dev/drbd7 is probably just trying to tell you that this is **not** an EXT file system, and you should try an other tool instead. Maybe xfs_growfs? But as mentioned, that does not take the block device, but the mount point as argument.

When shrinking (reducing) DRBD volumes, you need to shrink from top to bottom. So first make sure no one is using the space you want to cut off. Next, shrink the file system (if your file system supports that). Then tell DRBD to stop using that space, which is not so easy with DRBD internal meta data, because they are by design "at the end" of the backing device.

Once you are sure that DRBD won’t use the space anymore either, you can cut it off from the backing device, for example using lvreduce.

See also 在线缩容, 离线缩容.

4.15.1. 在线扩容

如果支持块设备可以在操作（联机）时增长，那么也可以在操作期间基于这些设备增加DRBD设备的大小。为此，必须满足两个标准：

1. 受影响资源的备份设备必须由逻辑卷管理子系统（如LVM）管理。
2. 资源当前必须处于连接状态。

在所有节点上增加了备份块设备后，请确保只有一个节点处于主状态。然后在一个节点上输入：

```
# drbdadm resize <resource>
```

这将触发新增部分的同步。同步是以从主节点到辅助节点的顺序完成的。

如果要添加的空间是干净的，可以使用—assume clean选项跳过同步新增的空间。

```
# drbdadm -- --assume-clean resize <resource>
```

4.15.2. 离线扩容

当两个节点上的备份块设备在DRBD处于非活动状态时增长，并且DRBD资源正在使用external meta data，则自动识别新大小。不需要人为干预。下次在两个节点上激活DRBD并成功建立网络连接后，DRBD设备将具

有新的大小。

然而，如果DRBD资源被配置为使用internal meta data，则在新的扩容空间可用之前，必须将该元数据移动到所生长设备的末端。为此，请完成以下步骤：

这是一个高级功能。请自己斟酌使用。

- 取消配置您的DRBD资源：

```
# drbdadm down <resource>
```

- 在调整大小之前，请将元数据保存在文本文件中：

```
# drbdadm dump-md <resource> > /tmp/metadata
```

必须在两个节点上执行此操作，对每个节点使用单独的转储文件。 **不要** 在一个节点上转储元数据，只需将转储文件复制到对等节点。**这. 行. 不. 通.**

- 在两个节点上扩展备份块设备。
- 在两个节点上相应地调整文件 /tmp/metadata 中的大小信息（ la-size-sect ）。请记住，必须在扇区中指定 la-size-sect。
- 重新初始化元数据区域：

```
# drbdadm create-md <resource>
```

- 在两个节点上重新导入更正的元数据：

```
# drbdmeta_cmd=$(drbdadm -d dump-md <resource>)
# ${drbdmeta_cmd/dump-md/restore-md} /tmp/metadata
Valid meta-data in place, overwrite? [need to type 'yes' to confirm]
yes
Successfully restored meta data
```

此示例使用 bash 参数替换。它可能在其他SHELL中工作，也可能不工作。如果不确定当前使用的是哪个SHELL，请检查您的 SHELL 环境变量。

- 重新启用DRBD资源：

```
# drbdadm up <resource>
```

- 在一个节点上，升级DRBD资源：

```
# drbdadm primary <resource>
```

- 最后，扩展文件系统，使其填充DRBD设备的扩展大小。

4.15.3. 在线缩容

仅外部元数据支持在线缩容。

在缩容DRBD设备之前，*必须* 收缩DRBD上面的层，通常是文件系统。由于DRBD无法询问文件系统实际使用了多少空间，因此必须小心操作，以免造成数据丢失。

_文件系统_是否可以在线缩容取决于所使用的文件系统。大多数文件系统不支持在线缩容。XFS完全不支持缩容。

要在线缩小DRBD，请先缩小位于其上的文件系统, *之后* , 输入以下命令：

```
# drbdadm resize --size=<new-size> <resource>
```

您可以使用常用的单位来表示 *<new-size>* （K、M、G等）。收缩DRBD后，还可以收缩其包含的块设备（如果它支持收缩）。

在调整底层设备的大小后，最好输入 drbdadm resize *<resource>* 命令，以便将DRBD元数据*真正*写入卷末尾的预期空间。

4.15.4. 离线缩容

如果在DRBD处于非活动状态时收缩备份块设备，DRBD将在下次尝试连接时拒绝连接到此块设备，因为它现在太小（如果使用外部元数据），或者找不到它的元数据（如果使用内部元数据）。要解决这些问题，请使用以下步骤（如果无法使用on-line shrinking）：

这是一个高级功能。请自己斟酌使用。

- 在DRBD仍然配置的情况下，从一个节点缩容文件系统。
- 取消配置您的DRBD资源：

```
# drbdadm down <resource>
```

- 在缩容之前将元数据保存在文本文件中：

```
# drbdadm dump-md <resource> > /tmp/metadata
```

必须在两个节点上执行此操作，对每个节点使用单独的转储文件。 **不要** 在一个节点上转储元数据，只需将转储文件复制到对等节点。**这. 行. 不. 通.**

- 缩容两个节点上的备份块设备。
- 在两个节点上相应地调整文件 /tmp/metadata 中的大小信息（ la-size-sect ）。请记住，必须在扇区中指定 la-size-sect。
- **仅当您使用内部元数据时** (此时可能由于收缩过程而丢失），才需要重新初始化元数据区域

```
# drbdadm create-md <resource>
```

- 在两个节点上重新导入更正的元数据：

```
# drbdmeta_cmd=$(drbdadm -d dump-md <resource>)
# ${drbdmeta_cmd/dump-md/restore-md} /tmp/metadata
Valid meta-data in place, overwrite? [need to type 'yes' to confirm]
yes
Successfully restored meta data
```

此示例使用 bash 参数替换。它可能在其他SHELL中工作，也可能不工作。如果不确定当前使用的是哪个SHELL，请检查您的 SHELL 环境变量。

- 重新启用DRBD资源：

```
# drbdadm up <resource>
```

4.16. 禁用备份设备刷新

在使用电池备份写缓存（BBWC）的设备上运行DRBD时，应禁用设备刷新。大多数存储控制器允许在电池耗尽时自动禁用写缓存，在电池耗尽时切换到直写模式。强烈建议启用此功能。

在没有BBWC的情况下运行或在电池耗尽的BBWC上运行时，禁用DRBD的刷新 *可能会导致数据丢失* ，不应尝试。

DRBD允许对复制的数据集和DRBD自己的元数据分别启用和禁用backing device flushes。默认情况下，这两个选项都已启用。如果您希望禁用其中一个（或两个），您可以在DRBD配置文件 /etc/DRBD.conf 的 disk 部分中设置它。

要禁用复制数据集的磁盘刷新，请在配置中包含以下行：

```
resource <resource>
  disk {
    disk-flushes no;
    ...
  }
  ...
}
```

要禁用DRBD元数据上的磁盘刷新，请包括以下行：

```
resource <resource>
  disk {
    md-flushes no;
    ...
  }
  ...
}
```

在修改了资源配置（当然，在节点之间同步了 /etc/drbd.conf ）之后，可以通过在两个节点上输入以下命令来启用这些设置：

```
# drbdadm adjust <resource>
```

如果只有一个服务有BBWC [6]，应将设置移动到主机部分，如下所示：

```
resource <resource> {
 disk {
  ... common settings ...
 }

 on host-1 {
  disk {
   md-flushes no;
  }
  ...
 }
 ...
}
```

4.17. 配置裂脑行为

4.17.1. 裂脑通知

DRBD调用 split brain 处理程序（如果已配置），随时检测到split brain。要配置此处理程序，请将以下项添加到资源配置中：

```
resource <resource>
 handlers {
  split-brain <handler>;
  ...
 }
 ...
}
```

<handler> 可以是系统中存在的任何可执行文件。

DRBD发行版包含一个split brain handler脚本，安装为 /usr/lib/DRBD/notify-split-brain.sh 。它会将通知电子邮件发送到指定的地址。要将处理程序配置为将消息发送到 root@localhost （预期是将通知转发给实际系统管理员的电子邮件地址），请按如下所示配置 split-brain handler :

```
resource <resource>
 handlers {
  split-brain "/usr/lib/drbd/notify-split-brain.sh root";
  ...
 }
 ...
}
```

对正在运行的资源进行此修改（并在节点之间同步配置文件）后，无需进行其他干预即可启用处理程序。DRBD只需在下一次出现split brain时调用新配置的处理程序。

4.17.2. 自动裂脑恢复策略

Configuring DRBD to automatically resolve data divergence situations resulting from split-brain (or other) scenarios is configuring for potential **automatic data loss**. Understand the implications, and don’ t do it if you don’ t mean to.

您应该花更多时间研究围栏策略、仲裁设置、群集管理器集成和冗余群集管理器通信链接，以便在第一时间*避免*数据差异。

为了能够启用和配置DRBD的自动裂脑恢复策略，您必须了解DRBD为此提供了几个配置选项。DRBD根据检测到裂脑时主要角色的节点数应用其裂脑恢复程序。为此，DRBD检查以下关键字，这些关键字都可以在资源的 net 配置部分找到：

after-sb-0pri
裂脑被检测到，但此时资源在任何主机上都不是主要角色。对于这个选项，DRBD理解以下关键字：

- disconnect：不要自动恢复，只需调用 split brain 处理程序脚本（如果已配置），断开连接并以断开模式继续。
- discard-younger-primary：放弃并回滚对最后担任主服务器角色的主机所做的修改。
- discard-least-changes：丢弃并回滚发生较少更改的主机上的更改。
- discard-zero-changes：如果有任何主机根本没有发生任何更改，只需在另一个主机上应用所做的所有修改并继续。

after-sb-1pri
裂脑被检测到，此时资源在一个主机上扮演主要角色。对于这个选项，DRBD理解以下关键字：

- disconnect：与 after-sb-0pri 一样，只需调用 split brain 处理程序脚本（如果已配置），断开连接并以断开模式继续。
- consensus：应用 after-sb-0pri 中指定的相同恢复策略。如果在应用这些策略后可以选择裂脑受害者，则自动解决。否则，行为就像指定了 disconnect 一样。
- call-pri-lost-after-sb：应用 after-sb-0pri 中指定的恢复策略。如果在应用这些策略后可以选择裂脑受害者，请调用受害者节点上的 pri-lost-after-sb 处理程序。必须在 handlers 部分中配置此处理程序，并要求将节点从集群中强制删除。
- discard-secondary：无论哪个主机当前处于第二个角色，使该主机成为裂脑受害者。

after-sb-2pri
裂脑被检测到，此时资源在两个主机上都处于主要角色。此选项接受与 after-sb-1pri 相同的关键字，但 discard-secondary 和 consensus 除外。

DRBD理解这三个选项的附加关键字，这里省略了这些关键字，因为它们很少使用。请参阅 drbd.conf 的手册页，以了解此处未讨论的脑分裂恢复关键字的详细信息。

例如，在双主模式下用作GFS或OCFS2文件系统的块设备的资源的恢复策略定义如下：

```
resource <resource> {
 handlers {
  split-brain "/usr/lib/drbd/notify-split-brain.sh root"
  ...
 }
 net {
  after-sb-0pri discard-zero-changes;
  after-sb-1pri discard-secondary;
  after-sb-2pri disconnect;
  ...
 }
 ...
}
```

4.18. 创建堆叠的三节点设置

三个节点的设置包括一个堆叠在另一个设备上的DRBD设备。

在DRBD版本9.x中，堆叠是不推荐使用的，因为可以在单个级别上实现更多节点。有关详细信息，请参见定义网络连接。

4.18.1. 设备堆叠注意事项

以下注意事项适用于此类型的设置：

- 堆叠的设备是活动的。假设您已经配置了一个DRBD设备 /dev/drbd0 ，并且上面的堆叠设备是 /dev/drbd10 ，那么 /dev/drbd10 将是您装载和使用的设备。
- 设备元数据将存储两次，分别存储在底层DRBD设备 *和* 堆叠的DRBD设备上。在堆叠设备上，必须始终使用internal meta data。这意味着，与未堆叠的设备相比，堆叠设备上的有效可用存储区域稍微小一些。
- 要使堆叠的上层设备运行，底层设备必须处于primary角色。
- 要同步备份节点，活动节点上的堆叠设备必须处于启动状态并且处于主要角色。

4.18.2. 配置堆叠资源

在下面的示例中，节点名为 alice 、 bob 和 charlie ，其中 alice 和 bob 组成一个两节点集群， charlie 是备份节点。

```
resource r0 {
  protocol C;
  device    /dev/drbd0;
  disk      /dev/sda6;
  meta-disk internal;

  on alice {
    address   10.0.0.1:7788;
  }

  on bob {
    address   10.0.0.2:7788;
  }
}

resource r0-U {
  protocol A;

  stacked-on-top-of r0 {
    device    /dev/drbd10;
    address   192.168.42.1:7789;
  }

  on charlie {
    device    /dev/drbd10;
    disk      /dev/hda6;
    address   192.168.42.2:7789; # Public IP of the backup node
    meta-disk  internal;
  }
}
```

与任何 drbd.conf 配置文件一样，它必须分布在集群中的所有节点上 - 在本例中是三个节点。请注意，在未备份的资源配置中未找到以下额外关键字：

stacked-on-top-of

此选项通知DRBD包含它的资源是堆叠资源。它替换了通常在任何资源配置中找到的 on 部分之一。不要在较低级别的资源中使用 stacked-on-top-of 。

对于堆叠资源，不需要使用Protocol A。您可以根据您的应用程序选择任何DRBD的复制协议。

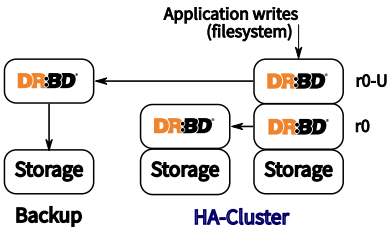


插图 8. 单堆叠设置

4.18.3. 启用堆叠资源

如果要启用堆叠资源，请先启用底层级别的资源后, 并将其提升：

```
drbdadm up r0
drbdadm primary r0
```

与未堆叠的资源一样，必须在堆叠的资源上创建DRBD元数据。使用以下命令完成此操作：

```
# drbdadm create-md --stacked r0-U
```

然后，可以启用堆叠资源：

```
# drbdadm up --stacked r0-U
# drbdadm primary --stacked r0-U
```

之后，您可以在备份节点上调出资源，启用三节点复制：

```
# drbdadm create-md r0-U
# drbdadm up r0-U
```

为了自动化堆叠资源管理，可以在群集管理器配置中集成堆叠资源。有关在Pacemaker群集管理框架管理的群集中执行此操作的信息，请参见在Pacemaker集群中使用堆叠的DRBD资源。

4.19. 永久无盘节点

在DRBD中，节点可能是永久无盘的。下面是一个配置示例，显示一个具有3个diskful节点（服务器）和一个永久无磁盘节点（客户端）的资源。

```
resource kvm-mail {
 device     /dev/drbd6;
 disk       /dev/vg/kvm-mail;
 meta-disk   internal;

 on store1 {
  address   10.1.10.1:7006;
  node-id   0;
 }
 on store2 {
  address   10.1.10.2:7006;
  node-id   1;
 }
 on store3 {
  address   10.1.10.3:7006;
  node-id   2;
 }

 on for-later-rebalancing {
  address   10.1.10.4:7006;
  node-id   3;
 }

 # DRBD "client"
 floating 10.1.11.6:8006 {
  disk     none;
  node-id   4;
 }

 # rest omitted for brevity
 ...
}
```

对于永久无盘节点，不分配位图插槽。对于此类节点，无盘状态显示为绿色，因为它不是错误或意外状态。

这个 *DRBD cliet* 是一种通过网络获取数据的简单方法，但它没有任何高级的iSCSI功能，比如 *Persistent Reservations*。 + 如果您的设置只需要基本的I/O需求，比如 *read*, *write*, *trim*/*discard* 又或是 *resize* (比如虚拟机），它应该会运转正常并满足基本的工作需求。

4.20. 数据再平衡

考虑到（示例）策略，即数据需要在3个节点上可用，您的设置至少需要3个服务器。

现在，随着存储需求的增长，您将需要更多的服务器。不必同时购买3台以上的服务器，您可以在一个额外的节点上重新平衡数据。

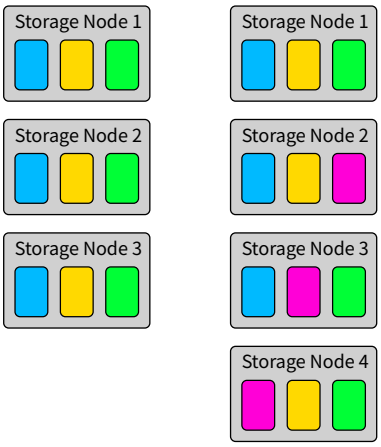


插图 9. DRBD数据再平衡

在上图中，您可以看到 *before* 和 *after* 状态：从3个节点（每个节点有3个25TiB卷）（净容量 75TiB）到4个节点（净容量 100TiB）。

要在集群中重新分配数据，必须选择一个 *新* 的节点，以及一个要删除此DRBD资源的节点。+ 请注意，从当前 *活动* 节点（即DRBD中 *主节点* ）中删除资源将涉及迁移服务或作为DRBD client在此节点上运行此资源；更容易选择处于 *次要角色* 的节点。（当然，这并不总是可能的。）

4.20.1. 准备位图插槽

您需要在每个具有要移动的资源的节点上有一个空闲的bitmap slot供临时使用。

您可以使用drbdadm create-md time再分配一个，或者只需在您的配置中放置一个占位符，这样 drbdadm 看到后就会再保留一个插槽：

```
resource r0 {
 ...
 on for-later-rebalancing {
  address   10.254.254.254:65533;
  node-id   3;
 }
}
```

如果您需要在实时使用期间提供该插槽，则必须

1. 转储元数据，
2. 扩大元数据空间，
3. 编辑转储文件，
4. 加载更改后的元数据。

在未来的版本中，drbdadm 将为您提供一个快捷方式；很可能您可以通过使用 drbdadm resize—peers *N* ，并让内核为您重写元数据。

4.20.2. 准备和激活新节点

首先，您必须在新节点上创建基础存储卷（使用例如 lvcreate ）。然后，可以用正确的主机名、地址和存储路径填充配置中的占位符。现在将资源配置复制到所有相关节点。

在新节点上，通过执行以下操作初始化元数据（一次）

```
# drbdadm create-md <resource>
v09 Magic number not found
Writing meta data...
initialising activity log
NOT initializing bitmap
New drbd meta data block successfully created.
```

4.20.3. 开始初始同步

现在新节点需要获取数据。

通过定义现有节点上的网络连接来完成操作

```
# drbdadm adjust <resource>
```

然后启动新节点上的DRBD设备

```
# drbdadm up <resource>
```

4.20.4. 检查连接

现在运行

```
# drbdadm status <resource>
```

在新节点上，检查是否连接了所有其他节点。

4.20.5. 初始同步后

只要新主机是 *UpToDate* 的，配置中的其他节点之一就可以重命名为 for-later-rebalancing ，并保留以进行另一次迁移。

也许您想对该节进行注释；尽管这样做的风险是，为新节点执行 drbdadm create md 时，下次重新平衡的位图插槽太少。+ 使用保留（未使用）的IP地址和主机名可能更容易。

再次复制更改的配置，并运行如下命令

```
# drbdadm adjust <resource>
```

在所有节点上。

4.20.6. 清理

在目前拥有数据但不再用于此资源的一个节点上，现在可以通过启动

```
# drbdadm down <resource>
```

现在，较低级别的存储设备不再使用，可以重新用于其他目的，或者，如果它是逻辑卷，则可以通过 lvremove 将其空间返回给卷组。

4.20.7. 结论和进一步步骤

其中一个资源已迁移到新节点。对于一个或多个其他资源也可以这样做，以释放现有集群中两个或三个节点上的空间。

Then new resources can be configured, as there are enough nodes with free space to achieve 3-way redundancy again.

不过，在使用DRBD Manage时，您可能需要将上述步骤与过程进行比较： [s-dm-rebalance]…

4.21. 配置仲裁

为了避免复制品的裂脑或数据分散，必须配置围栏。所有的围栏策略最终都依赖于冗余通信。这可能是一个管理网络的形式，它将节点连接到对等机的IPMI网络接口。在crm-fence-peer脚本的情况下，当DRBD的网络链路断开时，Pacemakers通信必须保持可用。

The quorum mechanism on the other hand takes a completely difference approach. The basic idea is that a cluster partition may only modify the replicated data set if the number of nodes that can communicate is greater than the half of the overall number of nodes. A node of such a partition *has quorum*. On the other hand a node does not have quorum needs to guarantee that the replicated data set it not touched, that it does not create a diverging data set.

通过将 quorum 资源选项设置为 majority 、all 或某个数值，可以启用DRBD中的仲裁实现。其中 majority 就是上一段中描述的行为。

4.21.1. 保证最小冗余

默认情况下，具有磁盘的每个节点都可以在仲裁选举中获得投票权。换句话说，只有无盘节点不计算在内。因此，具有两个 *不一致磁盘* 的分区将获得仲裁，而具有一个 *UpToDate* 分区的节点将在3节点群集中具有仲裁。通过配置 quorum minimum redundancy ，可以更改此行为，以便使得只有 UpToDate 节点在quorum选举中有投票权。该选项采用与 quorum 选项相同的参数。

使用此选项，表示您更希望等到最终必要的重新同步操作完成后再启动任何服务。因此，在某种程度上，您希望数据的最小冗余比服务的可用性得到保证。金融数据和服务就是一个浮现在脑海中的例子。

考虑以下5节点集群的示例。它要求一个分区至少有3个节点，其中两个必须是 UpToDate：

```
resource quorum-demo {
 options {
  quorum majority;
  quorum-minimum-redundancy 2;
  ...
 }
}
```

4.21.2. 丧失仲裁后的动作

当运行服务的节点失去仲裁时，它需要立即停止对数据集的写操作。这意味着IO会立即开始完成所有有错误的IO请求。通常这意味着不可能正常关闭，因为这需要对数据集进行更多修改。IO错误从块级别传播到文件系统，从文件系统传播到用户空间应用程序。

理想情况下，应用程序会在IO错误时终止。这允许Pacemaker卸载文件系统并将DRBD资源降级为次要角色。如果是这样，则应将 on-no-quorum 资源选项设置为 io-error 。下面是一个例子：

```
resource quorum-demo {
 options {
  quorum majority;
  on-no-quorum io-error;
  ...
 }
}
```

如果应用程序未在第一个IO错误时终止，则可以选择冻结IO并重新启动节点。下面是一个配置示例：

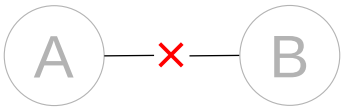
```
resource quorum-demo {
 options {
  quorum majority;
  on-no-quorum suspend-io;
  ...
 }

 handlers {
  quorum-lost "echo b > /proc/sysrq-trigger";
 }
 ...
}
```

4.21.3. 使用无盘节点作为分层断路器

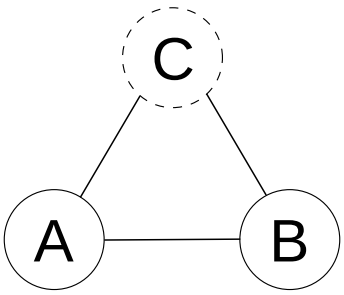
在仲裁协商过程中，可以使用连接到群集中所有节点的无盘节点来断开连接。

考虑以下两个节点群集，其中节点A是主节点，节点B是次节点：

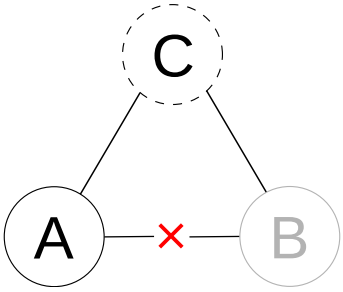
一旦两个节点之间的连接中断，它们就会失去仲裁，并且集群顶部的应用程序无法再写入数据。



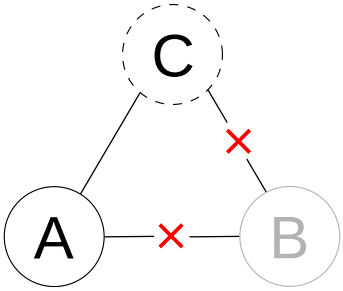
现在，如果我们将第三个节点C添加到集群并将其配置为无磁盘节点，我们就可以利用tiebraker机制。



在这种情况下，当主节点和辅助节点失去连接时，它们仍然可以 "看到" 无盘分层断路器。因此，主磁盘可以继续工作，而次磁盘将其磁盘降级为过期磁盘，因此无法在那里迁移服务。

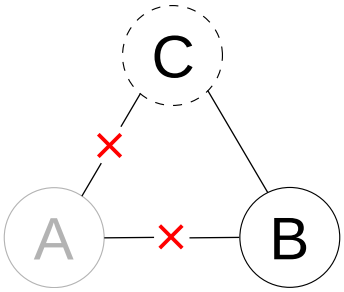


有一些特殊情况，以防两个连接失败。考虑以下场景：



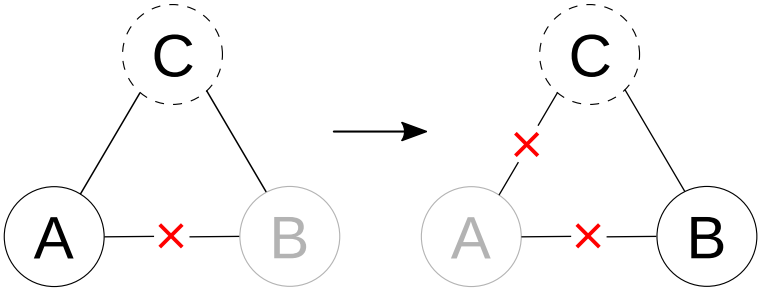
In this case, the tiebreaker node forms a partition with the primary node. The primary therefore keeps quorum, while the secondary becomes outdated. Note that the secondary’s disk state will be "UpToDate", but regardless it cannot be promoted to primary because it lacks quorum.

让我们考虑一下主断路器断开连接的可能性：



在这种情况下，主服务器将变得不可用，并进入 "quorum suspended" 状态。这有效地导致应用程序在DRBD之上接收I/O错误。然后，集群管理器可以将节点B提升为主节点，并使服务在那里运行。

如果无盘分层断路器 "切换侧"，则需要避免数据发散。考虑这个场景：

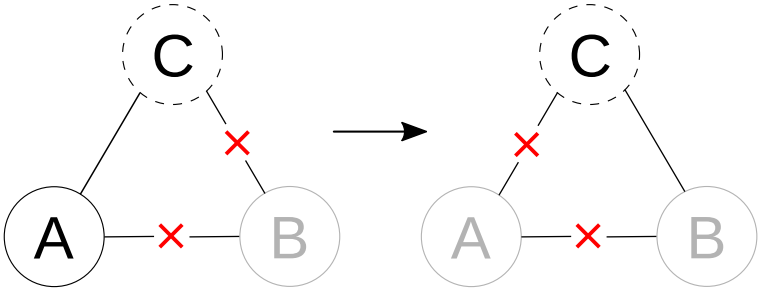


主节点和辅助节点之间的连接失败，应用程序继续在主节点上运行，此时主节点突然失去与无盘节点的连接。

在这种情况下，无法将任何节点升级到主节点，并且群集无法继续运行。

防止数据差异始终优先于确保服务可用性。

让我们看看另一个场景：



在这里，应用程序在主服务器上运行，而辅助服务器不可用。然后，tiebraker首先断开与主节点的连接，然后重新连接到辅助节点。这里需要注意的是， **失去仲裁的节点无法通过连接到无盘节点** 来重新获得仲裁。因此，在这种情况下，没有节点具有仲裁，集群将停止。

4.21.4. 最后站着的人

需要指出的是，优雅地离开集群的节点与失败节点的计数不同。在此上下文中， *优雅地离开* 意味着离开节点将其数据标记为过时，并且它能够告诉其余节点其数据已过时。

在所有磁盘都已过期的节点组中，该组中的任何人都无法升级到主角色 [7]。

An implication is that, if a cluster is left by all but one node, it can keep quorum as long as all the others left in a graceful way. If a single node leaves ungraceful, it has to assume that all away nodes form a partition and have access to up-to-date data. If the other partition might (potentially) be larger than my own partition it looses quorum.

[3] 至少这是编写时的状态-这是过去的情况，我们希望保持它的简单性。但谁知道呢？谁能讲明白呢？;)
[4] 经验法则是使用 ping 报告的时间
[5] 就像基准测试一样
[6] 例如，在DR站点中，您可能使用不同的硬件，对吗？
[7] 例外情况是使用—force标志进行手动升级。我们假设使用—force的人知道他在做什么

5. 使用DRBD代理

5.1. DRBD代理部署注意事项

DRBD Proxy进程可以直接位于设置drbd的机器上，也可以放置在不同的专用服务器上。一个DRBD代理实例可以作为分布在多个节点上的多个DRBD设备的代理。

DRBD代理对DRBD完全透明。通常，集群运行中有大量的数据包，因此活动日志应该相当大。由于这可能导致主节点崩溃后重新同步运行的时间更长，建议启用DRBD的 csums-alg 设置。

有关DRBD代理的基本原理的更多信息，请参见功能说明通过DRBD代理进行远程复制。

DRBD Proxy 3使用了几个内核特性，这些特性仅在2.6.26版之后才可用，因此在较旧的系统（如RHEL 5）上运行是不可能的；在这里，我们仍然可以提供DRBD Proxy v1版本的包. [8].

5.2. 安装

要获取DRBD代理，请与Linbit销售代表联系。除非另有说明，请始终使用最新的DRBD代理版本。

要在基于Debian和Debian的系统上安装DRBD Proxy，请使用dpkg工具，如下所示（用DRBD Proxy版本替换版本，用目标体系结构替换体系结构）：

```
# dpkg -i drbd-proxy_3.2.2_amd64.deb
```

要在基于RPM的系统（如SLES或RHEL）上安装DRBD代理，请使用以下RPM工具（将版本替换为DRBD代理版本，将体系结构替换为目标体系结构）：

```
# rpm -i drbd-proxy-3.2.2-1.x86_64.rpm
```

同时安装DRBD管理程序drbdadm，因为需要配置DRBD代理。

这将安装DRBD代理二进制文件以及通常进入 /etc/init.d 的init脚本。请始终使用init脚本启动/停止DRBD proxy，因为它还使用 drbdadm 工具配置DRBD proxy。

5.3. 许可证文件

从Linbit获取许可证时，将向您发送运行DRBD Proxy所需的DRBD Proxy许可证文件。该文件名为 drbd-proxy.license ，必须将其复制到目标计算机的 /etc 目录中，并由用户/组 drbdpxy 拥有。

```
# cp drbd-proxy.license /etc/
```

5.4. 使用LINSTOR配置

可以使用LINSTOR配置DRBD代理，如LINSTOR用户指南中所述。

5.5. 使用资源文件配置

DRBD代理也可以通过编辑资源文件来配置。它由主机部分中名为 proxy 的附加部分和名为 proxy on 的附加部分配置。

下面是直接在DRBD节点上运行的代理的DRBD配置示例：

```
resource r0 {
  protocol A;
  device    /dev/drbd15;
  disk      /dev/VG/r0;
  meta-disk  internal;

  proxy {
    memlimit 512M;
    plugin {
      zlib level 9;
    }
  }

  on alice {
    address 127.0.0.1:7915;
    proxy on alice {
      inside 127.0.0.1:7815;
      outside 192.168.23.1:7715;
    }
  }

  on bob {
    address 127.0.0.1:7915;
    proxy on bob {
      inside 127.0.0.1:7815;
      outside 192.168.23.2:7715;
    }
  }
}
```

inside IP地址用于DRBD和DRBD代理之间的通信，而 outside IP地址用于代理之间的通信。后一个通道可能必须在防火墙设置中被允许。

5.6. 控制DRBD代理

drbdadm 提供了 proxy-up 和 proxy-down 子命令，用于配置或删除与命名DRBD资源的本地DRBD代理进程的连接。这些命令由 /etc/init.d/drbdproxy 实现的 start 和 stop 操作使用。

DRBD代理有一个低级的配置工具，称为 drbd-proxy-ctl。在没有任何选项的情况下调用时，它以交互模式运行。

要避免交互模式，直接传递命令，请在命令后面使用 -c 参数。

要显示可用的命令，请使用：

```
# drbd-proxy-ctl -c "help"
```

注意传递的命令周围的双引号。

Here is a list of commands; while the first few ones are typically only used indirectly (via drbdadm proxy-up resp. drbdadm proxy-down), the latter ones give various status information.

add connection *<name> lots of arguments*

创建通信路径。由于这是通过 drbdadm proxy-up 运行的，因此这里省略了长参数列表。

del connection *<name>*
删除通信路径。

set memlimit *<name>* *<memlimit-in-bytes>*
设置连接的内存限制；这只能在重新设置连接时完成，无法在运行时更改。+ 这个命令理解常用的 k、 M 和 G 单位。

show
显示当前配置的通信路径。

show memusage
显示每个连接的内存使用情况。+

举个例子，

```
# watch -n 1 'drbd-proxy-ctl -c "show memusage"'
```

监视内存使用情况。请注意，上述的反斜杠是必需的。

show [h]subconnections
显示当前建立的个人连接以及一些统计信息。h 选项以可读格式输出字节。

show [h]connections
显示当前配置的连接及其状态，h 选项以可读格式输出字节。+

Status 列将显示以下状态之一：

- *Off*：与远程DRBD代理进程没有通信。
- *Half-up*： 可以建立到远程DRBD代理的连接；Proxy ⇒ DRBD路径尚未启动。
- *DRBD-conn*: The first few packets are being pushed across the connection; but still e.g. a Split-Brain situation might serve it again.
- *Up*: DRBD连接已完全建立。

shutdown
关闭 drbd proxy 程序。注意：这将无条件终止使用DRBD代理的任何DRBD连接。

quit
退出客户端程序（关闭控制连接），但保留DRBD代理运行。

print statistics
这将以易于解析的格式打印当前活动连接的详细统计信息。使用这个来集成到您的监控解决方案！+

虽然上面的命令只能从UID 0（即 root 用户）接受，但任何用户都可以使用这个命令（前提是unix权限允许访问位于 /var/run/drbd-proxy/drbd-proxy-ctl.socket ）的代理套接字；有关设置权限的信息，请参阅 /etc/init.d/drbdproxy 上的init脚本。

5.7. 关于DRBD代理插件

从DRBD Proxy版本3起，代理允许为广域网连接启用一些特定的插件。+ 当前可用的插件有 zstd 、 lz4 、 zlib 和 lzma（所有软件压缩）和 aha（硬件压缩支持，请参见 http://www.aha.com/data-compression/ ）。

zstd（Zstandard）是一种实时压缩算法，具有很高的压缩比。它提供了一个非常广泛的压缩/速度权衡，同时有一个非常快速的解码器支持。压缩率取决于 "level" 参数，该参数可以设置在1到22之间。在20级以上，Drbd代理将需要更多的内存。

lz4 是一种非常快速的压缩算法；数据通常被压缩1:2到1:4，可以节省一半到三分之二的带宽。

zlib 插件使用GZIP算法进行压缩；它使用的CPU比 lz4 多一点，但给出的比率是1:3到1:5。

lzma 插件使用 liblzma2 库。它可以使用几百个MiB的字典；这些字典允许对重复数据进行非常有效的增量压缩，即使是很小的更改。lzma 需要更多的CPU和内存，但其压缩效果要比 zlib 好得多 - 在DRBD上安装VM的实际测试给出了1:10到1:40的比率。必须在许可证中启用 lzma 插件。

aha 使用硬件压缩卡，如AHA367PCIe（10Gbit/sec）或AHA372（20GBit/sec）；这是当代硬件最快的压缩。+ 你需要在你的许可文件中有一个特殊的标志来启用这个插件。

请联系LINBIT为您的环境找到最佳设置-这取决于CPU（速度、线程数）、可用内存、输入和可用输出带宽以及预期的IO峰值。如果有一周的 sysstat 数据可用，这也能有助于确定配置。

请注意，proxy 部分中较旧的 compression on 已弃用，并将在以后的版本中删除。+ 目前它被视为 zlib level 9。

5.7.1. 使用广域网侧带宽限制

DRBD Proxy 的 bwrimit 选项已不可用。不要使用它，因为它可能会导致DRBD上的应用程序在IO上阻塞。它将被移除。

相反，使用Linux内核的流量控制框架来限制广域网端代理所消耗的带宽。

在下面的示例中，您需要替换对等机的接口名称、源端口和ip地址。

```
# tc qdisc add dev eth0 root handle 1: htb default 1
# tc class add dev eth0 parent 1: classid 1:1 htb rate 1gbit
# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 500kbit
# tc filter add dev eth0 parent 1: protocol ip prio 16 u32 \
      match ip sport 7000 0xffff \
      match ip dst 192.168.47.11 flowid 1:10
# tc filter add dev eth0 parent 1: protocol ip prio 16 u32 \
      match ip dport 7000 0xffff \
      match ip dst 192.168.47.11 flowid 1:10
```

您可以使用

```
# tc qdisc del dev eth0 root handle 1
```

5.8. 故障排除

DRBD代理使用 LOG DAEMON 工具通过syslog进行日志记录。通常您会在 /var/log/daemon.log 中找到DRBD代理消息。

可以使用以下命令在DRBD Proxy中启用调试模式。

```
# drbd-proxy-ctl -c 'set loglevel debug'
```

例如，如果代理连接失败，它将记录类似 Rejecting connection because I can’t connect on the other side 的内容。在这种情况下，请检查两个节点上的DRBD是否都在运行（不是独立模式），以及两个代理是否都在运行。还要仔细检查您的配置。

[8] v1使用不同的调度模型，因此不会达到与v3相同的性能；因此，即使您的生产设置仍然是RHEL 5，也许您可以在每个数据中心运行一个RHEL 6/7虚拟机?

6. 故障排除和错误恢复

本章描述硬件或系统故障时要执行的任务。

6.1. 处理硬盘故障

如何处理硬盘故障取决于DRBD处理磁盘I/O错误的配置方式（请参见磁盘错误处理策略），以及配置的元数据类型（请参见DRBD元数据）。

在大多数情况下，这里描述的步骤仅适用于直接在物理硬盘上运行DRBD的情况。它们通常不适用于在

- MD软件RAID集（在本例中，使用 mdadm 管理驱动器更换），
- 设备映射器RAID（使用 dmraid ），
- 一个硬件RAID设备（遵循供应商关于如何处理故障驱动器的说明），
- 一些非标准设备映射器虚拟块设备（请参阅设备映射器文档）。

6.1.1. 从硬盘手动分离DRBD

如果DRBD是configured to pass on I/O errors（不推荐），则必须首先分离DRBD资源，即，将其与其备份存储解除关联：

```
# drbdadm detach <resource>
```

通过运行 drbdadm status 或 drbdadm dstate 命令，您现在可以验证资源是否现在处于 *diskless mode*:

```
# drbdadm status <resource>
<resource> role:Primary
  volume:0 disk:Diskless
  <peer> role:Secondary
    volume:0 peer-disk:UpToDate
# drbdadm dstate <resource>
Diskless/UpToDate
```

If the disk failure has occurred on your primary node, you may combine this step with a switch-over operation.

6.1.2. I/O错误时自动分离

如果DRBD是configured to automatically detach upon I/O error时自动分离（推荐选项），DRBD应该已经自动将资源从其备份存储中分离，而无需手动干预。您仍然可以使用 drbdadm status 命令来验证资源实际上是在无盘模式下运行的。

6.1.3. 使用内部元数据时替换故障磁盘

如果使用internal meta data，则足以将DRBD设备绑定到新硬盘。如果新硬盘必须用另一个Linux设备名而不是缺陷磁盘寻址，则必须相应地修改DRBD配置文件。

此过程包括创建新的元数据集，然后重新附加资源：

```
# drbdadm create-md <resource>
v08 Magic number not found
Writing meta data...
initialising activity log
NOT initializing bitmap
New drbd meta data block successfully created.

# drbdadm attach <resource>
```

新硬盘的完全同步将立即自动启动。您将能够通过 drbdadm status --verbose 监视同步的进度，就像任何后台同步一样。

6.1.4. 使用外部元数据时替换故障磁盘

使用external meta data时，程序基本相同。然而，DRBD无法独立地识别硬盘驱动器已被交换，因此需要额外的步骤。

```
# drbdadm create-md <resource>
v08 Magic number not found
Writing meta data...
initialising activity log
NOT initializing bitmap
New drbd meta data block successfully created.

# drbdadm attach <resource>
# drbdadm invalidate <resource>
```

请确保在 *没有* 完好数据的节点上运行 drbdadm invalidate 此命令将导致本地内容被对等方的数据覆盖，因此在错误的节点上运行此命令可能会丢失数据！

这里，drbdadm invalidate 命令触发同步。同样，可以通过 drbdadm status --verbose 来观察同步进度。

6.2. 处理节点故障

当DRBD检测到其对等节点关闭时（通过真正的硬件故障或手动干预），DRBD将其连接状态从 *Connected* 更改为 *Connecting* ，并等待对等节点重新出现。然后，DRBD资源被称为在 *disconnected mode* 模式下工作。在断开连接模式下，资源及其关联的块设备完全可用，并且可以根据需要升级和降级，但不会将块修改复制到对等节点。相反，DRBD以每个对等点为基础存储断开连接时正在修改的块。

6.2.1. 处理临时辅助节点故障

如果当前具有次要角色中的资源的节点暂时出现故障（例如，由于随后通过替换RAM纠正的内存问题），则无需进一步干预-除了修复故障节点并使其重新联机的明显必要性之外。当这种情况发生时，两个节点只需在系统启动时重新建立连接。在此之后，DRBD将同时在主节点上所做的所有修改同步到辅助节点。

此时，由于DRBD的重新同步算法的性质，资源在次要节点上短暂地不一致。在短时间内，如果对等节点不可用，则辅助节点无法切换到主角色。因此，集群不冗余的时间段由实际的辅助节点停机时间加上随后的重新同步组成。

请注意，使用DRBD 9，每个资源可以连接两个以上的节点，因此对于例如4个节点，一个失败的辅助节点仍然保留两个其他的辅助节点用于故障转移。

6.2.2. 处理临时主节点故障

从DRBD的角度来看，主节点的故障几乎等同于次节点的故障。幸存节点检测到对等节点的故障，并切换到断开模式。DRBD不会将幸存节点提升为主要角色；集群管理应用程序有责任这样做。

当故障节点被修复并返回集群时，它将以次要角色执行此操作，因此，如前一节所述，不需要进一步的手动干预。同样，DRBD不会重新更改资源角色，这取决于集群管理器是否这样做（如果这样配置的话）。

DRBD通过一种特殊的机制，在主节点发生故障的情况下保证块设备的一致性。有关详细讨论，请参阅活动日志。

6.2.3. 处理永久性节点故障

如果节点遇到不可恢复的问题或永久性破坏，则必须执行以下步骤：

- 将出现故障的硬件替换为具有类似性能和磁盘容量的硬件。

可以用性能较差的节点替换出现故障的节点，但不建议这样做。不支持将出现故障的节点替换为磁盘容量较小的节点，这将导致DRBD拒绝连接到被替换的节点 [9].

- 安装基本系统和应用程序。
- 安装DRBD并从一个幸存的节点复制 /etc/drbd.conf 和所有 /etc/drbd.d/ 文件。
- 按照配置DRBD中概述的步骤操作，但不要忘记参考 初始设备同步.

此时不需要手动启动完全设备同步，它将在连接到幸存的主节点和/或辅助节点时自动启动。

6.3. 手动恢复裂脑

DRBD在连接再次可用且对等节点交换初始DRBD协议握手时检测到split brain。如果DRBD检测到这两个节点（或者在某个点上，在断开连接的情况下）都处于主角色中，它会立即断开复制连接。这是一条类似以下信息的信号，显示在系统日志中：

```
Split-Brain detected, dropping connection!
```

检测到裂脑后，一个节点的资源将始终处于 *StandAlone* 连接状态。另一个节点也可能处于 *StandAlone* 状态（如果两个节点同时检测到裂脑），或者处于 *Connecting*（如果对等节点在另一个节点有机会检测到裂脑之前断开了连接）。

此时，除非您将DRBD配置为自动从裂脑恢复，否则必须通过选择一个将放弃其修改的节点（该节点称为 *split brain victim*）进行手动干预。使用以下命令进行干预：

这项工作仍在进行中。

期待能大致有所改观。

```
# drbdadm disconnect <resource>
# drbdadm secondary <resource>
# drbdadm connect --discard-my-data <resource>
```

在另一个节点（*split brain survivor*）上，如果其连接状态也是 *StandAlone* 的，则输入：

```
# drbdadm disconnect <resource>
# drbdadm connect <resource>
```

如果节点已处于 *Connecting* 状态，则可以省略此步骤；然后将自动重新连接。

连接后，裂脑受害者立即将其连接状态更改为 *SyncTarget*，并由其他节点覆盖其修改。

裂脑受害者不受全设备同步的影响。取而代之的是，它的局部修改被回滚，对裂脑幸存者所做的任何修改都会传播给受害者。

在重新同步完成后，裂脑被视为已解决，节点再次形成完全一致的冗余复制存储系统。

[9] 无论如何，它无法复制数据！

支持DRBD的应用程序

7. 将DRBD与Pacemaker集成

结合使用DRBD和Pacemaker集群栈可以说是DRBD最常见的用例。Pacemaker也是使DRBD在各种各样的使用场景中非常强大的应用程序之一。

DRBD can be used in Pacemaker clusters in different ways:

- 作为后台服务运行的DRBD，用作SAN;或
- DRBD completely managed by Pacemaker via the DRBD OCF resource agent

两者都有一些优点和缺点，这些将在下面讨论。

It’s recommended to have either fencing configured or quorum enabled. (But not both. External fencing handler results may interact in conflicting ways with DRBD internal quorum.) If your cluster has communication issues (e.g. network switch loses power) and gets split, the parts might start the services (failover) and cause a **Split-Brain** when the communication resumes again.

7.1. Pacemaker基础概念

Pacemaker是一个复杂的、功能丰富的、广泛部署的Linux平台集群资源管理器。它附带了一组丰富的文档。为了理解本章，强烈建议阅读以下文件：

- Clusters from Scratch，配置高可用性集群的分步指南；
- CRM CLI（命令行界面）工具，CRM shell手册，一个与Pacemaker捆绑在一起的简单直观的命令行界面；
- 解释Pacemaker配置，解释Pacemaker背后的概念和设计的参考文件。

7.2. 在Pacemaker集群中使用DRBD作为后台服务

在本节中，您将看到使用自主DRBD存储看起来像本地存储；因此，在Pacemaker集群中集成是通过将挂载点指向DRBD来完成的。

首先，我们将使用DRBD的 auto-promote 特性，以便DRBD在需要时自动设置自己的 *Primary* 。这可能适用于您的所有资源，因此设置 common 部分中的默认值是有意义的：

```
common {
 options {
  auto-promote yes;
  ...
 }
}
```

现在您只需要使用存储，例如通过文件系统：

Listing 4. 使用 auto promote 为DRBD支持的MySQL服务配置Pacemaker

```
crm configure
crm(live)configure# primitive fs_mysql ocf:heartbeat:Filesystem \
          params device="/dev/drbd/by-res/mysql/0" \
           directory="/var/lib/mysql" fstype="ext3"
crm(live)configure# primitive ip_mysql ocf:heartbeat:IPaddr2 \
          params ip="10.9.42.1" nic="eth0"
crm(live)configure# primitive mysqld lsb:mysqld
crm(live)configure# group mysql fs_mysql ip_mysql mysqld
crm(live)configure# commit
crm(live)configure# exit
bye
```

实际上，所需要的只是一个挂载点（在本例中为`/var/lib/mysql`），DRBD资源在这里挂载。

只要Pacemaker有控制权，它就只允许在集群中安装一个实例。

See also Importing DRBD’s promotion scores into the CIB for additional information on ordering constraints for system startup and more.

7.3. 向集群配置添加DRBD支持的服务，包括主-从资源

本节介绍如何在Pacemaker集群中启用DRBD支持的服务。

如果您使用的是DRBD OCF资源代理，建议您将DRBD的启动、关闭、升级和降级 *专门* 推迟到OCF资源代理。这意味着您应该禁用DRBD init脚本：

```
chkconfig drbd off
```

ocf:linbit:drbd ocf资源代理提供主/从功能，允许Pacemaker启动和监视多个节点上的drbd资源，并根据需要进行升级和降级。但是，您必须了解，DRBD RA在Pacemaker关闭时，以及在为节点启用待机模式时，会断开并分离它管理的所有DRBD资源。

DRBD附带的OCF资源代理属于 linbit 提供程序，因此安装为`/usr/lib/OCF/resource.d/linbit/DRBD`。有一个遗留资源代理作为OCF资源代理包的一部分提供，它使用 heartbeat 提供程序并安装到 /usr/lib/OCF/resource.d/heartbeat/drbd 。旧的OCF RA已弃用，不应再使用。

为了使用 DRBD OCF资源代理为Pacemaker CRM集群中的MySQL数据库启用DRBD支持的配置，必须同时创建必要的资源和Pacemaker约束，以确保您的服务仅在以前升级的DRBD资源上启动。您可以使用 crm shell 执行此操作，如下例所示：

Listing 5. 使用 master-slave 资源为DRBD支持的MySQL服务配置Pacemaker

```
crm configure
crm(live)configure# primitive drbd_mysql ocf:linbit:drbd \
        params drbd_resource="mysql" \
        op monitor interval="29s" role="Master" \
        op monitor interval="31s" role="Slave"
crm(live)configure# ms ms_drbd_mysql drbd_mysql \
        meta master-max="1" master-node-max="1" \
          clone-max="2" clone-node-max="1" \
          notify="true"
crm(live)configure# primitive fs_mysql ocf:heartbeat:Filesystem \
        params device="/dev/drbd/by-res/mysql/0" \
         directory="/var/lib/mysql" fstype="ext3"
crm(live)configure# primitive ip_mysql ocf:heartbeat:IPaddr2 \
        params ip="10.9.42.1" nic="eth0"
crm(live)configure# primitive mysqld lsb:mysqld
crm(live)configure# group mysql fs_mysql ip_mysql mysqld
crm(live)configure# colocation mysql_on_drbd \
          inf: mysql ms_drbd_mysql:Master
crm(live)configure# order mysql_after_drbd \
          inf: ms_drbd_mysql:promote mysql:start
crm(live)configure# commit
crm(live)configure# exit
bye
```

在此之后，应启用配置。Pacemaker现在选择一个节点，在该节点上提升DRBD资源，然后在同一节点上启动DRBD支持的资源组。

See also Importing DRBD’s promotion scores into the CIB for additional information on location contraints for placing the Master role.

7.4. 在Pacemaker集群中使用资源级围栏

本节概述了在DRBD复制链接中断时防止Pacemaker升级DRBD主/从资源所需的步骤。这使得Pacemaker不会使用过时的数据启动服务，也不会在启动过程中造成不必要的 时间扭曲 。

要为DRBD启用任何资源级围栏，必须在资源配置中添加以下行：

```
resource <resource> {
  net {
    fencing resource-only;
    ...
  }
}
```

You will also have to make changes to the handlers section depending on the cluster infrastructure being used.

Corosync-based Pacemaker clusters can use the functionality explained in 使用集群信息库（CIB）的资源级围栏.

It is absolutely vital to configure at least two independent cluster communications channels for this functionality to work correctly. Corosync clusters should list at least two redundant rings in corosync.conf, respectively several paths for knet.

7.4.1. 使用集群信息库（CIB）的资源级围栏

要为Pacemaker启用资源级围栏，必须在 drbd.conf 中设置两个选项：

```
resource <resource> {
 net {
  fencing resource-only;
  ...
 }
 handlers {
  fence-peer "/usr/lib/drbd/crm-fence-peer.9.sh";
  unfence-peer "/usr/lib/drbd/crm-unfence-peer.9.sh";
  # Note: we used to abuse the after-resync-target handler to do the
  # unfence, but since 2016 have a dedicated unfence-peer handler.
  # Using the after-resync-target handler is wrong in some corner cases.
  ...
 }
 ...
}
```

因此，如果DRBD复制链接断开连接， crm-fence-peer.9.sh 脚本将联系集群管理器，确定与此DRBD资源关联的Pacemaker主/从资源，并确保主/从资源不再在当前活动的节点以外的任何节点上升级。相反，当重新建立连接并且DRBD完成其同步过程时，则移除该约束，并且集群管理器可以自由地再次提升任何节点上的资源。

7.5. 在Pacemaker集群中使用堆叠的DRBD资源

DRBD版本9.x中不建议使用堆栈，因为可以在单个级别上实现更多节点。有关详细信息，请参见定义网络连接。

堆叠资源允许DRBD用于多节点集群中的多级冗余，或建立非现场灾难恢复能力。本节描述如何在这种配置中配置DRBD和Pacemaker。

7.5.1. 将异地灾难恢复添加到Pacemaker集群

在这个配置场景中，我们将在一个站点中处理一个两节点高可用性集群，外加一个单独的节点，该节点可能位于异地。第三个节点充当灾难恢复节点，是一个独立的服务器。考虑下面的插图来描述这个概念。

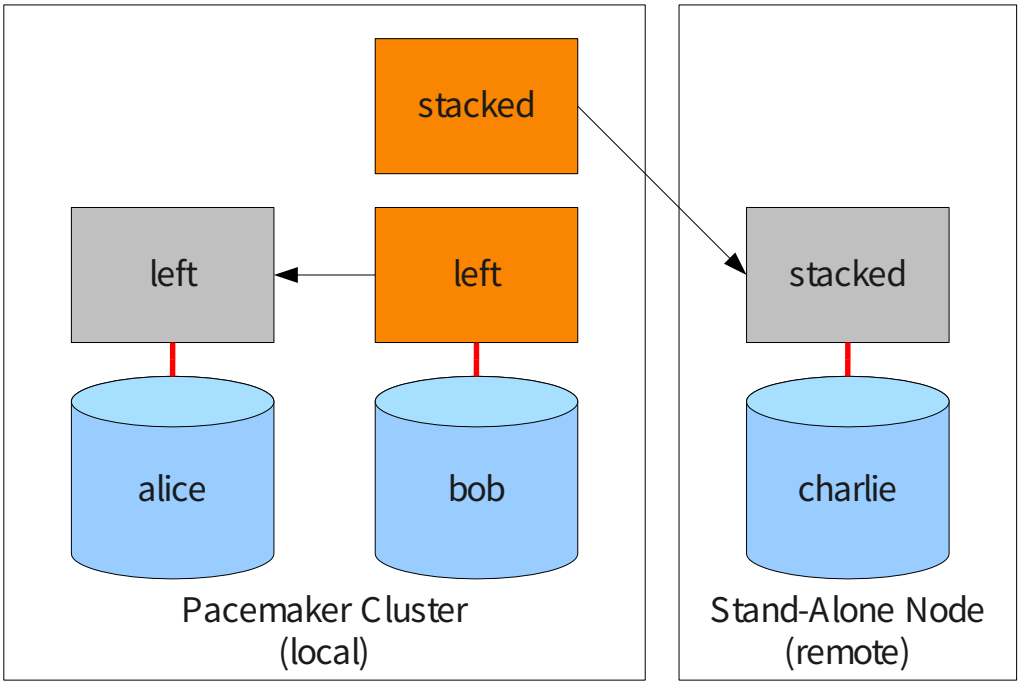


插图 10. Pacemaker集群中的DRBD资源叠加

在本例中， alice 和 bob 组成了一个两节点Pacemaker集群，而 charlie 是一个非现场节点，不由Pacemaker管理。

要创建这样的配置，首先要配置和初始化DRBD资源，如创建堆叠的三节点设置中所述。然后，使用以下CRM配置配置Pacemaker：

```
primitive p_drbd_r0 ocf:linbit:drbd \
  params drbd_resource="r0"

primitive p_drbd_r0-U ocf:linbit:drbd \
  params drbd_resource="r0-U"

primitive p_ip_stacked ocf:heartbeat:IPaddr2 \
  params ip="192.168.42.1" nic="eth0"

ms ms_drbd_r0 p_drbd_r0 \
  meta master-max="1" master-node-max="1" \
    clone-max="2" clone-node-max="1" \
    notify="true" globally-unique="false"

ms ms_drbd_r0-U p_drbd_r0-U \
  meta master-max="1" clone-max="1" \
    clone-node-max="1" master-node-max="1" \
    notify="true" globally-unique="false"

colocation c_drbd_r0-U_on_drbd_r0 \
    inf: ms_drbd_r0-U ms_drbd_r0:Master

colocation c_drbd_r0-U_on_ip \
    inf: ms_drbd_r0-U p_ip_stacked

colocation c_ip_on_r0_master \
    inf: p_ip_stacked ms_drbd_r0:Master

order o_ip_before_r0-U \
    inf: p_ip_stacked ms_drbd_r0-U:start

order o_drbd_r0_before_r0-U \
    inf: ms_drbd_r0:promote ms_drbd_r0-U:start
```

假设您在名为 /tmp/crm.txt 的临时文件中创建了此配置，则可以使用以下命令将其导入实时集群配置：

```
crm configure < /tmp/crm.txt
```

此配置将确保在 alice / bob 集群上按正确顺序执行以下操作：

1. Pacemaker在两个集群节点上启动DRBD资源 r0 ，并将一个节点提升为Master（DRBD Primary）角色。
2. Pacemaker然后启动IP地址192.168.42.1，堆叠的资源将用于复制到第三个节点。它在先前提升为 r0 DRBD资源的Master角色的节点上执行此操作。
3. 在现在具有 r0 的主要角色和 r0-U 的复制IP地址的节点上，Pacemaker现在启动 r0-U DRBD资源，该资源连接并复制到非站点节点。
4. 然后Pacemaker也将 r0-U 资源提升为主要角色，以便应用程序可以使用它。

因此，这种Pacemaker配置确保集群节点之间不仅有完全的数据冗余，而且还有到第三个非现场节点的数据冗余。

这种设置通常与DRBD Proxy一起部署。

7.5.2. 利用堆叠资源实现Pacemaker集群的4路冗余

在这种配置中，总共使用三个DRBD资源（两个未堆叠，一个堆叠）来实现4路存储冗余。这意味着，对于一个4节点集群，最多有3个节点在仍然提供服务可用性的情况下可能会出现故障。

考虑下面的插图来解释这个概念。

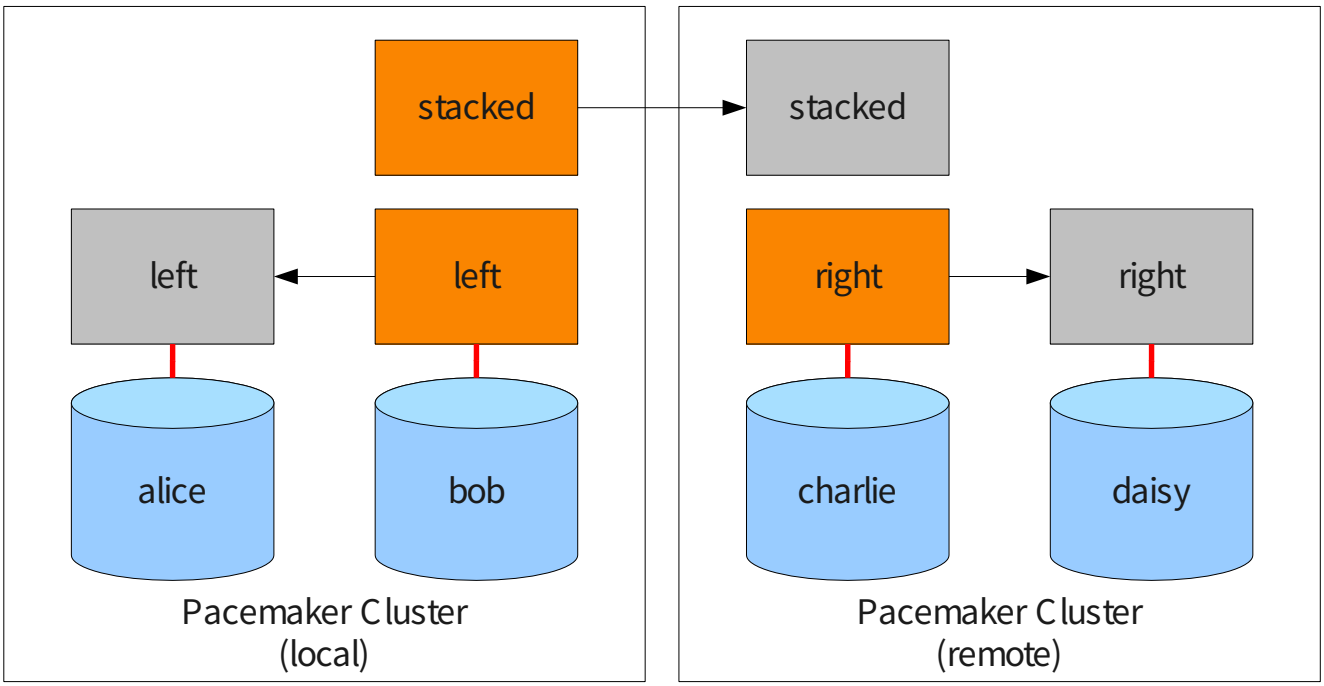


插图 11. Pacemaker集群中的DRBD资源叠加

在本例中， alice 、 bob 、 charlie 和 daisy 构成两个双节点Pacemaker集群alice和bob组成了名为 left 的集群，并在它们之间使用DRBD资源复制数据，而charlie和daisy在名为 right 的集群中使用单独的DRBD资源复制数据。第三者，堆叠的DRBD资源连接两个集群。

由于Pacemaker集群管理器自Pacemaker 1.0.5版起的限制，在不禁用CIB验证的情况下无法在单个四节点集群中创建此设置，这是一个高级过程，不建议用于一般用途。预计这将在未来的Pacemaker版本中得到解决。

要创建这样的配置，您首先要配置和初始化DRBD资源，如创建堆叠的三节点设置中所述（除了DRBD配置的远程部分也是堆叠的，而不仅仅是本地集群）。然后，使用以下CRM配置配置Pacemaker，从集群 left 开始：

```
primitive p_drbd_left ocf:linbit:drbd \
   params drbd_resource="left"

primitive p_drbd_stacked ocf:linbit:drbd \
   params drbd_resource="stacked"

primitive p_ip_stacked_left ocf:heartbeat:IPaddr2 \
   params ip="10.9.9.100" nic="eth0"

ms ms_drbd_left p_drbd_left \
   meta master-max="1" master-node-max="1" \
      clone-max="2" clone-node-max="1" \
      notify="true"

ms ms_drbd_stacked p_drbd_stacked \
   meta master-max="1" clone-max="1" \
      clone-node-max="1" master-node-max="1" \
      notify="true" target-role="Master"

colocation c_ip_on_left_master \
      inf: p_ip_stacked_left ms_drbd_left:Master

colocation c_drbd_stacked_on_ip_left \
      inf: ms_drbd_stacked p_ip_stacked_left

order o_ip_before_stacked_left \
      inf: p_ip_stacked_left ms_drbd_stacked:start

order o_drbd_left_before_stacked_left \
      inf: ms_drbd_left:promote ms_drbd_stacked:start
```

假设您在名为 /tmp/crm.txt 的临时文件中创建了此配置，则可以使用以下命令将其导入实时集群配置：

```
crm configure < /tmp/crm.txt
```

将此配置添加到CIB后，Pacemaker将执行以下操作：

1. 调出DRBD资源 left ，在 alice 和 bob 之间复制，将资源提升到其中一个节点上的Master角色。
2. 调出IP地址10.9.9.100（在 alice 或 bob 上，具体取决于其中哪一个拥有资源 left 的主角色）。
3. 将DRBD资源 stacked 调到保存刚才配置的IP地址的同一节点上。
4. 将堆叠的DRBD资源提升为主角色。

现在，在集群 right 上继续创建以下配置：

```
primitive p_drbd_right ocf:linbit:drbd \
   params drbd_resource="right"

primitive p_drbd_stacked ocf:linbit:drbd \
   params drbd_resource="stacked"

primitive p_ip_stacked_right ocf:heartbeat:IPaddr2 \
   params ip="10.9.10.101" nic="eth0"

ms ms_drbd_right p_drbd_right \
   meta master-max="1" master-node-max="1" \
      clone-max="2" clone-node-max="1" \
      notify="true"

ms ms_drbd_stacked p_drbd_stacked \
   meta master-max="1" clone-max="1" \
      clone-node-max="1" master-node-max="1" \
      notify="true" target-role="Slave"

colocation c_drbd_stacked_on_ip_right \
      inf: ms_drbd_stacked p_ip_stacked_right

colocation c_ip_on_right_master \
      inf: p_ip_stacked_right ms_drbd_right:Master

order o_ip_before_stacked_right \
      inf: p_ip_stacked_right ms_drbd_stacked:start

order o_drbd_right_before_stacked_right \
      inf: ms_drbd_right:promote ms_drbd_stacked:start
```

将此配置添加到CIB后，Pacemaker将执行以下操作：

1. 在 charlie 和 daisy 之间启动DRBD资源 right 复制，将资源提升到其中一个节点上的Master角色。
2. 调出IP地址10.9.10.101（在 charlie 或 daisy 上，具体取决于其中哪个拥有资源 right 的主角色）。
3. 将DRBD资源 stacked 调到保存刚才配置的IP地址的同一节点上。
4. 将堆叠的DRBD资源保留在次要角色中（由于 target role= Slave ``）。

7.6. 配置DRBD以在两个支持SAN的Pacemaker集群之间进行复制

这是一种比较高级的设置，通常用于拆分站点配置。它包括两个独立的Pacemaker集群，每个集群都可以访问单独的存储区域网络（SAN）。然后使用DRBD通过站点之间的IP链路复制存储在该SAN上的数据。

考虑下面的插图来描述这个概念。

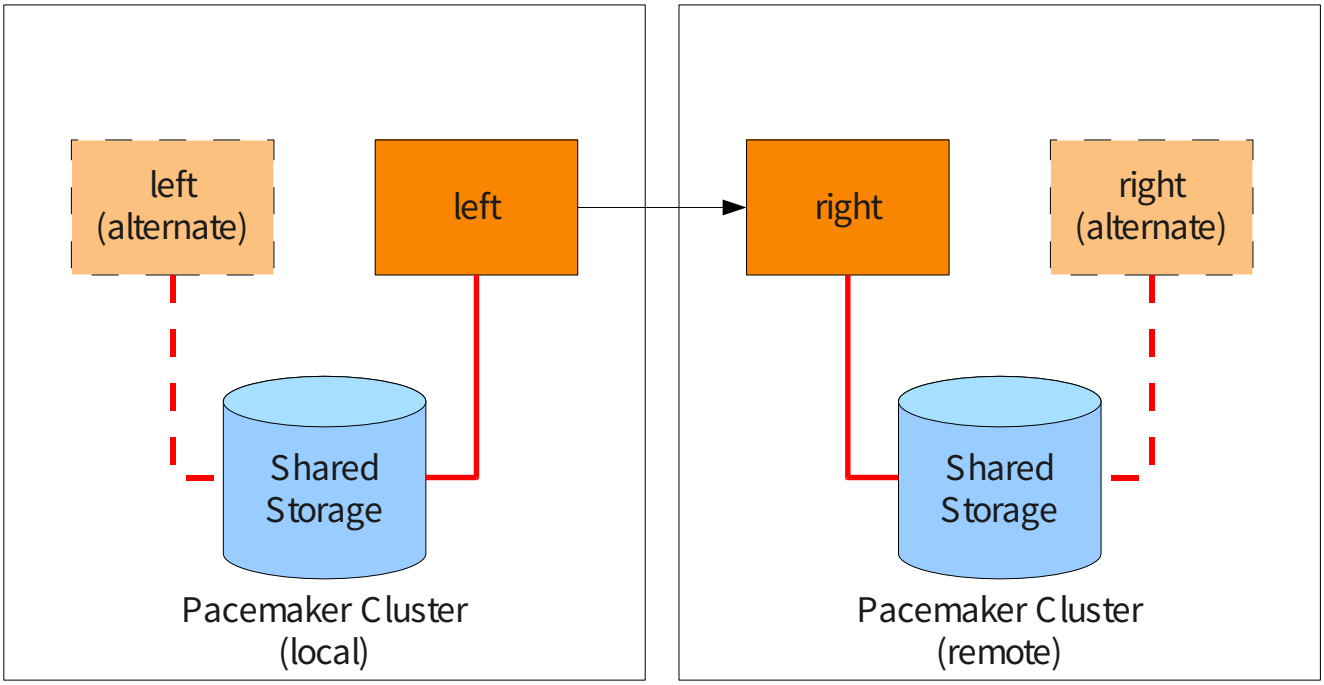


插图 12. 使用DRBD在基于SAN的集群之间进行复制

每个站点中当前充当DRBD对等点的单个节点中的哪一个没有明确定义 - DRBD对等点被称为是 *浮动的*,；也就是说，DRBD绑定到没有绑定到特定物理机的虚拟IP地址。

这种类型的设置通常与DRBD Proxy和/或truck based replication一起部署。

由于这种类型的设置处理共享存储，因此对STONITH进行配置和测试对于它的正常工作至关重要。

7.6.1. DRBD资源配置

要使DRBD资源浮动，请按以下方式在 DRBD.conf 中配置它：

```
resource <resource> {
  ...
  device /dev/drbd0;
  disk /dev/sda1;
  meta-disk internal;
  floating 10.9.9.100:7788;
  floating 10.9.10.101:7788;
}
```

floating 关键字替换通常在资源配置中找到的 on <host> 部分。在这种模式下，DRBD通过IP地址和TCP端口而不是主机名来标识对等点。需要注意的是，指定的地址必须是虚拟集群IP地址，而不是物理节点IP地址，才能使浮点正常工作。如示例所示，在拆分站点配置中，这两个浮动地址可能属于两个独立的IP网络——因此，路由器和防火墙必须正确地允许节点之间的DRBD复制通信。

7.6.2. Pacemaker资源配置

就Pacemaker配置而言，DRBD浮动对等点设置包括以下各项（在涉及的两个Pacemaker集群中的每一个中）：

- 虚拟集群IP地址。
- 主/从DRBD资源（使用DRBD OCF资源代理）。
- Pacemaker约束确保资源以正确的顺序在正确的节点上启动。

要使用复制地址 10.9.9.100 在2节点集群的浮动对等配置中配置名为 mysql 的资源，请使用以下 crm 命令配置Pacemaker：

```
crm configure
crm(live)configure# primitive p_ip_float_left ocf:heartbeat:IPaddr2 \
        params ip=10.9.9.100
crm(live)configure# primitive p_drbd_mysql ocf:linbit:drbd \
        params drbd_resource=mysql
crm(live)configure# ms ms_drbd_mysql drbd_mysql \
        meta master-max="1" master-node-max="1" \
          clone-max="1" clone-node-max="1" \
          notify="true" target-role="Master"
crm(live)configure# order drbd_after_left \
         inf: p_ip_float_left ms_drbd_mysql
crm(live)configure# colocation drbd_on_left \
         inf: ms_drbd_mysql p_ip_float_left
crm(live)configure# commit
bye
```

将此配置添加到CIB后，Pacemaker将执行以下操作：

1. 调出IP地址10.9.9.100（在 alice 或 bob 上）。
2. 根据配置的IP地址调出DRBD资源。
3. 将DRBD资源提升为主要角色。

然后，为了在另一个集群中创建匹配的配置，请使用以下命令配置Pacemaker实例：

```
crm configure
crm(live)configure# primitive p_ip_float_right ocf:heartbeat:IPaddr2 \
        params ip=10.9.10.101
crm(live)configure# primitive drbd_mysql ocf:linbit:drbd \
        params drbd_resource=mysql
crm(live)configure# ms ms_drbd_mysql drbd_mysql \
        meta master-max="1" master-node-max="1" \
          clone-max="1" clone-node-max="1" \
          notify="true" target-role="Slave"
crm(live)configure# order drbd_after_right \
         inf: p_ip_float_right ms_drbd_mysql
crm(live)configure# colocation drbd_on_right
         inf: ms_drbd_mysql p_ip_float_right
crm(live)configure# commit
bye
```

将此配置添加到CIB后，Pacemaker将执行以下操作：

1. 调出IP地址10.9.10.101（在 charlie 或 daisy 上）。
2. 根据配置的IP地址调出DRBD资源。
3. 将DRBD资源保留在次要角色中（由于 `target role="Slave" `）。

7.6.3. 站点故障转移

在拆分站点配置中，可能需要将服务从一个站点传输到另一个站点。这可能是计划过渡的结果，也可能是灾

难性事件的结果。如果转换是正常的预期事件，建议采取以下行动：

- 连接到站点上即将放弃资源的集群，并将受影响的DRBD资源的 target role 属性从 Master 更改为 Slave 。这将根据DRBD资源的主要角色关闭所有资源，将其降级并继续运行，以便从新的主要资源接收更新。
- 连接到站点上即将接管资源的集群，并将受影响的DRBD资源的 target role 属性从 Slave 更改为 Master 。这将提升DRBD资源，根据DRBD资源的主要角色启动任何其他Pacemaker资源，并将更新复制到远程站点。
- 若要退回，只需颠倒程序即可。

在活动站点发生灾难性停机的情况下，可以预期该站点已脱机，不再复制到备份站点。在这种情况下：

- 连接到仍在运行的站点资源上的集群，并将受影响的DRBD资源的 target role 属性从 Slave 更改为 Master 。这将提升DRBD资源，并根据DRBD资源的主要角色启动任何其他Pacemaker资源。
- 还原或重建原始站点时，可以再次连接DRBD资源，然后使用相反的过程进行故障恢复。

7.7. Importing DRBD’s promotion scores into the CIB

Everything described in this section depends on the drbd-attr OCF resource agent. It is available since drbd-utils version 9.15.0. On Debian/Ubuntu systems this is part of the drbd-utils package. On RPM based Linux distributions you need to install the drbd-pacemaker package.

Every DRBD resource exposes a *promotion score* on each node where it is configured. It is a numeric value that might be 0 or positive. The value reflects how desirable it is to promote the resource to master on this particular node. A node that has an *UpToDate* disk and two *UpToDate* replicas has a higher score than a node with an *UpToDate* disk and just one *UpToDate* replica.

During startup, the *promotion score* is 0. E.g., before the DRBD device has its backing device attached, or, if quorum is enabled, before quorum is gained. A value of 0 indicates that a promotion request will fail, and is mapped to a pacemaker score that indicates *must not run here*.

The drbd-attr OCF resource agent imports these promotion scores into node attributes of a Pacemaker cluster. It needs to be configured like this:

```
primitive drbd-attr ocf:linbit:drbd-attr
clone drbd-attr-clone drbd-attr
```

These are *transient* attributes (have a *lifetime* of *reboot* in pacemaker speak). That means, after a *reboot* of the node, or local restart of pacemaker, those attributes will not exist until an instance of drbd-attr is started on that node.

You can inspect the generated attributes with crm_mon -A -1.

These attributed can be used in constraints for services that depend on the DRBD devices, or, when managing DRBD with the ocf:linbit:drbd resource agent, for the *Master* role of that DRBD instance.

Here is an example location constraint for the example resource from 在Pacemaker集群中使用DRBD作为后台服务

```
location lo_fs_mysql fs_mysql \
    rule -inf: not_defined drbd-promotion-score-mysql \
    rule drbd-promotion-score-mysql: defined drbd-promotion-score-mysql
```

This means, that as long as the attribute is not defined, the fs_mysql file system cannot be mounted

here. When the attribute is defined, its value becomes the score of the location constraint.

This can also be used to cause Pacemaker to migrate a service away when DRBD loses a local backing device. Because a failed backing block device causes the promotion score to drop, other nodes with working backing devices will expose higher promotion scores.

The attributes are updated live, independent of the resource-agent's monitor operation, with a dampening delay of 5 seconds by default.

The resource agent has these optional parameters, see also its man page ocf_linbit_drbd-attr(7):

- dampening_delay
- attr_name_prefix
- record_event_details

8. DRBD与红帽集群的集成

本章介绍如何将DRBD用作Red Hat Cluster高可用性集群的复制存储。

本指南使用非正式术语 Red Hat Cluster 指的是在其历史上拥有多个正式产品名称的产品，包括 Red Hat Cluster Suite 和 Red Hat Enterprise Linux高可用性附加组件 。

8.1. 红帽集群背景信息

8.1.1. Fencing

最初主要为共享存储集群设计的Red Hat Cluster依靠节点围栏来防止对共享资源的并发、不协调访问。Red Hat集群围栏基础设施依赖于围栏守护进程 fenced，以及作为shell脚本实现的围栏代理。

尽管基于DRBD的集群不使用共享存储资源，因此从DRBD的角度来看，并不严格要求设置围栏，但即使在基于DRBD的配置中，Red Hat集群套件仍然需要设置围栏。

8.1.2. 资源组管理器

资源组管理器（rgmanager，或者 clurgmgr）类似于Pacemaker。它充当集群管理套件与配置为管理的应用程序的主接口。

Red Hat集群资源

在Red Hat Cluster术语中，一个高可用的应用程序、文件系统、IP地址等被称为 资源 。

在资源相互依赖的地方-例如，NFS导出依赖于正在装载的文件系统-它们形成一个资源树，一种在另一个资源树中嵌套资源的形式。嵌套的内部级别中的资源可以从外部嵌套级别中的资源继承参数。Pacemaker中没有资源树的概念。

红帽集群服务

当资源形成一个共同依赖的集合时，该集合称为一个服务。这与Pacemaker不同，Pacemaker将此类集合称为 资源组 。

rgmanager资源代理

rgmanager 调用的资源代理与Pacemaker使用的资源代理类似，因为它们使用的是在开放集群框架（OCF）中定义的相同的基于shell的API，尽管Pacemaker使用了一些在框架中未定义的扩展。因此，在理论上，资源代理在Red Hat Cluster Suite和Pacemaker之间基本上是可以互换的——然而，实际上，这两个集群管理套件使用不同的资源代理，甚至用于相似或相同的任务。

Red Hat Cluster resource agents安装到 /usr/share/Cluster/ 目录中。与Pacemaker OCF资源代理不同，按照惯例，这些资源代理是自包含的，一些Red Hat集群资源代理被拆分为包含实际SHELL代码的 .sh 文件和包含XML资源代理元数据的 .metadata 文件。

DRBD包括一个Red Hat集群资源代理。它作为 drbd.sh 和 drbd.metadata 安装到习惯目录中。

8.2. Red Hat集群配置

本节概述了运行红帽集群所需的配置步骤。准备集群配置相当简单；一个基于DRBD的Red Hat集群需要两个参与节点（在Red Hat的文档中称为 集群成员 ）和一个围栏设备。

有关配置Red Hat集群的更多信息，请参阅 http://www.Red Hat.com/docs/manuals/csgfs/[Red Hat有关Red Hat集群和GFS的文档。]

8.2.1. cluster.conf 文件

RHEL集群将它们的配置保存在一个配置文件中， /etc/Cluster/Cluster.conf。您可以通过以下方式管理集群配置：

直接编辑配置文件
这是最直接的方法。除了提供文本编辑器之外，它没有任何先决条件。

使用 系统配置集群 GUI
这是一个使用Glade用Python编写的GUI应用程序。它需要X显示的可用性（直接在服务器控制台上，或者通过SSH隧道）。

使用Conga基于web的管理基础设施
Conga基础设施包括一个与本地集群管理器、集群资源管理器和集群LVM守护程序通信的节点代理（ ricci）和一个管理web应用程序（ luci），可以使用简单的web浏览器配置集群基础设施。

8.3. 在Red Hat集群故障转移集群中使用DRBD

本节专门讨论为不涉及GFS的Red Hat集群故障转移集群设置DRBD。有关GFS（和GFS2）配置，请参见将GFS与DRBD结合使用。

本节与将DRBD与Pacemaker集成类似，假设您将使用以下配置参数配置高可用的MySQL数据库：

- 用作数据库存储区的DRBD资源名为mysql，它管理设备 /dev/drbd0。
- DRBD设备拥有一个ext3文件系统，该文件系统将被挂载到 /var/lib/mysql（默认的mysql数据目录）。
- MySQL数据库将利用这个文件系统，监听一个专用的集群IP地址192.168.42.1。

8.3.1. 设置集群配置

要配置高度可用的MySQL数据库，请创建或修改 /etc/cluster/cluster.conf 文件以包含以下配置项。

为此，请使用首选的文本编辑应用程序打开 /etc/cluster/cluster.conf 。然后，在资源配置中包括以下项：

```
<rm>
 <resources />
 <service autostart="1" name="mysql">
  <drbd name="drbd-mysql" resource="mysql">
   <fs device="/dev/drbd/by-res/mysql/0"
      mountpoint="/var/lib/mysql"
      fstype="ext3"
      name="mysql"
      options="noatime"/>
  </drbd>
  <ip address="192.168.42.1" monitor_link="1"/>
  <mysql config_file="/etc/my.cnf"
       listen_address="192.168.42.1"
       name="mysqld"/>
 </service>
</rm>
```

本例假设一个卷资源。

在 <service/> 中相互嵌套资源引用是表示资源依赖关系的Red Hat集群方式。

完成配置后，请确保在根元素 <cluster> 中增加 config_version 属性。然后，发出以下命令将更改提交到正在运行的集群配置：

```
# ccs_tool update /etc/cluster/cluster.conf
# cman_tool version -r <version>
```

在第二个命令中，请确保用新的集群配置版本号替换 *<version>* 。

system config cluster GUI配置实用程序和基于Conga web的集群管理基础设施在将 drbd 资源代理包含在 cluster.conf 文件中之后，都会抱怨您的集群配置。这是由于这两个应用程序提供的Python集群管理包装器的设计，它们不希望对集群基础设施进行第三方扩展。

因此，在集群配置中使用 drbd 资源代理时，不建议使用 system config cluster 或 Conga 进行集群配置。但是，使用这两种工具中的任何一种只监视集群的状态，都可以正常工作。

9. 在DRBD中使用LVM

本章讨论如何结合LVM2管理DRBD。特别是，它包括

- 使用LVM逻辑卷作为DRBD的备份设备；
- 使用DRBD设备作为LVM的物理卷；
- 结合这些概念，使用DRBD实现分层LVM方法。

如果您碰巧不熟悉这些术语，那么LVM基础可以作为您的lvm起点 - 当然，我们始终鼓励您在本节提供的更多细节中熟悉lvm。

9.1. LVM基础

LVM2是Linux device mapper框架上下文中逻辑卷管理的实现。它实际上与原始LVM实现没有任何共同点，除了名称和缩写。旧的实现（现在追溯命名为 "LVM1"）被认为是过时的；本节不涉及它。

在使用LVM时，必须了解其最基本的概念：

物理卷（PV）

PV是由LVM独占管理的底层块设备。pv可以是整个硬盘，也可以是单独的分区。通常的做法是在硬盘上创建一个分区表，其中一个分区专用于Linux LVM的使用。

分区类型 "Linux LVM"（签名 0x8E）可用于标识供LVM独占使用的分区。然而，这并不是必需的 - LVM通过在PV初始化时写入设备的签名来识别PV。

卷组（VG）

VG是LVM的基本管理单元。VG可以包括一个或多个pv。每个VG都有一个唯一的名称。VG可以在运行时通过添加额外的PV或通过扩大现有的PV来扩展。

逻辑卷（LV）

LVs可以在VGs中的运行时创建，并且可以作为常规块设备供内核的其他部分使用。因此，它们可用于保存文件系统，或用于任何其他目的，块设备可用于。LVs可以在联机时调整大小，也可以从一个PV移动到另一个PV（只要PV是同一VG的一部分）。

快照逻辑卷（SLV）

快照是LVs的临时时间点副本。创建快照是一个几乎立即完成的操作，即使原始LV（原始卷）的大小为几百吉比特。通常，快照需要的空间比原始LV少得多。

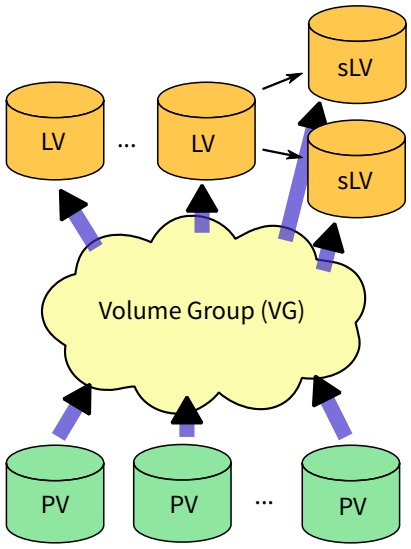


插图 13. LVM概述

9.2. 使用逻辑卷作为DRBD备份设备

要以这种方式使用LV，只需创建它们，然后像通常那样为DRBD初始化它们。

此示例假设在启用LVM的系统上的两个节点上都已存在名为 foo 的卷组，并且您希望使用该卷组中的逻辑卷创建名为 r0 的DRBD资源。

首先，创建逻辑卷：

```
# lvcreate --name bar --size 10G foo
Logical volume "bar" created
```

当然，您必须在DRBD集群的两个节点上完成此命令。之后，在任一节点上都应该有一个名为 /dev/foo/bar 的块设备。

然后，只需在资源配置中输入新创建的卷：

```
resource r0 {
  ...
  on alice {
    device /dev/drbd0;
    disk   /dev/foo/bar;
    ...
  }
  on bob {
    device /dev/drbd0;
    disk   /dev/foo/bar;
    ...
  }
}
```

现在您可以continue to bring your resource up，就像使用非LVM块设备一样。

9.3. 在DRBD同步期间使用自动LVM快照

当DRBD正在同步时，同步目标的状态是不一致的，直到同步完成。如果在这种情况下，SyncSource发生故障（无法修复），这将使您处于一个不幸的位置：具有良好数据的节点已死亡，而幸存的节点具有不良（不一致）数据。

当从LVM逻辑卷上服务DRBD时，可以通过在同步启动时创建自动快照，并在同步成功完成后自动删除同一快照来缓解此问题。

为了在重新同步期间启用自动快照，请在资源配置中添加以下行：

Listing 6. 在DRBD同步之前自动生成快照

```
resource r0 {
  handlers {
    before-resync-target "/usr/lib/drbd/snapshot-resync-target-lvm.sh";
    after-resync-target "/usr/lib/drbd/unsnapshot-resync-target-lvm.sh";
  }
}
```

这两个脚本解析DRBD自动传递给它调用的任何 handler 的 $DRBD_RESOURCE 环境变量。然后，

snapshot-resync-target-lvm.sh
脚本为资源包含的任何卷创建一个lvm快照，然后开始同步。如果脚本失败，则同步不会开始。

同步完成后， unsnapshot-resync-target-lvm.sh 脚本将删除不再需要的快照。如果取消快照失败，快照将继续徘徊。

你应该尽快检查悬挂的快照。完整快照会导致快照本身及其原始卷失败。

如果您的同步源在任何时候确实出现无法修复的故障，并且您决定还原到对等机上的最新快照，则可以通过输入 lvconvert -M 命令来执行此操作。

9.4. 将DRBD资源配置为物理卷

。为此，请在资源当前处于主要角色的节点上发出以下命令之一：

```
# pvcreate /dev/drbdX
```

或

```
# pvcreate /dev/drbd/by-res/<resource>/0
```

本例假设一个含有单个卷的资源。

现在，有必要将这个设备包含在LVM扫描PV签名的设备列表中。为此，必须编辑LVM配置文件，通常命名为 /etc/LVM/LVM.conf。在 devices 部分找到包含 filter 关键字的行，并相应地对其进行编辑。如果要将所有pv存储在DRBD设备上，则以下是适当的 filter 选项：

```
filter = [ "a|drbd.*|", "r|.*|" ]
```

此筛选器表达式接受在任何DRBD设备上找到的PV签名，同时拒绝（忽略）所有其他签名。

默认情况下，LVM扫描在 /dev 中找到的所有块设备以获取PV签名。这相当于 filter=["a|.*|"]。

如果要将堆叠资源用作LVM pv，则需要更明确的筛选器配置。您需要确保LVM在堆栈资源上检测到PV签名，而在相应的低级资源和备份设备上忽略它们。此示例假设较低级别的DRBD资源使用设备子级0到9，而堆叠的资源使用的设备子级从10到9：

```
filter = [ "a|drbd1[0-9]|", "r|.*|" ]
```

此筛选器表达式接受仅在DRBD设备 /dev/drbd10 到 `/dev/drbd19' 上找到的PV签名，同时拒绝（忽略）所有其他签名。

修改 lvm.conf 文件后，必须运行 vgscan 命令，以便lvm放弃其配置缓存并重新扫描设备以获取PV签名。

当然，您可以使用不同的 filter 配置来匹配您的特定系统配置。然而，重要的是要记住，你需要

- 接受（包括）您希望用作pv的DRBD设备；
- 拒绝（排除）相应的低级设备，以避免LVM发现重复的PV签名。

此外，应通过设置以下内容禁用LVM缓存：

```
write_cache_state = 0
```

禁用LVM缓存后，请确保通过删除 /etc/LVM/cache/.cache ，删除所有过时的缓存条目。

您还必须在对等节点上重复上述步骤。

如果系统的根文件系统位于LVM上，则卷组将在引导期间从初始ramdisk（initrd）激活。这样，LVM工具将计算initrd映像中包含的 LVM.conf 文件。因此，在对 lvm.conf 进行任何更改之后，您应该确定使用适合您的发行版的实用程序（ mkinitrd 、update-initramfs 等）更新initrd。

配置新的PV后，可以继续将其添加到卷组，或从中创建新的卷组。当然，DRBD资源在执行此操作时必须处于主要角色。

```
# vgcreate <name> /dev/drbdX
```

虽然可以在同一个卷组中混合DRBD和非DRBD物理卷，但不建议这样做，也不太可能有任何实际价值。

创建VG后，可以使用 lvcreate 命令（与非DRBD支持的卷组一样）开始从中切分逻辑卷。

9.5. 将新的DRBD卷添加到现有卷组

有时，您可能希望向卷组中添加新的DRBD支持的物理卷。无论何时执行此操作，都应将新卷添加到现有资源配置中。这将保留复制流并确保VG中所有pv的写入保真度。

如果您的LVM卷组由Pacemaker管理，如带Pacemaker的高可用LVM中所述，则在更改DRBD配置之前，必须将集群置于维护模式。

扩展资源配置以包括附加卷，如下例所示：

```
resource r0 {
  volume 0 {
    device    /dev/drbd1;
    disk      /dev/sda7;
    meta-disk internal;
  }
  volume 1 {
    device    /dev/drbd2;
    disk      /dev/sda8;
    meta-disk internal;
  }
  on alice {
    address   10.1.1.31:7789;
  }
  on bob {
    address   10.1.1.32:7789;
  }
}
```

确保您的DRBD配置在节点间是相同的，然后输入：

```
# drbdadm adjust r0
```

这将隐式调用 drbdsetup new-minor r0 1，以启用资源 r0 中的新卷 1。将新卷添加到复制流后，可以初始化并将其添加到卷组：

```
# pvcreate /dev/drbd/by-res/<resource>/1
# vgextend <name> /dev/drbd/by-res/<resource>/1
```

这将把新的PV /dev/drbd/by res/<resource>/1 添加到 <name> VG中，从而在整个VG中保持写保真度。

9.6. 带DRBD的嵌套LVM配置

如果稍微高级一点，可以同时使用 Logical Volumes作为DRBD的备份设备，同时使用DRBD设备本身作为 Physical Volume。要提供示例，请考虑以下配置：

- 我们有两个分区，名为 /dev/sda1 ，和 /dev/sdb1 ，打算用作物理卷。
- 这两个pv都将成为名为 local 的卷组的一部分。
- 我们想在这个VG中创建一个10 GiB的逻辑卷，名为 r0。
- 这个LV将成为DRBD资源的本地备份设备，也称为 r0，它对应于设备 /dev/drbd0。
- 此设备将是另一个名为 replicated 的卷组的唯一PV。
- 这个VG将包含另外两个名为 foo（4 GiB）和 bar（6 GiB）的逻辑卷。

要启用此配置，请执行以下步骤：

- 在 /etc/lvm/lvm.conf 中设置适当的 filter 选项：

  ```
  filter = ["a|sd.*|", "a|drbd.*|", "r|.*|"]
  ```

  这个过滤器表达式接受在任何SCSI和DRBD设备上找到的PV签名，同时拒绝（忽略）所有其他的。

  修改 lvm.conf 文件后，必须运行 vgscan 命令，以便lvm放弃其配置缓存并重新扫描设备以获取PV签名。

- 通过设置禁用LVM缓存：

  ```
  write_cache_state = 0
  ```

  禁用LVM缓存后，请确保通过删除 /etc/LVM/cache/.cache ，删除所有过时的缓存条目。

- 现在，您可以将两个SCSI分区初始化为PVs:

  ```
  # pvcreate /dev/sda1
  Physical volume "/dev/sda1" successfully created
  # pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
  ```

- 下一步是创建名为 local 的低级VG，它由刚刚初始化的两个pv组成：

```
# vgcreate local /dev/sda1 /dev/sda2
Volume group "local" successfully created
```

- 现在您可以创建用作DRBD的备份设备的逻辑卷：

```
# lvcreate --name r0 --size 10G local
Logical volume "r0" created
```

- 在对等节点上重复所有步骤，直到现在。
- 然后，编辑 /etc/drbd.conf 以创建名为 r0 的新资源：

```
resource r0 {
  device /dev/drbd0;
  disk /dev/local/r0;
  meta-disk internal;
  on <host> { address <address>:<port>; }
  on <host> { address <address>:<port>; }
}
```

创建新的资源配置后，请确保将 drbd.conf 内容复制到对等节点。

- 在此之后，按首次启用资源中所述初始化资源（在两个节点上）。
- 然后，提升资源（在一个节点上）：

```
# drbdadm primary r0
```

- 现在，在刚刚升级资源的节点上，将DRBD设备初始化为一个新的物理卷：

```
# pvcreate /dev/drbd0
Physical volume "/dev/drbd0" successfully created
```

- 使用刚刚初始化的PV在同一节点上创建名为 replicated 的VG:

```
# vgcreate replicated /dev/drbd0
Volume group "replicated" successfully created
```

- 最后，通过

```
# lvcreate --name foo --size 4G replicated
Logical volume "foo" created
# lvcreate --name bar --size 6G replicated
Logical volume "bar" created
```

逻辑卷 foo 和 bar 现在可以作为本地节点上的 /dev/replicated/foo 和 /dev/replicated/bar 使用。

9.6.1. 将VG切换到另一个节点

要使它们在另一个节点上可用，请首先在主节点上发出以下命令序列：

```
# vgchange -a n replicated
0 logical volume(s) in volume group "replicated" now active
# drbdadm secondary r0
```

然后，在另一个（仍然是辅助）节点上发出以下命令：

```
# drbdadm primary r0
# vgchange -a y replicated
2 logical volume(s) in volume group "replicated" now active
```

之后，块设备 /dev/replicated/foo 和 /dev/replicated/bar 将在另一个（现在是主）节点上可用。

9.7. 带Pacemaker的高可用LVM

在对等机之间传输卷组并使相应的逻辑卷可用的过程可以自动化。PacemakerLVM资源代理正是为此目的而设计的。

要将现有的、由DRBD支持的卷组置于Pacemaker管理下，请在 crm shell中运行以下命令：

Listing 7. 支持DRBD的LVM卷组的Pacemaker配置

```
primitive p_drbd_r0 ocf:linbit:drbd \
  params drbd_resource="r0" \
  op monitor interval="29s" role="Master" \
  op monitor interval="31s" role="Slave"
ms ms_drbd_r0 p_drbd_r0 \
  meta master-max="1" master-node-max="1" \
     clone-max="2" clone-node-max="1" \
     notify="true"
primitive p_lvm_r0 ocf:heartbeat:LVM \
  params volgrpname="r0"
colocation c_lvm_on_drbd inf: p_lvm_r0 ms_drbd_r0:Master
order o_drbd_before_lvm inf: ms_drbd_r0:promote p_lvm_r0:start
commit
```

提交此配置后，Pacemaker将自动使 r0 卷组在当前具有DRBD资源主（Master）角色的节点上可用。

10. 将GFS与DRBD结合使用

本章概述了将DRBD资源设置为包含共享的Global File System(GFS)的块设备所需的步骤。它包括GFS和GFS2。

要在DRBD上使用GFS，必须在indexterm中配置DRBD:[dual-primary mode]dual-primary mode。

所有的集群文件系统都需要围栏-不仅通过DRBD资源，而且还需要STONITH！必须kill掉有问题的成员。

您需要这些设置：

```
net {
   fencing resource-and-stonith;
}
handlers {
   # Make sure the other node is confirmed
   # dead after this!
   outdate-peer "/sbin/kill-other-node.sh";
}
```

一定有不稳定的缓存！有关更多信息，请参见 https://fedorahosted.org/cluster/wiki/DRBD_Cookbook。

10.1. GFS基础

Red Hat全局文件系统（GFS）是redhat对并发访问共享存储文件系统的实现。与任何此类文件系统一样，GFS允许多个节点以读/写方式同时访问同一存储设备，而不会造成数据损坏的风险。它通过使用分布式锁管理器（DLM）来管理来自集群成员的并发访问。

GFS从一开始就被设计用于传统的共享存储设备。无论如何，完全有可能在dual-primary模式下使用DRBD作为GFS的复制存储设备。由于DRBD通常从本地存储器读取和写入数据，而GFS通常配置为从SAN设备运行，因此应用程序可能会从减少的读/写延迟中受益。当然，DRBD还为每个GFS文件系统添加了一个额外的物理拷贝，从而为这个概念增加了冗余。

GFS使用LVM的集群感知变量 称为集群逻辑卷管理器或 CLVM。因此，在使用DRBD作为GFS的数据存储和使用<s-lvm-DRBD-As-pv，DRBD作为常规lvm的物理卷之间存在一些并行性。

GFS文件系统通常与Red Hat自己的集群管理框架 Red Hat Cluster紧密集成。本章解释在红帽集群上下文中DRBD与GFS的结合使用。

GFS、CLVM和Red Hat Cluster在Red Hat Enterprise Linux（RHEL）及其派生的发行版中可用，例如 CentOS和 Debian GNU/Linux中也提供了从相同来源构建的包。本章假设在Red Hat Enterprise Linux系统上运行GFS。

10.2. 创建适合GFS的DRBD资源

由于GFS是一个共享的集群文件系统，需要从所有集群节点进行并发读/写存储访问，因此用于存储GFS文件系统的任何DRBD资源都必须在dual-primary mode中配置。此外，建议使用一些DRBD的features for automatic recovery from split brain。为此，在资源配置中包括以下行：

```
resource <resource> {
  net {
    allow-two-primaries yes;
    after-sb-0pri discard-zero-changes;
    after-sb-1pri discard-secondary;
    after-sb-2pri disconnect;
    ...
  }
  ...
}
```

通过配置自动恢复策略，您可以有效地配置自动数据丢失！一定要明白其中的含义。

一旦您将这些选项添加到your freshly-configured resource中，您就可以initialize your resource as you normally would, 提升为两个节点上的主角色。

10.3. 配置LVM以识别DRBD资源

GFS使用CLVM（LVM的集群感知版本）来管理GFS要使用的块设备。为了在DRBD中使用CLVM，请确保LVM配置

- 使用集群锁定。为此，请在 /etc/lvm/lvm.conf 中设置以下选项：

```
locking_type = 3
```

- 扫描DRBD设备以识别基于DRBD的物理卷（pv）。这适用于传统（非集群）LVM；有关详细信息，请参见将DRBD资源配置为物理卷。

10.4. 配置集群以支持GFS

创建新的DRBD资源并 completed your initial cluster configurations后，必须在GFS集群的两个节点上启用并启动以下系统服务：

- cman（这也会启动 ccsd 和 fenced），
- clvmd.

10.5. 创建GFS文件系统

为了在dual-primary DRBD资源上创建GFS文件系统，必须首先将其初始化为Logical Volume for LVM。

与传统的、非集群感知的LVM配置相反，由于CLVM的集群感知特性，只能在一个节点上完成以下步骤：

```
# pvcreate /dev/drbd/by-res/<resource>/0
Physical volume "/dev/drbd<num>" successfully created
# vgcreate <vg-name> /dev/drbd/by-res/<resource>/0
Volume group "<vg-name>" successfully created
# lvcreate --size <size> --name <lv-name> <vg-name>
Logical volume "<lv-name>" created
```

本例假设一个单个卷资源。

CLVM将立即通知对等节点这些更改；]在对等节点上发出 lvs（或 lvdisplay ）将列出新创建的逻辑卷。

现在，您可以通过创建实际的文件系统来继续：

```
# mkfs -t gfs -p lock_dlm -j 2 /dev/<vg-name>/<lv-name>
```

或者，对于GFS2文件系统：

```
# mkfs -t gfs2 -p lock_dlm -j 2 -t <cluster>:<name>
  /dev/<vg-name>/<lv-name>
```

此命令中的 -j 选项指为GFS保留的日志数。这必须与GFS集群中具有并发主节点角色的节点数相同；因为在DRBD 9.1之前，DRBD不支持超过两个主节点，所以此处设置的值始终为2。

-t 选项仅适用于GFS2文件系统，它定义了锁表名。这遵循格式 *cluster>:<name>* ，其中 *<cluster>* 必须与 /etc/cluster/cluster.conf 中定义的集群名称匹配。因此，只允许该集群的成员使用文件系统。相比之下，集群中的任意文件系统名是唯一的。

10.6. 使用GFS文件系统

创建文件系统后，可以将其添加到 /etc/fstab ：

```
/dev/<vg-name>/<lv-name> <mountpoint> gfs defaults 0 0
```

对于GFS2文件系统，只需更改文件系统类型：

```
/dev/<vg-name>/<lv-name> <mountpoint> gfs2 defaults 0 0
```

不要忘记在两个（或者，对于DRBD 9.1，all）集群节点上进行此更改。

在此之后，您可以启动 gfs 服务（在两个节点上）来挂载新的文件系统：

```
# service gfs start
```

从那时起，只要您将DRBD配置为在系统启动时自动启动，在RHCS服务和 gfs 服务之前，您就可以使用这个gfs文件系统，就像使用在传统共享存储上配置的文件系统一样。

11. 在DRBD中使用OCFS2

本章概述了将DRBD资源设置为包含共享Oracle Cluster文件系统版本2（OCFS2）的块设备所需的步骤。

所有的群集文件系统都需要围栏-不仅通过DRBD资源，而且还需要STONITH！必须kill掉有问题的成员。

您需要这些设置：

```
net {
  fencing resource-and-stonith;
}
handlers {
  # Make sure the other node is confirmed
  # dead after this!
  outdate-peer "/sbin/kill-other-node.sh";
}
```

一定有不稳定的缓存！你可能会在https://fedorahosted.org/cluster/wiki/DRBD_Cookbook 上得到一些提示，尽管那是关于GFS2的，而不是OCFS2的。

11.1. OCFS2基础

Oracle集群文件系统（OCFS2）是Oracle公司开发的一个并发访问共享存储文件系统。与它的前身OCFS不同，OCFS是专门设计的，只适用于Oracle数据库有效负载，OCFS2是实现大多数POSIX语义的通用文件系统。OCFS2最常见的用例可以说是Oracle Real Application Cluster（RAC），但是OCFS2也可以用于实现比如负载平衡的NFS集群。

虽然OCFS2最初设计用于传统的共享存储设备，但它同样非常适合部署在dual-Primary DRBD上。从文件系统中读取的应用程序可能会受益于减少的读取延迟，这是因为DRBD从本地存储中读取和写入，而不是在正常情况下运行的SAN设备OCFS2。此外，DRBD通过向每个文件系统映像添加一个额外的副本来增加OCFS2的冗余，而不是仅仅共享一个文件系统映像。

与其他共享群集文件系统（如GFS）一样，OCFS2允许多个节点以读/写模式同时访问同一存储设备，而不会导致数据损坏。它通过使用分布式锁管理器（DLM）来管理来自集群节点的并发访问。DLM本身使用一个虚拟文件系统（ocfs2_dlmfs），它独立于系统上的实际ocfs2文件系统。

OCFS2可以使用一个内在的集群通信层来管理集群成员和文件系统的装载和卸载操作，或者将这些任务推迟到Pacemaker集群基础设施。

OCFS2在SUSE Linux企业服务器（它是主要受支持的共享集群文件系统）、CentOS、Debian GNU/Linux和Ubuntu服务器版本中可用。Oracle还为Red Hat Enterprise Linux（RHEL）提供了软件包。本章假设在SUSE Linux企业服务器系统上运行OCFS2。

11.2. 创建适合OCFS2的DRBD资源

由于OCFS2是一个共享的群集文件系统，需要从所有群集节点进行并发读/写存储访问，因此用于存储OCFS2文件系统的任何DRBD资源都必须在dual-primary mode中配置。此外，建议使用一些DRBD的features for automatic recovery from split brain。为此，在资源配置中包括以下行：

```
resource <resource> {
 net {
  # allow-two-primaries yes;
  after-sb-0pri discard-zero-changes;
  after-sb-1pri discard-secondary;
  after-sb-2pri disconnect;
  ...
 }
 ...
}
```

通过设置自动恢复策略，您可以有效地配置自动数据丢失！一定要明白其中的含义。

不建议在初始配置时将 allow-two-primaries 选项设置为 yes 。您应该在初始资源同步完成后执行此操作。

一旦您将这些选项添加到your freshly-configured resource中，您就可以initialize your resource as you normally would. 当您设置了 allow-two-primaries 选项为 yes ，你就可以promote the resource将两节点角色都提升为主要角色。

在DRBD 9.1中，可能有两个以上的节点处于主要角色；在DRBD 9.0中，只能使用两个主要角色，但可以有更多的节点处于次要角色以提供更多冗余。

11.3. 创建OCFS2文件系统

Now, use OCFS2’ s mkfs implementation to create the file system:

```
# mkfs -t ocfs2 -N 2 -L ocfs2_drbd0 /dev/drbd0
mkfs.ocfs2 1.4.0
Filesystem label=ocfs2_drbd0
Block size=1024 (bits=10)
Cluster size=4096 (bits=12)
Volume size=205586432 (50192 clusters) (200768 blocks)
7 cluster groups (tail covers 4112 clusters, rest cover 7680 clusters)
Journal size=4194304
Initial number of node slots: 2
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 0 block(s)
Formatting Journals: done
Writing lost+found: done
mkfs.ocfs2 successful
```

这将在 /dev/drbd0 上创建一个具有两个节点插槽的OCFS2文件系统，并将文件系统标签设置为 ocfs2_drbd0。您可以在 mkfs 调用中指定其他选项；有关详细信息，请参阅 mkfs.ocfs2 系统手册页。

11.4. Pacemaker OCFS2管理

11.4.1. 向Pacemaker添加Dual-Primary DRBD资源

现有的Dual-Primary DRBD resource可通过以下 crm 配置添加到Pacemaker资源管理：

```
primitive p_drbd_ocfs2 ocf:linbit:drbd \
  params drbd_resource="ocfs2"
ms ms_drbd_ocfs2 p_drbd_ocfs2 \
  meta master-max=2 clone-max=2 notify=true
```

注意 master max=2 元变量；它为Pacemaker 主/从设备启用双主模式。这要求在DRBD配置中将 allow-two-primaries 也设置为 yes。否则，Pacemaker将在资源验证期间标记配置错误。

11.4.2. 为Pacemaker添加OCFS2管理功能

为了管理OCFS2和内核分布式锁管理器（DLM），Pacemaker总共使用三种不同的资源代理：

- ocf:pacemaker:controld—Pacemaker与DLM的接口;
- ocf:ocfs2:o2cb--Pacemaker与ocfs2集群管理的接口;
- ocf:heartbeat:Filesystem--通用文件系统管理资源代理，配置为Pacemaker克隆时支持群集文件系统。

您可以通过创建具有以下“crm”配置的克隆资源组，为OCFS2管理启用Pacemaker群集中的所有节点：

```
primitive p_controld ocf:pacemaker:controld
primitive p_o2cb ocf:ocfs2:o2cb
group g_ocfs2mgmt p_controld p_o2cb
clone cl_ocfs2mgmt g_ocfs2mgmt meta interleave=true
```

提交此配置后，Pacemaker将在群集中的所有节点上启动“controld”和“o2cb”资源类型的实例。

11.4.3. 向Pacemaker添加OCFS2文件系统

Pacemaker使用传统的“ocf:heartbeat:Filesystem”资源代理管理OCFS2文件系统，尽管处于克隆模式。要将OCFS2文件系统置于Pacemaker管理下，请使用以下“crm”配置：

```
primitive p_fs_ocfs2 ocf:heartbeat:Filesystem \
  params device="/dev/drbd/by-res/ocfs2/0" directory="/srv/ocfs2" \
      fstype="ocfs2" options="rw,noatime"
clone cl_fs_ocfs2 p_fs_ocfs2
```

本例假设一个卷资源。

11.4.4. 添加所需的Pacemaker约束以管理OCFS2文件系统

In order to tie all OCFS2-related resources and clones together, add the following constraints to your Pacemaker configuration:

```
order o_ocfs2 ms_drbd_ocfs2:promote cl_ocfs2mgmt:start cl_fs_ocfs2:start
colocation c_ocfs2 cl_fs_ocfs2 cl_ocfs2mgmt ms_drbd_ocfs2:Master
```

11.5. 传统OCFS2管理（不带Pacemaker）

本节介绍的信息适用于Pacemaker中不支持OCFS2 DLM的传统系统。此处仅作参考之用。新装置应始终使用Pacemaker方法。

11.5.1. 配置群集以支持OCFS2

创建配置文件

OCFS2使用一个中央配置文件 /etc/OCFS2/cluster.conf。

创建OCFS2集群时，请确保将两个主机都添加到集群配置中。默认端口（7777）通常是群集互连通信的可接受选择。如果您选择任何其他端口号，请确保选择的端口号与DRBD使用的现有端口（或任何其他配置的TCP/IP）不冲突。

如果您觉得直接编辑cluster.conf文件不太舒服，还可以使用通常更方便的 ocfs2console 图形配置实用程序。不管您选择什么方法，您的 /etc/ocfs2/cluster.conf 文件内容应该大致如下：

```
node:
  ip_port = 7777
  ip_address = 10.1.1.31
  number = 0
  name = alice
  cluster = ocfs2

node:
  ip_port = 7777
  ip_address = 10.1.1.32
  number = 1
  name = bob
  cluster = ocfs2

cluster:
  node_count = 2
  name = ocfs2
```

配置完群集配置后，请使用 scp 将配置分发到群集中的两个节点。

配置O2CB驱动程序

SUSE Linux Enterprise systems

在SLES上，可以使用 o2cb init脚本的 configure 选项：

```
# /etc/init.d/o2cb configure
Configuring the O2CB driver.

This will configure the on-boot properties of the O2CB driver.
The following questions will determine whether the driver is loaded on
boot.  The current values will be shown in brackets ('[]').  Hitting
<ENTER> without typing an answer will keep that current value.  Ctrl-C
will abort.

Load O2CB driver on boot (y/n) [y]:
Cluster to start on boot (Enter "none" to clear) [ocfs2]:
Specify heartbeat dead threshold (>=7) [31]:
Specify network idle timeout in ms (>=5000) [30000]:
Specify network keepalive delay in ms (>=1000) [2000]:
Specify network reconnect delay in ms (>=2000) [2000]:
Use user-space driven heartbeat? (y/n) [n]:
Writing O2CB configuration: OK
Loading module "configfs": OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading module "ocfs2_nodemanager": OK
Loading module "ocfs2_dlm": OK
Loading module "ocfs2_dlmfs": OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Starting O2CB cluster ocfs2: OK
```

.Debian上的Debian GNU/Linux系统， /etc/init.d/o2cb **的** configure

选项不可用。请重新配置 ocfs2-tools 包以启用驱动程序：

```
# dpkg-reconfigure -p medium -f readline ocfs2-tools
Configuring ocfs2-tools
Would you like to start an OCFS2 cluster (O2CB) at boot time? yes
Name of the cluster to start at boot time: ocfs2
The O2CB heartbeat threshold sets up the maximum time in seconds that a node
awaits for an I/O operation. After it, the node "fences" itself, and you will
probably see a crash.

It is calculated as the result of: (threshold - 1) x 2.

Its default value is 31 (60 seconds).

Raise it if you have slow disks and/or crashes with kernel messages like:

o2hb_write_timeout: 164 ERROR: heartbeat write timeout to device XXXX after NNNN
milliseconds
O2CB Heartbeat threshold: `31`
      Loading filesystem "configfs": OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading stack plugin "o2cb": OK
Loading filesystem "ocfs2_dlmfs": OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Setting cluster stack "o2cb": OK
Starting O2CB cluster ocfs2: OK
```

11.5.2. 使用OCFS2文件系统

完成集群配置并创建文件系统后，可以将其装载为任何其他文件系统：

```
# mount -t ocfs2 /dev/drbd0 /shared
```

然后，内核日志（通过发出命令 dmesg 可以访问）应该包含一行类似的内容：

```
ocfs2: Mounting device (147,0) on (node 0, slot 0) with ordered data mode.
```

从此时起，您应该能够以读/写模式同时在两个节点上挂载OCFS2文件系统。

12. 在DRBD中使用Xen

本章概述了将DRBD用作虚拟块设备（VBD）的方法，用于使用Xen虚拟机监控程序的虚拟化环境。

12.1. Xen基础

Xen是一个虚拟化框架，最初在剑桥大学（英国）开发，后来由XenSource，Inc.维护（现在是Citrix的一部分）。它包含在大多数Linux发行版的最新版本中，如Debian GNU/Linux（4.0版以后）、SUSE Linux Enterprise Server（10版以后）、Red Hat Enterprise Linux（5版以后）和许多其他发行版。

Xen使用半虚拟化（一种虚拟化方法，涉及虚拟化主机和来宾虚拟机之间的高度协作）和选定的来宾操作系统，与传统的虚拟化解决方案（通常基于硬件仿真）相比，可提高性能。Xen还支持在支持适当虚拟化扩展的cpu上进行完全硬件仿真；用Xen的话说，这就是HVM（"硬件辅助虚拟机"）。

在撰写本文时，Xen for HVM支持的CPU扩展是英特尔的虚拟化技术（VT，以前的代号为"Vanderpool"）和AMD的安全虚拟机（SVM，以前的代号为"Pacifica"）。

Xen支持 *live migration*，它是指在不中断的情况下，将正在运行的来宾操作系统从一个物理主机传输到另一个物理主机的能力。

当DRBD资源用作Xen的复制虚拟块设备（VBD）时，它用于使domU虚拟磁盘的全部内容在两台服务器上可用，然后可以将其配置为自动故障转移。这样，DRBD不仅为Linux服务器提供冗余（在非虚拟化DRBD部署场景中），而且也为可以在Xen下虚拟化的任何其他操作系统提供冗余，Xen实际上包括32位或64位Intel兼容体系结构上可用的任何操作系统。

12.2. 设置DRBD模块参数以用于Xen

对于Xen Domain-0内核，建议加载DRBD模块，并将参数 disable_sendpage 设置为 1。为此，创建（或打开）文件 /etc/modprobe.d/drbd.conf，并输入以下行：

```
options drbd disable_sendpage=1
```

12.3. 创建适合用作Xen VBD的DRBD资源

配置将用作Xen虚拟块设备的DRBD资源相当简单 - 实际上，典型配置与用于任何其他目的的DRBD资源的配置相匹配。但是，如果要为来宾实例启用实时迁移，则需要为此资源启用 dual-primary mode：

```
resource <resource> {
 net {
  allow-two-primaries yes;
  ...
 }
 ...
}
```

启用dual-primary模式是必要的，因为Xen在启动实时迁移之前，会检查资源配置为在源主机和目标主机上用于迁移的所有VBD上的写访问。

12.4. 使用DRBD VBDs

为了将DRBD资源用作虚拟块设备，必须在Xen domU配置中添加如下行：

```
disk = [ 'drbd:<resource>,xvda,w' ]
```

此示例配置使名为 *resource* 的DRBD资源在读/写模式（w）下以 /dev/xvda 的形式对domU可用。

当然，您可以将多个DRBD资源与一个domU一起使用。在这种情况下，只需在 disk 选项中添加更多类似于示例中提供的条目，并用逗号分隔。

在以下三种情况下，您不能使用此方法：

- 您正在配置完全虚拟化（HVM）domU。
- 您正在使用图形安装实用程序安装domU，图形安装程序不支持 drbd: 语法。
- 您正在配置一个没有 kernel、 initrd 和 extra 选项的domU，依赖 bootloader 和 bootloader_args 参数来使用Xen pseudo-bootloader，该pseudo-bootloader不支持 drbd: 语法。
  - pygrub+（在Xen 3.3之前）和 domUloader.py（在SUSE Linux企业服务器10上随Xen一起提供）是两个不支持 drbd: 虚拟块设备配置语法的伪引导加载程序示例。
  - 自Xen 3.3 以后的 pygrub 和SLES 11后附带的 domUloader.py 版本都支持这种语法。

在这些情况下，必须使用传统的 phy: 设备语法和与资源关联的DRBD设备名，而不是资源名。但是，这要求您在Xen之外管理DRBD状态转换，这是一种比 DRBD 资源类型提供的更不灵活的方法。

12.5. 启动、停止和迁移DRBD支持的domU

启动domU
一旦配置了DRBD支持的domU，就可以像启动其他domU一样启动它：

```
# xm create <domU>
Using config file "/etc/xen/<domU>".
Started domain <domU>
```

在此过程中，您配置为VBD的DRBD资源将提升为主要角色，并按预期让Xen访问。

停止domU
这同样直截了当：

```
# xm shutdown -w <domU>
Domain <domU> terminated.
```

同样，如您所料，在domU成功关闭后，DRBD资源将返回到次要角色。

迁移domU
这也是使用常用的Xen工具完成的：

```
# xm migrate --live <domU> <destination-host>
```

在这种情况下，会自动连续快速地执行几个管理步骤: * 资源将提升为目标主机上的主要角色。 * 本地主机上启动了 *domU* 的实时迁移。 * 迁移到目标主机完成后，资源将在本地降级为次要角色。

两个资源必须在两台主机上以主角色短暂运行，这是首先必须以dual-primary模式配置资源的原因。

12.6. DRBD/Xen集成的内部机制

Xen本机支持两种虚拟块设备类型：

phy
此设备类型用于将主机环境中可用的 "physical" 块设备以基本透明的方式移交给来宾domU。

file
此设备类型用于使基于文件的块设备映像对来宾domU可用。它的工作原理是从原始图像文件创建一个循环块设备，然后以与 phy 设备类型几乎相同的方式将该块设备传递给domU。

如果在domU配置的 disk 选项中配置的虚拟块设备使用除 phy: 、 file: 之外的任何前缀，或者根本不使用前缀（在这种情况下，Xen默认使用 phy 设备类型），Xen希望在Xen scripts目录（通常是 /etc/Xen/scripts ）中找到名为 block- *prefix* 的助手脚本。

DRBD发行版为 drbd 设备类型提供了这样一个脚本，名为 /etc/xen/scripts/block-drbd 。该脚本处理必要的DRBD资源状态转换，如本章前面所述。

12.7. Xen与Pacemaker的集成

In order to fully capitalize on the benefits provided by having a DRBD-backed Xen VBD’s, it is recommended to have Pacemaker manage the associated domU’s as Pacemaker resources.

您可以将Xen domU配置为Pacemaker资源，并自动进行资源故障转移。为此，请使用Xen OCF资源代理。如果您使用本章中描述的 drbd Xen设备类型，则不需要配置任何单独的 drbd 资源以供Xen群集资源使用。相反，block-drbd 助手脚本将为您执行所有必要的资源转换。

优化DRBD性能

13. 测量块设备性能

13.1. 测量吞吐量

当测量使用DRBD对系统I/O吞吐量的影响时，系统所能达到的绝对吞吐量几乎没有相关性。更有趣的是DRBD对I/O性能的相对影响。因此，无论有无DRBD，总是有必要测量I/O吞吐量。

本节中描述的测试是侵入性的；它们覆盖数据并使DRBD设备失去同步。因此，只在测试完成后可以丢弃的暂存卷上执行它们是非常重要的。

I/O吞吐量估计的工作原理是将大量数据写入块设备，并测量系统完成写入操作所需的时间。这可以使用相当普遍的实用程序 dd 轻松完成，该实用程序的最新版本包括内置的吞吐量估计。

一个简单的基于 dd 的吞吐量基准，假设您有一个名为 test 的临时资源，该资源当前已连接并且在两个节点上都处于次要角色，如下所示：

```
# TEST_RESOURCE=test
# TEST_DEVICE=$(drbdadm sh-dev $TEST_RESOURCE | head -1)
# TEST_LL_DEVICE=$(drbdadm sh-ll-dev $TEST_RESOURCE | head -1)
# drbdadm primary $TEST_RESOURCE
# for i in $(seq 5); do
    dd if=/dev/zero of=$TEST_DEVICE bs=1M count=512 oflag=direct
  done
# drbdadm down $TEST_RESOURCE
# for i in $(seq 5); do
    dd if=/dev/zero of=$TEST_LL_DEVICE bs=1M count=512 oflag=direct
  done
```

此测试只需将512MiB的数据写入DRBD设备，然后写入其备份设备进行比较。这两项测试各重复5次，以便进行一些统计平均。相关结果是由 dd 生成的吞吐量测量。

对于刚刚启用的DRBD设备，在第一次 dd 运行时性能会略有下降是正常的。这是因为活动日志是 冷 的，不需要担心。

有关一些性能数据，请参阅我们的 优化DRBD吞吐量 一章。

13.2. 测量延迟

延迟测量的目标与吞吐量基准完全不同：在I/O延迟测试中，一个人会写入非常小的数据块（理想情况下是系统可以处理的最小数据块），并观察完成该写入所需的时间。这个过程通常重复几次，以解释正常的统计波动。

与吞吐量测量一样，可以使用无处不在的 dd 实用程序执行I/O延迟测量，尽管设置不同，观察的焦点完全不同。

下面提供了一个简单的基于 dd 的延迟微基准测试，假设您有一个名为 test 的临时资源，该资源当前已连接，并且在两个节点上都处于次要角色：

```
# TEST_RESOURCE=test
# TEST_DEVICE=$(drbdadm sh-dev $TEST_RESOURCE | head -1)
# TEST_LL_DEVICE=$(drbdadm sh-ll-dev $TEST_RESOURCE | head -1)
# drbdadm primary $TEST_RESOURCE
# dd if=/dev/zero of=$TEST_DEVICE bs=4k count=1000 oflag=direct
# drbdadm down $TEST_RESOURCE
# dd if=/dev/zero of=$TEST_LL_DEVICE bs=4k count=1000 oflag=direct
```

这个测试用4kiB分别将1000个数据块写入DRBD设备，然后写入其备份设备进行比较。4096字节是Linux系统（除了s390以外的所有体系结构）、现代硬盘和固态硬盘预计要处理的最小块大小。

重要的是要了解由 dd 生成的吞吐量测量与此测试完全无关；重要的是在完成所述1000次写入过程中经过的时间。将此时间除以1000可得出单个块写入的平均延迟。

这是最坏的情况，因为它是单线程的，并且在之前的一个之后严格执行一个写操作，即I/O深度为1的运行。请查看延迟与IOPs。

此外，有关一些典型的性能值，请参阅我们的优化DRBD延迟一章。

14. 优化DRBD吞吐量

本章讨论优化DRBD吞吐量。它研究了与吞吐量优化有关的一些硬件考虑因素，并详细介绍了为此目的提出的优化建议。

14.1. 硬件注意事项

DRBD吞吐量受底层I/O子系统（磁盘、控制器和相应缓存）的带宽和复制网络的带宽的影响。

I/O子系统吞吐量

I/O子系统吞吐量在很大程度上取决于可并行写入的存储单元（磁盘、固态硬盘、其他闪存[如FusionIO]，…）的数量和类型。一个相当新的SCSI或SAS磁盘通常允许向单个磁盘流式写入大约40MiB/秒；一个固态硬盘将执行300MiB/秒；一个最新的闪存（NVMe）将为1GiB/秒。在分条配置中部署时，I/O子系统将并行化跨磁盘的写入，有效地将单个磁盘的吞吐量乘以配置中的条带数。因此，同样的，40MB/s磁盘将允许有效吞吐量120MB/s的RAID-0或RAID-1+0配置（带三个条带），或200MB/s（带五个条带）；使用固态硬盘和/或NVMe，您可以轻松地达到1GiB/s。

一个带有RAM和BBU的RAID控制器可以加速短峰值（通过缓冲它们），因此太短的基准测试可能也会显示类似于1GiB/s的速度；对于持续的写操作，它的缓冲区只会满负荷运行，然后就没有多大帮助。

硬件中的磁盘镜像（RAID-1）通常对吞吐量的影响很小（如果有的话）。带奇偶校验的磁盘条带（RAID-5）确实对吞吐量有影响，与条带相比通常是不利的；软件中的RAID-5和RAID-6更是如此。

网络吞吐量

网络吞吐量通常由网络上存在的流量以及存在的任何路由/交换基础设施的吞吐量决定。然而，这些问题在通常是专用的back-to-back网络连接的DRBD复制链路中基本上是不相关的。因此，可以通过切换到更高吞吐量的硬件（例如10 Gigabit以太网或56 GiBInfiniband）或通过使用多个网络链路上的链路聚合来提高网络吞吐量，就像可以使用Linux bonding 网络驱动程序那样。

14.2. 吞吐量开销预期

在估计与DRBD相关联的吞吐量开销时，必须考虑以下自然限制：

- DRBD吞吐量受到原始I/O子系统的限制。
- DRBD的吞吐量受到可用网络带宽的限制。

这两个值的 *lower* 建立了DRBD可用的理论吞吐量 *最大值* 。然后，DRBD通过其额外的开销（可以预期小于3%）来减少吞吐量。

- 以两个集群节点为例，它们包含能够达到600 MB/s吞吐量的I/O子系统，并且在它们之间有一个千兆以太网链路。千兆位以太网可以预期为TCP连接产生110MB/s的吞吐量，因此网络连接将是此配置中的瓶颈，并且可以预期最大DRBD吞吐量约为110 MB/s。
- 相比之下，如果I/O子系统仅能够以80 MB/s的速度进行持续的写操作，那么它就构成了瓶颈，您应该只期望大约77 MB/s的最大DRBD吞吐量。

14.3. 调整建议

DRBD提供了许多配置选项，这些选项可能会影响系统的吞吐量。本节列出了一些有关调整吞吐量的建议。然而，由于吞吐量在很大程度上依赖于硬件，因此调整此处描述的选项的效果可能因系统而异。重要的是要明白，这些建议不应被解释为能神奇地消除任何和所有吞吐量瓶颈的 银弹 。

14.3.1. 设置 max-buffers 和 max-epoch-size

这些选项会影响辅助节点上的写入性能。max buffers 是DRBD为将数据写入磁盘而分配的最大缓冲区数，而 max-epoch-size 是两个写入屏障之间允许的最大写入请求数。max buffers 必须等于或大于 max-epoch-size ，才能提高性能。两者的默认值都是2048；对于大多数合理的高性能硬件RAID控制器来说，将其设置

为8000左右应该没问题。

```
resource <resource> {
 net {
  max-buffers   8000;
  max-epoch-size 8000;
  ...
 }
 ...
}
```

14.3.2. 调整TCP发送缓冲区大小

TCP发送缓冲区是用于传出TCP通信的内存缓冲区。默认情况下，它的大小设置为128 KiB。对于在高吞吐量网络（如专用千兆以太网或负载平衡的绑定连接）中使用，将其大小增加到2MiB或更大可能是有意义的。通常不建议发送缓冲区大小超过16MiB（而且也不太可能产生任何吞吐量改进）。

```
resource <resource> {
 net {
  sndbuf-size 2M;
  ...
 }
 ...
}
```

DRBD还支持TCP发送缓冲区自动调整。启用此功能后，DRBD将动态选择适当的TCP发送缓冲区大小。只需将缓冲区大小设置为零即可启用TCP发送缓冲区自动调整：

```
resource <resource> {
 net {
  sndbuf-size 0;
  ...
 }
 ...
}
```

请注意，您的 sysctl 设置 net.ipv4.tcp_rmem 和 net.ipv4.tcp_wmem 仍将影响行为；您应该检查这些设置，并可能将它们设置为类似于 131072 1048576 16777216 （最小128kiB，默认1MiB，最大16MiB）。

net.ipv4.tcp_mem 是另一个怪兽，有不同的单位-不要碰，错误的值很容易把你的机器推入内存不足的情况！

14.3.3. 调整活动日志大小

如果使用DRBD的应用程序是写密集型的，因为它经常在设备上分散地发出小的写操作，那么通常建议使用相当大的活动日志。否则，频繁的元数据更新可能会损害写入性能。

```
resource <resource> {
  disk {
    al-extents 6007;
    ...
  }
  ...
}
```

14.3.4. 禁用屏障和磁盘刷新

本节概述的建议应仅适用于具有非易失性（电池支持）控制器缓存的系统。

配备电池支持的写缓存的系统配备了内置的方法，可以在断电时保护数据。在这种情况下，允许禁用为相同目的而创建的一些DRBD自己的安全措施。这可能对吞吐量有利：

```
resource <resource> {
  disk {
    disk-barrier no;
    disk-flushes no;
    ...
  }
  ...
}
```

14.4. 通过增加冗余实现更好的读取性能

如 read-balancing 下 drbd.conf 的手册页所述，可以通过添加更多数据副本来提高读取性能。

一个大概的数字是：单节点处理读请求时，FusionIO卡上的 fio`给了我们10万IOPs；启用 `读平衡 后，性能跃升到18万IOPs，即+80%！

因此，如果您运行的是以读为主的工作负载（具有大量随机读操作的大型数据库），那么可能值得尝试启用 读平衡 ，并且，也许可以添加另一个副本以获得更大的读IO吞吐量。

15. 优化DRBD延迟

本章讨论优化DRBD延迟。它研究了与延迟最小化有关的一些硬件考虑，并详细介绍了为此目的而提出的优化建议。

15.1. 硬件注意事项

DRBD延迟既受底层I/O子系统（磁盘、控制器和相应缓存）的延迟影响，也受复制网络的延迟影响。

I/O子系统延迟

对于 *旋转媒体* ，I/O子系统延迟主要是磁盘旋转速度的函数。因此，使用快速旋转的磁盘是减少I/O子系统延迟的有效方法。

对于 *固态媒体* （如固态硬盘），闪存控制器是决定因素；下一个最重要的因素是未使用的容量。使用DRBD的Trim/Discard支持将帮助您向控制器提供所需的信息，哪些块可以循环使用。这样，当一个写请求进入时，它可以使用一个提前清理过的块，而不必等待至 **现在** 直到有可用的空间为止。[10]

同样，使用 battery-backed write cache（BBWC）可以减少写入完成时间，还可以减少写入延迟。大多数合理的存储子系统都带有某种形式的电池备份缓存，并允许管理员配置该缓存的哪个部分用于读写操作。建议的方法是完全禁用磁盘读缓存，并将所有可用的缓存内存用于磁盘写缓存。

网络延迟

网络延迟，实际上是包在 主机之间的往返时间（RTT）。它受到许多因素的影响，其中大多数因素与推荐用作DRBD复制链路的专用背靠背网络连接无关。因此，接受网络链路中始终存在一定量的延迟就足够了，对于千兆位以太网，该延迟通常在100到200微秒（µs）的分组RTT上。

Network latency may typically be pushed below this limit only by using lower-latency network protocols, such as running DRBD over Dolphin Express using Dolphin SuperSockets, or a 10GBe direct connection; these are typically in the 50µs range. Even better is InfiniBand, which provides even lower latency.

15.2. 延迟开销预期

至于吞吐量，在估计与DRBD相关联的延迟开销时，需要考虑一些重要的自然限制：

- DRBD延迟受原始I/O子系统的延迟限制。
- DRBD延迟受可用网络延迟的约束。

两者之和确定了DRBD所产生的理论延迟最小值 [11]。然后，DRBD会在这个延迟上增加一点额外的延迟开销，预计这个开销将小于1%。

- 以本地磁盘子系统为例，其写延迟为3ms，网络链路为0.2ms。那么，预期的DRBD延迟将为3.2ms，或者仅写到本地磁盘的延迟大约增加7%。

延迟可能会受到许多其他因素的影响，包括CPU缓存未命中、上下文切换和其他因素。

15.3. 延迟与IOPs

IOPs 是 "每秒I/O操作数" 的缩写。

市场营销通常不喜欢数字变小；新闻稿的撰写不会描述成 "延迟减少10微秒，从50微秒减少到40微秒！"！ - 记住，他们更喜欢 "IOPs性能提高了25%，从20000提高到现在的25000" 。因此，IOPs被发明出来，以得到一个 "越高越好" 的数字。

换句话说，IOPs是延迟的倒数。您需要记住的是，在测量延迟中给出的方法为您提供了延迟响应。纯顺序单线程IO加载的IOPs数，而大多数其他文档将给出一些高度并行加载的数字[12]，因为这给了很多 "漂亮" 的数字。有了这种技巧，DRBD也能为您提供100000 IOPs！

So, please don’t shy away from measuring serialized, single-threaded latency. If you want lots of IOPS, run the fio utility with threads=8 and an iodepth=16, or some similar settings… But please keep in mind that these number will not have any meaning to your setup, unless you’re driving a database with many tens or hundreds of client connections active at the same time.

15.4. 调整建议

15.4.1. 设置DRBD的CPU掩码

DRBD允许为其内核线程设置显式CPU掩码。这对于那些在CPU周期上与DRBD竞争的应用程序尤其有利。

CPU掩码是其二进制表示中的最低有效位表示第一CPU、第二最低有效位表示第二CPU的数字，依此类推。位掩码中的设置位表示DRBD可以使用相应的CPU，而清除位表示它不能使用。因此，例如，CPU掩码1（00000001）意味着DRBD只能使用第一个CPU。掩码12（00001100）表示DRBD可以使用第三和第四CPU。

资源的CPU掩码配置示例如下：

```
resource <resource> {
 options {
  cpu-mask 2;
  ...
 }
 ...
}
```

当然，为了减少DRBD和使用它的应用程序之间的CPU竞争，您需要将应用程序配置为只使用DRBD不使用的那些CPU。

一些应用程序可以通过配置文件中的一个条目来提供这一点，就像DRBD本身一样。其他方法包括在应用程序init脚本中调用 taskset 命令。

保持DRBD线程在相同的L2/L3缓存上运行是有意义的。

但是：CPU的编号并不一定与物理分区相关。您可以尝试X11的 lstopo （或 hwloc ls ）程序或控制台输出的 hwloc-info -v -p 程序，以获得拓扑的概述。

15.4.2. 修改网络MTU

将复制网络的最大传输单元（MTU）大小更改为高于默认值1500字节的值可能是有益的。通俗地说，这称为 "启用Jumbo frames"。

可以使用以下命令更改MTU：

```
# ifconfig <interface> mtu <size>
```

或

```
# ip link set <interface> mtu <size>
```

<interface> 指用于DRBD复制的网络接口。*<size>* 的典型值为9000（字节）。

15.4.3. 启用deadline I/O调度器

当与高性能、支持写回的硬件RAID控制器结合使用时，DRBD延迟可能从使用简单的deadline I/O scheduler而不是CFQ调度器中受益匪浅。后者通常在默认情况下启用。

对I/O调度程序配置的修改可以通过挂载在 /sys 的 sysfs 虚拟文件系统执行。调度程序配置位于 /sys/block/*device* 其中 *<device>* 是DRBD使用的备份设备。

Enabling the deadline scheduler works via the following command:

```
# echo deadline > /sys/block/<device>/queue/scheduler
```

然后，您还可以设置以下值，这些值可以提供额外的延迟好处：

- 禁用前台合并：

  ```
  # echo 0 > /sys/block/<device>/queue/iosched/front_merges
  ```

- 将读取I/O deadline时间缩短到150毫秒（默认为500毫秒）：

  ```
  # echo 150 > /sys/block/<device>/queue/iosched/read_expire
  ```

- 将写入I/O deadline时间缩短到1500毫秒（默认值为3000毫秒）：

  ```
  # echo 1500 > /sys/block/<device>/queue/iosched/write_expire
  ```

如果这些值影响显著的延迟改进，则可能需要将它们永久化，以便在系统启动时自动设置它们。 Debian和Ubuntu系统通过 sysfsutils 包和 /etc/sysfs.conf 配置文件提供此功能。

还可以通过通过内核命令行传递 elevator 选项来选择全局I/O调度程序。为此，编辑引导加载程序配置（如果使用的是grub引导加载程序，通常可以在 /etc/default/grub 中找到），并将 elevator=deadline 添加到内核引导选项列表中。

[10] 在低端硬件上，您可以通过保留一些空间来帮助这一点，只需保留总空间的10%到20%就可以了。
[11] 对于 protocol C，因为其他节点也必须写入稳定存储
[12] 如在 "*16线程，IO深度为32*" – 这意味着512个I/O请求是并行完成的！

了解更多

16. DRBD内幕

本章介绍了DRBD的一些内部算法和结构的背景信息。它是为有兴趣的用户希望获得DRBD一定程度的背景知识。它没有深入研究DRBD的内部工作机制，不能作为DRBD开发人员的参考。为此，请参阅出版物中列出的论文，当然也请参阅DRBD源代码中的注释。

16.1. DRBD元数据

DRBD存储关于它保存在一个专用区域中的数据的各种信息。此元数据包括：

- DRBD设备的大小，
- 生成标识符（GI，详细说明见生成标识符），
- 活动日志（AL，在活动日志中详细描述）。
- 快速同步位图（详见快速同步位图），

此元数据可以存储在内部或外部。所使用的方法是基于每个资源可配置的。

16.1.1. 内部元数据

将资源配置为使用内部元数据意味着DRBD将其元数据存储在与实际生产数据相同的物理低级设备上。它通过在设备的末端留出一个区域来存储元数据。

优势

由于元数据与实际数据有着千丝万缕的联系，因此在硬盘出现故障时，管理员无需执行任何特殊操作。元数据与实际数据一起丢失，也一起还原。

劣势

如果较低级别的设备是单个物理硬盘（而不是RAID集），内部元数据可能会对写入吞吐量产生负面影响。应用程序的写请求的性能可能触发DRBD中的元数据的更新。如果元数据存储在硬盘的同一个磁盘上，则写入操作可能导致硬盘的写/读磁头的两个额外移动。

如果您计划将内部元数据与现有的较低级别设备结合使用，而该设备已具有您希望保留的数据，则 *必须* 考虑DRBD元数据所需的空间。

否则，在创建DRBD资源时，新创建的元数据将覆盖较低级别设备末尾的数据，从而可能在该过程中破坏现有文件。

要避免这种情况，您必须执行以下操作之一：

- 扩容你的下层设备。只要在相应的卷组中有可用空间，任何逻辑卷管理工具（如 LVM）都可以做到这一点。硬件存储解决方案也可能支持它。
- 缩容较低级别设备上的现有文件系统。您的文件系统可能支持，也可能不支持。
- 如果两者都不可能，请改用external meta data。

要估计必须放大低级设备或缩小文件系统的数量，请参见估计元数据大小。

16.1.2. 外部元数据

外部元数据简单地存储在一个独立的专用块设备上，与保存生产数据的设备不同。

优势

对于某些写操作，使用外部元数据会产生稍有改进的延迟行为。

劣势

元数据与实际生产数据并非密不可分。这意味着，在硬件故障仅破坏生产数据（而非DRBD元数据）的情况下，需要手动干预，以实现从幸存节点到随后更换的磁盘的完全数据同步。

如果适用以下所有条件，使用外部元数据也是唯一可行的选择：

- 您正在使用DRBD复制已包含要保留的数据的现有设备，*并且*
- 现有设备不支持扩容，*并且*
- 设备上现有的文件系统不支持收缩。

要估计专用于保存设备元数据的块设备所需的大小，请参见估计元数据大小。

外部元数据至少需要1MB的设备大小。

16.1.3. 估计元数据大小

您可以使用以下公式计算DRBD元数据的确切空间需求：

$$M_s = \lceil \frac{C_s}{2^{18}} \rceil * 8 * N + 72$$

插图 14. 计算DRBD元数据大小（精确）

C_s is the data device size in sectors, and ***N*** is the number of peers.

您可以通过发出 `blockdev --getsize64 <device>` ；为转换为MB，除以1048576（=2^{20} 或1024^2）来检索设备大小（字节）。

实际上，您可以使用一个相当好的近似值，如下所示。注意，在这个公式中，单位是兆字节，而不是扇区：

$$M_{MB} < \frac{C_{MB}}{32768} * N + 1$$

插图 15. 估计DRBD元数据大小（大约）

16.2. 生成标识符

DRBD使用生成标识符（GIs）来标识复制数据的 "生成"。

这是DRBD的内部机制，用于

- 确定这两个节点是否实际上是同一集群的成员（与意外连接的两个节点相反），
- 确定背景重新同步的方向（如有必要），
- 确定是否需要完全重新同步或部分重新同步是否足够，
- 识别裂脑。

16.2.1. 数据生成

DRBD在以下每一次出现时都标志着新数据生成的开始：

- 初始设备完全同步，
- 断开连接的资源切换到主角色，
- 主角色中的资源正在断开连接。

因此，我们可以总结出，每当资源处于 *Connected* 连接状态，并且两个节点的磁盘状态都是 *UpToDate* 时，两个节点上的当前数据生成都是相同的。反之亦然。注意，当前实现使用最低的位来编码节点的角色（主/次）。因此，即使被认为具有相同的数据生成，不同节点上的最低位也可能不同。

每个新的数据生成都由一个8字节的通用唯一标识符（UUID）标识。

16.2.2. 生成标识符元组

DRBD在本地资源元数据中保留一些有关当前和历史数据生成的信息：

当前UUID
从本地节点的角度看，这是当前数据生成的生成标识符。当资源连接并完全同步时，节点之间的当前UUID是相同的。

Bitmap UUIDs
这是此磁盘上位图跟踪更改（每个远程主机）所依据的生成的UUID。与磁盘上同步位图本身一样，此标识符仅在远程主机断开连接时才相关。

历史UUIDs
这些是当前数据生成之前的数据生成的标识符，大小为每个（可能的）远程主机有一个插槽。

这些项统称为 *generation identifier tuple* ，简称为 "*GI tuple*"。

16.2.3. 生成标识符如何更改

开始新的数据生成

当处于 *主要* 角色的节点失去与其对等节点的连接时（通过网络故障或手动干预），DRBD按以下方式修改其本地生成标识符：

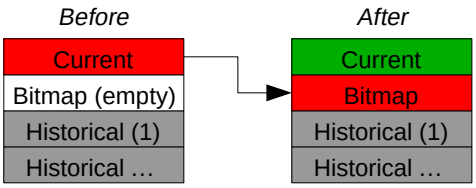


插图 16. 在新数据生成开始时更改GI元组

1. 主节点为新的数据生成创建一个新的UUID。这将成为主节点的当前UUID。
2. 前一个当前的UUID现在引用位图跟踪更改的生成，因此它成为主节点的新位图UUID。
3. 在次节点上，GI元组保持不变。

重新同步完成

当重新同步结束时，同步目标采用来自同步源的整个GI元组。

同步源保持相同的集合，并且不生成新的uuid。

16.2.4. DRBD如何使用生成标识符

当节点之间建立连接时，两个节点交换其当前可用的生成标识符，并相应地继续。存在许多可能的结果：

两个节点上的当前uuid均为空
本地节点检测到其当前UUID和对等方的当前UUID都为空。这是新配置的资源的正常情况，该资源尚未启动初始完全同步。不进行同步；必须手动启动。

一个节点上的当前uuid为空
本地节点检测到对等方的当前UUID为空，而其自身的UUID为空。对于刚刚启动初始完全同步的新配置资源，这是正常情况，已选择本地节点作为初始同步源。DRBD现在设置磁盘同步位图中的所有位（意味着它认为整个设备不同步），并作为同步源开始同步。在相反的情况下（本地当前UUID为空，对等端为非空），DRBD执行相同的步骤，只是本地节点成为同步目标。

匹配当前 UUIDs
本地节点检测到其当前UUID和对等方的当前UUID不为空且相等。这是资源在处于次要角色时进入断开连接模式的正常情况，并且在断开连接时未在任何节点上升级。不需要同步，因为不需要同步。

位图UUID与对等方的当前UUID匹配

本地节点检测到其位图UUID与对等方的当前UUID匹配，并且对等方的位图UUID为空。这是在次要节点发生故障（本地节点处于主要角色）后发生的正常且预期的事件。这意味着对等方从来没有在同一时间成为主要的，并且一直在同一数据生成的基础上工作。DRBD现在启动一个正常的后台重新同步，本地节点成为同步源。相反，如果本地节点检测到其位图UUID为空，并且该位图与本地节点的当前UUID匹配，则这是本地节点发生故障后的正常和预期发生。再次，DRBD现在启动一个正常的后台重新同步，本地节点成为同步目标。

当前UUID与对等方的历史UUID匹配

The local node detects that its current UUID matches one of the peer’s historical UUIDs. This implies that while the two data sets share a common ancestor, and the peer node has the up-to-date data, the information kept in the peer node’s bitmap is outdated and not usable. Thus, a normal synchronization would be insufficient. DRBD now marks the entire device as out-of-sync and initiates a full background re-synchronization, with the local node becoming the synchronization target. In the opposite case (one of the local node’s historical UUID matches the peer’s current UUID), DRBD performs the same steps, except that the local node becomes the synchronization source.

位图uuid匹配，当前uuid不匹配

The local node detects that its current UUID differs from the peer’s current UUID, and that the bitmap UUIDs match. This is split brain, but one where the data generations have the same parent. This means that DRBD invokes split brain auto-recovery strategies, if configured. Otherwise, DRBD disconnects and waits for manual split brain resolution.

当前uuid和位图uuid都不匹配

The local node detects that its current UUID differs from the peer’s current UUID, and that the bitmap UUIDs *do not* match. This is split brain with unrelated ancestor generations, thus auto-recovery strategies, even if configured, are moot. DRBD disconnects and waits for manual split brain resolution.

无Uuid匹配

最后，如果DRBD在两个节点的GI元组中甚至检测不到一个匹配的元素，它会记录一个关于不相关数据和断开连接的警告。这是DRBD防止两个以前从未听说过的集群节点意外连接的保护措施。

16.3. 活动日志

16.3.1. 目的

在写操作期间，DRBD将写操作转发到本地备份块设备，但也通过网络发送数据块。出于所有实际目的，这两个动作同时发生。随机定时行为可能导致写入操作已完成，但经由网络的传输尚未发生的情况，反之亦然。

如果此时活动节点发生故障，并且正在启动故障转移，则此数据块在节点之间不同步—它在崩溃之前已写入到发生故障的节点上，但复制尚未完成。因此，当节点最终恢复时，必须在后续同步期间从数据集中删除此块。否则，崩溃的节点将比幸存的节点 "提前一次写入"，这将违反复制存储的 "全部或无" 原则。这是一个不仅限于DRBD的问题，事实上，这个问题实际上存在于几乎所有的复制存储配置中。许多其他存储解决方案（就像DRBD本身一样，在0.7之前）因此要求在活动节点发生故障后，数据在恢复后必须完全同步。

DRBD的方法，因为版本0.7，是不同的。存储在元数据区域中的 *活动日志*（AL）跟踪那些 "最近" 被写入的块。通俗地说，这些地区被称为 "热区"。

如果同步发生故障时处于活动模式的临时故障节点，则只需要同步AL中突出显示的热数据块（加上当前活动对等机上位图中标记的任何块），而不需要同步整个设备。这大大缩短了活动节点崩溃后的同步时间。

16.3.2. 活动范围

活动日志有一个可配置的参数，即活动扩展的数量。每个活动数据块都会在主崩溃后重新传输的数据量上增加4MiB。必须将此参数理解为以下两个对立面之间的折衷：

许多活动范围

保持较大的活动日志可以提高写吞吐量。每次激活新数据块时，旧数据块都会重置为非活动数据块。此转换需要对元数据区域执行写操作。如果活动扩展数据块的数量很高，则旧的活动扩展数据块很少被换出，从而减少元数据写入操作，从而提高性能。

活动范围很少
保持一个小的活动日志可以减少活动节点故障和后续恢复后的同步时间。

16.3.3. 选择适当的活动日志大小

应根据给定同步速率下的所需同步时间来考虑扩展数据块的数量。活动范围的数量可以按以下方式计算：

$$E = \frac{R * t_{sync}}{4MiB}$$

插图 17. 基于同步速率和目标同步时间的活动数据块计算

R 是同步速率，以MiB/s为单位。*t~sync* 是目标同步时间，以秒为单位。*E* 是活动范围的结果数量。

举个例子，假设集群有一个I/O子系统，其吞吐量（*R*）为200兆字节/秒，配置为60兆字节/秒的同步速率，并且我们希望将目标同步时间（t_{tsync}）保持在4分或240秒：

插图 18. 基于同步速率和目标同步时间的活动数据块计算（示例）

最后，DRBD 9需要在次要节点上保持AL，因为它们的数据可能用于同步其他次要节点。

16.4. 快速同步位图

quick sync bitmap是内部数据结构，DRBD在每个对等资源上使用它来跟踪正在同步（在两个节点上相同）或不同步的块。它只在资源处于断开连接模式时才相关。

在快速同步位图中，一位表示磁盘上的4 KiB数据块。如果该位被清除，则表示对应的块仍与对等节点同步。这意味着自断开连接时起就没有写入块。相反，如果设置了位，则意味着块已被修改，并且需要在连接再次可用时重新同步。

当DRBD检测到断开连接的设备上的写I/O，并因此开始在快速同步位图中设置位时，它在RAM中这样做——从而避免了昂贵的同步元数据I/O操作。只有当相应的块变冷时（即，从Activity Log过期），DRBD才会在快速同步位图的磁盘表示中进行适当的修改。同样，如果在断开连接的同时碰巧手动关闭了剩余节点上的资源，DRBD会将快速同步位图刷新到持久存储。

当对等节点恢复或重新建立连接时，DRBD结合来自两个节点的位图信息来确定必须重新同步的 *总数据集* 。同时，DRBDexamines the generation identifiers以确定同步的 *方向* 。

然后，作为同步源的节点将商定的块发送到对等节点，在同步目标确认修改时清除位图中的同步位。如果重新同步现在被中断（例如，被另一个网络中断）并随后恢复，它将在中断的地方继续—当然，同时修改的任何其他块都会被添加到重新同步数据集中。

使用 drbdadm pause-sync 和 `drbdadm resume-sync`命令也可以暂停并手动恢复重新同步。但是，您不应该心平气和地这样做——中断重新同步会使辅助节点的磁盘不一致的时间超过必要的时间。

16.5. 节点围栏接口

DRBD有一个为fencing定义的接口[13]如果复制链接被中断，则是对等节点。与Heartbeat捆绑在一起的“drbd peer outdater”助手是此接口的引用实现。但是，您可以轻松实现自己的同伴击剑助手程序。

只有在出现以下情况时才会调用击剑助手

1. 在资源的（或“common”）“handler s”部分中定义了“fence peer”处理程序，然后_
2. 资源的“fencing”选项设置为“resource only”或“resource and stonith”_
3. 复制链接中断的时间足够DRBD [14]以检测网络故障。

当调用指定为“fence　peer”处理程序的程序或脚本时，它具有“DRBD_RESOURCE”和“DRBD_peer”环境变量。它们分别包含受影响的DRBD资源的名称和对等主机名。

任何对等围栏帮助程序（或脚本）必须返回以下退出代码之一：

表标题 1. `fence peer` handler退出代码

Exit code	Implication
3	Peer’ s disk state was already *Inconsistent*.
4	Peer’ s disk state was successfully set to *Outdated* (or was *Outdated* to begin with).
5	Connection to the peer node failed, peer could not be reached.
6	Peer refused to be outdated because the affected resource was in the primary role.
7	Peer node was successfully fenced off the cluster. This should never occur unless fencing is set to resource-and-stonith for the affected resource.

[13] 有关围栏和STONITH的讨论，请参见相应的起搏器页面http://clusterlabs.org/doc/crm\u Fencing.html.
[14] 这意味着TCP超时、“ping超时”或内核触发连接中止，例如因为网络链接断开。

17. 获取更多信息

17.1. 商业DRBD支持

商业DRBD支持、咨询和培训服务可从项目发起公司 linbit 获得。

17.2. 公开邮件列表

关于DRBD的一般用法问题的公开邮件列表是 drbd-user@lists.linbit.com。这是一个仅限订户的邮件列表，您可以在 https://lists.linbit.com/listinfo/drbd-user/ 上订阅。完整的列表存档可在 https://lists.linbit.com/pipermail/drbd-user/ 上获得。

17.3. 公共IRC频道

有些DRBD开发人员有时可以在 irc.freenode.net 公共irc服务器上找到，特别是在以下通道中：

- #drbd，和
- #clusterlabs.

联系IRC是讨论DRBD改进建议和进行开发人员级讨论的好方法。

询问问题时，请使用一些公共粘贴服务来提供DRBD配置、日志文件、`drbdsetup status --verbose --statistics`输出和 `/proc/drbd' 内容。没有这些数据是很难帮助的。

17.4. 官方推特帐户

linbit 维护 官方 twitter帐户。

如果你在tweet上发布了关于DRBD的信息，请附上 #drbd 标签。

17.5. 出版物

DRBD的作者已经撰写并发表了一些关于DRBD的论文，或者DRBD的一个特定方面。下面是一个简短的选择：

- Philipp Reisner. 'DRBD 9 - What’ s New'. 2012. Available at http://www.drbd.org/fileadmin/drbd/publications/whats-new-drbd-9.pdf // FIXME only bad version at https://www.netways.de/fileadmin/images/Events_Trainings/Events/OSDC/2013/Slides_2013/Philipp_Reisner_Neues_in_DRBD9.pdf // FIXME - put fixed version there
- Lars Ellenberg. 'DRBD v8.0.x and beyond'. 2007. Available at http://drbd.linbit.com/fileadmin/drbd/publications/drbd8.linux-conf.eu.2007.pdf
- Philipp Reisner. 'DRBD v8 - Replicated Storage with Shared Disk Semantics'. 2007. Available at http://drbd.linbit.com/fileadmin/drbd/publications/drbd8_orig.pdf FIXME sagt 2005?
- Philipp Reisner. 'Rapid resynchronization for replicated storage'. 2006. Available at http://drbd.linbit.com/fileadmin/drbd/publications/drbd-activity-logging_v6.pdf.

您可以在 http://drbd.linbit.com/home/publications/上找到更多信息。

17.6. 其他有用资源

- Wikipedia keeps an entry on DRBD.
- Both the Linux-HA wiki and
- ClusterLabs website 提供了一些关于在高可用集群中优化DRBD的有用信息

附录

附录标题 A: 最近的变化

本附录适用于从早期DRBD版本升级到DRBD 9.0的用户。它强调了对DRBD的配置和行为的一些重要更改。

A.1. 连接

使用DRBD 9，数据可以跨两个以上的节点复制。

这也意味着现在不赞成堆叠DRBD卷（尽管仍有可能）；现在可以使用DRBD作为网络阻隔设备（a*DRBD client*) now makes sense.

与此更改关联的是

- 元数据大小更改（每个节点一个bitmap）
- /proc/drbd 现在只提供最少的信息，请参见 <s-drbdadm-status，` drbdadm status'>
- 从multiple peers同步或同步到multiple peers
- 即使在activity log角色中也会使用Secondary

A.2. 自动升级功能

DRBD 9可以配置为按需自动执行 *主/次* 角色切换。

此功能将替换 become-primary-on 配置值以及旧的Heartbeat v1 drbddisk 脚本。

有关详细信息，请参见自动提升资源。

A.3. 提高性能

DRBD 9已经看到了显著的性能改进，这取决于您的特定硬件，它的速度快了两个数量级（测量随机写入的I/O操作数/秒）。

A.4. 一个资源中有多个卷

卷是DRBD 8.4中的一个新概念。在8.4之前，每个资源只有一个与其相关联的块设备，因此DRBD设备和资源之间存在一对一的关系。自8.4以来，多个卷（每个卷对应于一个块设备）可以共享一个复制连接，而该复制连接又对应于一个资源。

这会导致一些其他必要的行为改变：

A.4.1. 更改以指定udev符号链接

DRBD udev集成脚本管理指向各个块设备节点的符号链接。它们存在于 /dev/drbd/by-res/ 和 /dev/drbd/by-disk 目录中。

从DRBD 8.4起，一个资源可以对应于多个卷， /dev/DRBD/by-res/*<resource>* 是一个 *目录* ，包含指向各个卷的符号链接：

Listing 8. DRBD 8.4中udev管理的DRBD符号链接

```
lrwxrwxrwx 1 root root 11 2015-08-09 19:32 /dev/drbd/by-res/home/0 -> ../../drbd0
lrwxrwxrwx 1 root root 11 2015-08-09 19:32 /dev/drbd/by-res/data/0 -> ../../drbd1
lrwxrwxrwx 1 root root 11 2015-08-09 19:33 /dev/drbd/by-res/nfs-root/0 -> ../../drbd2
lrwxrwxrwx 1 root root 11 2015-08-09 19:33 /dev/drbd/by-res/nfs-root/1 -> ../../drbd3
```

当移动到DRBD 8.4时，必须更新这样一个符号链接引用文件系统的配置，通常只需在符号链接路径上附加 /0

。如果文件系统是通过 /etc/fstab 中的 UUID= 或 /dev/drbd*Xx* 引用的，则不需要进行任何更改。

A.5. 配置语法的更改

本节重点介绍对配置语法的更改。它影响 /etc/drbd.d ，和 /etc/drbd.conf 中的DRBD配置文件。

drbdadm 解析器仍然接受8.4之前的配置语法，并在内部自动转换为当前语法。除非您计划使用DRBD 9的新特性，否则不需要将配置修改为当前语法。但是，建议您最终采用新语法，因为DRBD 9不再支持旧格式。

A.5.1. 布尔配置选项

conf 支持多种布尔配置选项。在DRBD 8.4之前的语法中，这些布尔选项设置如下：

Listing 9. 带有布尔选项的DRBD 8.4之前的配置示例

```
resource test {
  disk {
    no-md-flushes;
  }
}
```

如果要在 common 配置部分中设置布尔变量，然后对单个资源重写该变量，则会导致配置问题：

Listing 10. Pre DRBD 8.4配置示例，在 common 部分提供布尔选项

```
common {
  disk {
    no-md-flushes;
  }
}
resource test {
  disk {
    # No facility to enable disk flushes previously disabled in
    # "common"
  }
}
```

在DRBD 8.4中，所有布尔选项的值都为 yes 或 no，这使得它们很容易从 common 和单个 resource 部分进行配置：

Listing 11. DRBD 8.4 common 部分中带有布尔选项的配置示例

```
common {
 md-flushes no;
}
resource test {
 disk {
  md-flushes yes;
 }
}
```

A.5.2. syncer 部分不再存在

在DRBD 8.4之前，syncer 部分的配置语法在8.4中已经过时。所有以前存在的 syncer 选项现在都移到了资源的 net 或 disk 部分。

Listing 12. 带有 syncer 部分的Pre-DRBD 8.4配置示例

```
resource test {
  syncer {
    al-extents 3389;
    verify-alg md5;
  }
  ...
}
```

以上示例用DRBD 8.4语法表示如下：

Listing 13. DRBD 8.4替换了 syncer 部分的配置示例

```
resource test {
  disk {
    al-extents 3389;
  }
  net {
    verify-alg md5;
  }
  ...
}
```

A.5.3. protocol 选项不再特殊

在以前的DRBD版本中，“protocol”选项需要自己指定，而不是作为“net”部分的一部分。DRBD 8.4消除了这个异常：

Listing 14. 带有独立“protocol”选项的Pre-DRBD 8.4配置示例

```
resource test {
  protocol C;
  ...
  net {
    ...
  }
  ...
}
```

等效的DRBD 8.4配置语法为：

Listing 15. DRBD 8.4配置示例，在 net 部分中有 protocol 选项

```
resource test {
  net {
    protocol C;
    ...
  }
  ...
}
```

A.5.4. 按资源新建 option 部分

DRBD 8.4引入了一个新的 options 部分，可以在 resource 或 common 部分中指定。 cpu-mask`选项已经从 `syncer 部分移到了这个部分，在这个部分中它以前配置得很笨拙。on-no-data-accessible 选项也移到了这个部分，而不是8.4以前的 disk 中版本。

Listing 16. Pre DRBD 8.4配置示例，带有 cpu-mask 和 on-no-data-accessible

```
resource test {
  syncer {
    cpu-mask ff;
  }
  disk {
    on-no-data-accessible suspend-io;
  }
  ...
}
```

等效的DRBD 8.4配置语法为：

Listing 17. 带有 options 部分的DRBD 8.4配置示例

```
resource test {
  options {
    cpu-mask ff;
    on-no-data-accessible suspend-io;
  }
  ...
}
```

A.6. 网络通信的在线变化

A.6.1. 更改复制协议

在DRBD 8.4之前，当资源处于联机和活动状态时，无法更改复制协议。您必须更改资源配置文件中的 protocol 选项，然后在两个节点上分别输入 drbdadm disconnect 和 drbdadm connect 。

在DRBD 8.4中，复制协议可以动态更改。例如，您可以临时将连接从正常同步复制模式切换到异步复制模式。

Listing 18. 建立连接时更改复制协议

```
drbdadm net-options --protocol=A <resource>
```

A.6.2. 从单主复制更改为双主复制

在DRBD 8.4之前，当资源处于联机和活动状态时，无法在单主到双主之间切换或切换回。您必须更改资源配置文件中的 allow-two-primaries 选项，然后在两个节点上分别发出 drbdadm disconnect 和 drbdadm connect。

在DRBD 8.4中，可以在线切换模式。

使用DRBD双主模式的应用程序 *需要* 使用集群文件系统或其他一些分布式锁定机制。无论双主模式是临时启用还是永久启用，这都适用。

当资源在线时，请参阅临时双主模式以切换到双主模式。

A.7. 对 drbdadm 命令的更改

A.7.1. 更改pass-through选项

在DRBD 8.4之前，如果希望 drbdadm 将特殊选项传递给 drbdsetup ，则必须使用神秘的 -- --<option> 语法，如下例所示：

Listing 19. Pre DRBD 8.4 drbdadm 传递选项

```
drbdadm -- --discard-my-data connect <resource>
```

相反， drbdadm 现在接受这些传递选项作为普通选项：

Listing 20. DRBD 8.4 drbdadm 传递选项

```
drbdadm connect --discard-my-data <resource>
```

旧语法仍然受支持，但强烈建议不要使用它。但是，如果选择使用新的更直接的语法，则必须在子命令（connect）*之后* 指定选项（ --discard-my-data），在资源标识符 *之前* 指定选项。

A.7.2. --force 选项替换了 --overwrite-data-of-peer

--overwrite-data-of-peer 选项在DRBD 8.4中不再存在。它已经被更简单的 --force 所取代。因此，要启动初始资源同步，不再使用以下命令：

Listing 21. Pre-DRBD 8.4初始同步 drbdadm 命令'

```
drbdadm -- --overwrite-data-of-peer primary <resource>
```

改为使用以下命令：

Listing 22. DRBD 8.4初始同步 drbdadm 命令'

```
drbdadm primary --force <resource>
```

A.8. 更改的默认值

在DRBD 8.4中，更新了几个 drbd.conf 默认值，以匹配Linux内核和可用服务器硬件的改进。

A.8.1. 并发活动日志扩展数据块数（al extents）

al-extents　以前的默认值127已更改为1237，通过减少元数据磁盘写入操作的数量，可以获得更好的性能。考虑到千兆位以太网和更高带宽复制链路的普遍性，在主节点崩溃后相关的延长重新同步时间（这一变化引入）是微不足道的。

A.8.2. 运行长度编码（use-rle）

位图传输的运行长度编码（RLE）在DRBD　8.4中默认启用；使用RLE选项的默认值是yes。RLE极大地减少了在quick-sync bitmap交换过程中（在两个断开连接的节点重新连接时）传输的数据量。

A.8.3. I/O错误处理策略（ on-io-error）

DRBD 8.4默认为masking I/O errors，它取代了<<fp io error pass on的早期行为，将它们传递给I/O堆栈的上层。这意味着在故障驱动器上运行的DRBD卷将自动切换到　*无盘*　磁盘状态，并继续从其对等节点提供数据。

A.8.4. 变速率同步

Variable-rate synchronization在DRBD 8.4中默认打开。默认设置相当于以下配置选项：

Listing 23. DRBD 8.4变速率同步的默认选项

```
resource test {
 disk {
  c-plan-ahead 20;
  c-fill-target 50k;
  c-min-rate 250k;
 }
 ...
```

A.8.5. 可配置的DRBD设备数量（ minor-count）

在DRBD　8.4中，可配置的DRBD设备（以前是255）的最大数量是1048576（2^{20}）。这更像是生产系统中不可能达到的理论极限。