DRBD 中文应用指南

写在前面的话：

为了方便学习，将 drbd 的英文文档翻译出来，以供学习和参考。但因为本人英语水平有限，以及对 drbd 的了解并不足够深入，翻译中出现很多不通顺或者是语病的地方，请大家谅解。如果问题可以反馈给 15038051897@163.com，鄙人将在第一时间进行更正。

如果是学习外语文刊的翻译，那么他的水平也仅仅是停留在翻译者的水平，因此建议有实力的同学还是从官网获取第一手信息。

可在 http://download.csdn.net/source/3540203 中下载相关 PDF，在这里很多的图片无法显示，请谅解！

刘运锋

2011-08-22

一．关于本指南

本指南主要为分布式复制块设备（DRBD 技术）的用户提供一个简明的参考指南和手册。

DRBD 是由 LINBIT 项目所在的公司赞助的，免费，并比较有益处的 DRBD 技术社区。该指南在不断的更新，以保持和（作者）新添加在 DRBD 技术新功能的信息的同步。在线的 HTML 本指南版本可以参见 http://www.drbd.org/users-guide/。

重要

本指南可能有些段落包含“草稿”字样，是因为最近这些段落已增加了可能不是很权威的文档说明。欢迎广大读者反馈意见，以示对我们的鼓励。邮箱地址为：drbd-user@lists.linbit.com。

本指南分五个部分和一个附录：

第一部分，介绍“DRBD”：介绍 DRBD 技术的基本功能。简要概述 DRBD 技术在 Linux I / O 堆栈的 DRBD 技术的定位和基本 DRBD 技术概念另外还增加了被认为是 DRBD 技术的最重要特点。

第二部分“DRBD 技术编译，安装和配置”在源代码中如何建立 DRBD 技术，DRBD 技术编译安装包，DRBD 技术的集群系统，以描述工作流程。

第三部分“DRBD 技术”用在如何管理 DRBD 技术，DRBD 技术的源码配置，修改，并介绍如何解决常见问题。

第四部分“DRBD 技术应用相结合”，以增加存储的复制如何利用的 DRBD 技术优势和应用程序可用性。存活状态的探测，以及在与群集管理器，LVM 和先进的组合的组合，整合 DRBD LVM 和 GFS 以实现高可用性,并添加到 Xen 虚拟化环境中。

第五部分“优化 DRBD 性能”：DRBD 的配置获得最佳性能指南。

第六部分，“关于 DRBD 技术更多学习”：DRBD 的内部原理的深入理解，还包含一些其他资源。希望这本指南对读者有用。

最后，附录 A，DRBD 技术系统手册页包含了 Linux 手册分发给了最新版本的 DRBD 技术参考之用，网页的在线版本。

对 DRBD 技术培训或支援服务感兴趣的用户请联系<sales@linbit.com>。

第一篇　　DRBD 介绍

目录

1.DRBD 的基本功能

分布式复制块设备（DRBD 技术）是一种基于软件的，无共享，复制的存储解决方案，在服务器之间的对块设备（硬盘，分区，逻辑卷等）进行镜像。

DRBD 镜像数据

实时性：当应用对磁盘的数据进行修改时，复制立即发生。

透明性：应用程序的数据存储在镜像设备上是独立和透明的，数据可存储在不同的服务器上。

同步镜像和异步镜像：同步镜像，当本地发申请进行写操作进行时，同步写到两台服务器上。异步镜像，当本地写申请已经完成对本地的写操作时，开始对对应的服务器进行写操作。

1.1 内核模块

DRBD 技术的核心功能是通过一个 Linux 内核模块实现的。具体来说，DRBD 包含一个虚拟的块设备，因此 DRBD 是位于“右底部附近的“一个系统的 I/ O 堆栈。正因为如此，DRBD 极为灵活，这使得它成为几乎适合任何程序的一个高可用的块复制解决方案。

重要

DRBD 技术，顾名思义，由 Linux 内核架构所支撑，它建立在不可知的层上面。因此，DRBD 不可能添加了上层所具备的一些新特性。例如，DRBD 技术不能自动检测文件系统损坏或添加双主动群集能力，像 Ext3 或者 XFS 文件系统。

图 1.1　DRBD 在 Linux 的 I/O 堆栈中的位置

1.2 用户空间管理工具

为了能够管理和配置 DRBD 的资源，DRBD 配备了一些管理工具与内核模块进行通信。

drbdadm：高层的 DRBD 程序管理套件工具。它从配置文件/etc/drbd.conf 中获取所有配置参数。drbdadm 为 drbdsetup 和 drbdeta 两个命令充当程序的前端应用，执行 drbdadm 实际是执行的 drbdsetup 和 drbdeta 两个命令。

drbdsetup：drbdsetup 可以让用户配置已经加载在内核中运行的 DRBD 模块，它是底层的 DRBD 程序管理套件工具。使用该命令时，所有的配置参数都需要直接在命令行中定义，虽然命令和灵活，但是大大的降低了命令的简单易用性，因此很多的用户很少使用 debdsetup。

drbdmeta：drbdmeta 允许用户创建、转储、还原和修改 drbd 的原数据结构。这个命令也是用户极少用到。

1.3 资源

在 DRBD 中，资源是所有可复制移动存储设备的总称。这些措施包括：

资源名：资源名可以指定除了空格外 us-ascii 中的任意字符。

DRBD 设备：DRBD 的虚拟块设备。它有一个主设备号为 147 的设备，默认的它的次要号码编从 0 开始。相关的块设备需命名为/ dev/ drbdm，其中 M 是设备的次要号码。

备忘

早期的 DRBD 版本中，被“劫持”的 NBD 设备主号码为 43。这个很早就过时了，147 是 LANANA-注册的 DRBD 设备主号码。

磁盘配置：DRBD 内部应用需要本地数据副本，元数据。

网络配置：各个对等接点间需要进行数据通信。

1.4 资源角色

在 DRBD 中，每个节点都有自己的角色，比如主或者备。

备忘

这里属于的选择不是随意的，这些角色的没有被 DRBD 的作者命名为“主动”和“被动”。主和

备的概念是依赖与存储的可用性的，而主动和被动反应的是一个应用的可用性。通常情况下在高可用环境中主节点也往往是活动的节点，但是这不是必须的。

在主 DRBD 设备中可以进行不受限制的读和写的操作。他可用来创建和挂载文件系统、初始化或者是直接 I/O 的快设备，等等。

在次 DRBD 设备中，接收所有来自对等节点的更新，但是与此同时也就完全拒绝了访问。它即不能被应用也不能被读写访问。为次节不能被读写访问是为了保持缓冲一致性，这就意味着次节点是不可能以任何形式被访问的。

人工干预和管理程序的自动聚类算法都可以改变资源的角色。资源从次节点变为主节点为升级，而反操作则成为降级。

2.DRBD 的特点

这一章主要介绍 DRBD 的各种有用的特性，并介绍关于这些特性一些背景信息。其中的一部分会对大多数的用户很有用，其他的就和具体的部署方式有关系。

,普通的行政任务与第 7 章、故障排除和差错恢复含说明如何实现和使用这些功能,在日常操作。

将在第 6 章的常见的管理任务和第 7 章故障排除和错误恢复，介绍在日常操作管理中如何启用和利用这些特性。

2.1 单主模式

在单主模式下，任何资源在任何特定的时间，集群中只存在一个主节点。正是因为这样在集群中只能有一个节点可以随时操作数据,这种模式可用在任何的文件系统上（EXT3、EXT4、XFS 等等）。

部署 DRBD 单主节点模式可保证集群的高可用性（fail-over 遇故障转移的能力）。

2.2 双主模式

这是 DRBD8.0 之后的新特性。

在双主模式下，任何资源在任何特定的时间，集群中都存在两个主节点。由于双方数据存在并发的可能性，这种模式需要一个共享的集群文件系统，利用分布式的锁机制进行管理，如 GFS 和 OCFS2。

部署双主模式时，DRBD 是负载均衡的集群，这就需要从两个并发的主节点中选取一个首选的访问数据。这种模式默认是禁用的，如果要是用的话必须在配置文件中进行声明。

可参见“启用双主模式”一节，了解有关启用双主模式的信息。

2.3 复制模式

DRBD 提供了三种不同的复制方式，允许三种度的复制：

协议 A：一旦本地磁盘写入已经完成，数据包已在发送队列中，则写被认为是完成的 。在一个节点发生故障时，可能发生数据丢失，因为被写入到远程节点上的数据可能仍在发送队列。尽管，在故障转移节点上的数据是一致的，但没有及时更新。这通常是用于地理上分开的节点。

协议 B：一旦本地磁盘写入已完成且复制数据包达到了对等节点则认为写在主节点上被认为是完成的。数据丢失可能发生在参加的两个节点同时故障的情况下，因为在飞行中的数据可能不会被提交到磁盘。

协议 C：只有在本地和远程节点的磁盘已经确认了写操作完成，写才被认为完成。没有任何数据丢失，所以这是一个群集节点的流行模式，但 I/O 吞吐量依赖于网络带宽。

简言之：

A 数据一旦写入磁盘并发送到网络中就认为完成了写入操作。

B 收到接收确认就认为完成了写入操作。

C 收到写入确认就认为完成了写入操作。

就目前而言应用最多和应用最广泛的为协议 C。

备忘

两个因素影响你对复制模式的选择：保护和延迟。相比之下吞吐量很大决定了对复制模式的选择。

参见“配置你的资源”一节，查看资源配置的复制模式的配置。

2.4 多复制传输

这个特性在 DRBD8.2.7 及以后的版本中可用。

DRBD 的复制和同步框架套接字层支持多个级别的传输：

IPv4 中的 TCP 协议：这是规范的实施方式，也是 DRBD 默认使用的，可用于任何开启 IPv4 协议的系统上。

IPv6 的 TCP 协议：当配置使用标准的 TCP 套接字用于复制和同步时，DRBD 还可使用 IPv6 的网络协议。虽然是不同于 IPv4，但是这是和使用 IPv4 的相当的。

超级套接字：超级套接字取代 TCP/IP 的单一堆栈，单片，高效和 RDMA 能够用套接字实现。DRBD 可使用该延迟较低的套接字类型。超级套接字目前必须运行由单一供应商（Dolphin 互联解决方案）提供的特定硬件设备上。

2.5 高效同步

（重新）同步不同于从设备上复制。复制发生于任何对主节点资源进行写时，同步则往往伴随写。相反，它则影响到整个设备。

不管什么理由：主节点失败、次节点失效还是复制链接被中断，只要复制链接被中断同步就是必须的。在这个意义上同步是有效的，DRBD 不同步修改块而保持块的顺序写，而维持它的线性秩序，这将会导致如下后果：

同步速度很快，因为这几个连续的写操作发生一次块的同步。

为保持块同步在自然磁盘上的区块分布，同步也是伴随着磁盘磁道的寻找。

数据的不一致性，因为在同步的过程中，数据在备用节点上部分过时或者部分被更新，这种数据状态更称为数据不一致性。

一个节点不能投入使用，从而保证数据的不一致性保持尽量的短。数据同步服务是在主节点上是不间断的以后台进程运行着的。

用下面的公式，你可以简单的估算出同步的时间：

公式 2.1。同步时间

t_{sync}就是预期的同步时间。D 是要同步的数据量，你不可能对它有任何的影响的（这个数据由你的应用程序修改而将复制链接打破而产生的），R 是同步的速度，这个是可以配置的，通过对复制的限制而控制网络的吞吐量和 I/O 子系统的。

DRBD 的同步效率可进一步提升对数据的消化，也就是校验。当使用校验同步时，不是强制执行覆盖标记不同的块，而是同步他们之前所读取的块，在磁盘上计算出散列的内容。然后比较这个来自对等节点的 hash 表，如果 hash 表匹配，则忽略重写该区块。这样就可以大大减少用于解决在断开模式下文件系统重写相同内容的同步时间。

参阅“配置同步速率”和“配置基础校验同步”二节的关于同步配置方面的建议。

2.6 在线设备验证

这个特性在 DRBD8.2.5 及以后版本中可用。

在线设备验证，用于可以在节点之间做一个非常有效的方式：逐块逐块得数据完整性检查。

备忘

请注意，“高效”这里是指网络带宽的高效，并保证在任何情况下验证不破坏冗余。在线验证仍是一个资源密集型的操作，会明显提升 CPU 利用率和负载。

它通过一个节点（验证源）对每一个底层的设备的某一资源的块存储设备依次计算出一个加密摘要。接着传输 DRBD 摘要到对等节点（验证目标），对摘要对应的本地副本块进行验证。如果两个摘要不匹配，标示出块不同进行同步。由于 DRBD 传输的只是摘要，而不是完整的块，因此对网络资源的要求比较低，从而提高了网络带宽的利用率。

这个过程被称为在线验证，因为它不要求验证的 DRBD 资源在验证时置为不可用。因此，尽快在线验证需要消耗一部分资源，但是它不会造成服务的中断或者是系统的中断，而且在验证的过程中也不影响后续同步的进行。

在线验证可由常见本地 cron 调度服务调度在线验证管理运行，比如一周一次或者是一个月一次。

参阅“利用在线设备验证”信息一节，查看如何启用、调用和自动在线验证。

2.7 复制传输完整性验证

该特性在 DRBD8.2.0 及以后版本可用。

DRBD 使用 MD5、SHA-1 或者是 CRC-32C 等加密算法对摘要信息进行终端到终端的信息完整性验证。

备忘

DRBD 不提供这些信息摘要的算法，而是由 linux 的内核的 API 提供，DRBD 只是调用。因此 DRBD 利用的任何摘要算法需要系统内核提供支持。

利用该特性，DRBD 对每一个复制到对等节点的数据块都生成信息摘要，对等节点也采用同样的方式对复制的数据块进行完整性验证。如果验证信息不对，则对等节点请求重新发送。这样 DRBD 复制保护可以放置一些错误的资源，所有这些，如果不检查，将可能导致数据在复制的过程产生脏数据：

排位错误（“位翻转”）发生在主存储器和网络接口传输数据到对端时（比较常见的一个事实是数据在没有通过 TCP 校验就被丢到网卡上）

位翻转发生在数据从网络接口传送到接受节点的主存储器中时（原因同样是 TCP 校验丢弃）

因为竞争、网络接口固件或者程序错误所产生的任何行形式的脏数据。

节点之间的被网络组建组装的位翻转或随机脏数据的注入（如果不使用直连而是后端到后端的连接）

参阅“配置完整性复制传输检查”一节了解如果启用复制传输完整性检查。

2.8 裂脑通知和自动修复

在 DRBD8.0 及更高版本，可实现裂脑自动修复。裂脑自动修复在 DRBD0.7 中可用，但是主要采取对新的主节点的修改采取抛弃的方式，而且还是不可配置的。默认情况下自动裂脑修复在 DRBD 中是禁用的。

裂脑通知从 DRBD8.2.1 之后开始可用。

裂脑大部分情况下是由集群节点间的网络连接临时故障、集群软件管理干预或者是人为错误，导致两个节点都切换为主节点而断开连接。这是一个潜在的有害状态，因为它意味着不能复制数据到对等节点，这样就可能导致两个节点的数据产生分歧，产生不可合并的分裂。

备忘

DRBD 的裂脑不同于集群软件的裂脑，如 Heartbeat，是分布式集群的通过集群管理软件管理的主机之间失去了所有的连接。本指南采用如下方式加以区分：

在该段的裂脑指的是 DRBD 的裂脑。

采用集群失去所有连接作为集群裂脑的替代专业术语。
当 DRBD 探测到裂脑时可以设置自动通知（采用邮件或者是其他的方式）。参见“裂脑通知”一节了解更详细的的信息，查看如何配置该特性。

虽然一般情况下建议手工解决裂脑的问题，但是为了彻底的解决裂脑问题，在某些情况下裂脑自动修复还是比较可取的。DRBD 有可利用一下方法解决该问题：

丢弃比较新的主节点的所做的修改。在这种模式下，当网络重新建立连接并且发现了裂脑，DRBD 就会丢弃自切换到主节点后所修改的数据。

丢弃老的主节点所做的修改。在这种模式下，DRBD 将丢弃首先切换到主节点后所修改的数据。

丢弃修改比较少的主节点的修改。在这种模式下，DRBD 会检查两个节点的数据，然后丢弃修改比较少的主机上的节点。

一个节点数据没有发生变化的完美的修复裂脑。在这种模式下，如果其中一台主机的在发生裂脑时数据没有发生修改，则可简单的完美的修复并声明已经解决裂脑问题。需要注意的是，这几乎是不可能发生的一个情况，即使是两台主机上的只挂载文件系统在 DRBD 块设备上（设置是只读的），设备的内容也会被修改，也就排除了自动修复的可能性。

注意

自动裂脑自动修复能不能被接受取决于个人应用。考虑建立一个 DRBD 的例子库。在“丢弃修改比较少的主节点的修改”兴许对 web 应用好过数据库应用。与此相反，财务的数据库则是对于任何修改的丢失都是不能容忍的，这就需要不管在什么情况下都需要手工修复裂脑问题。因此需要在启用裂脑自动修复前考虑你的应用情况。

参阅“自动修复裂脑策略”一章关于配置 DRBD 裂脑自动修复的策略的详细信息。

2.9 对磁盘缓冲的支持

当本地块设备如磁盘或者是 RAID 逻辑磁盘启用高速写缓存时，一旦将写缓存发回出来就告知磁盘写“完成”。控制管制造商称之为回写模式，而相反的则成为 wirte-through。如果在处在回写模式时遇到停电的情况，则可能在发生此之前的回写数据没有来的及写会磁盘，则可能造成数据的丢失。

DRBD 使用磁盘刷新来解决这个问题。磁盘刷新是只有在完成相关数据稳定存储的一个写操作，也就是说刷新是有些的写入磁盘而不是缓存。DRVD 使用磁盘刷新用于写操作完成其复制数据和原数据。实际上，DRBD 绕开写入缓存是非常必要的，如在活动日志更新或隐写后写的执法依赖。这就意味着及时在断电的情况下也能保证其可靠性。

但是比较重要的是，DRBD 的需要设置支持才能使用磁盘刷新。在新近比较合理的内核中支持大多数的 SCSI 和 STAT 设备的磁盘刷新。Linux 软件 RAID（md）中的 RAID-1 也支持磁盘刷新，所有组件所提供的设备也都支持磁盘刷新。这写舍用也同样适用于映射设备（LVM2、DM-raid、多路径）等。

电池供电控制器的高速写缓存（BBWC）使用电池备份他们的挥发性存储。在这些设备上，当电源供电恢复时，在刷新前控制器将电池支持的最近高速写缓存写入磁盘，确保所有的写入由不稳定的存储到稳定的存储上。当这些设备运行在 DRBD 上时，它可能支持禁用磁盘刷新，从而提高 DRBD 写入性能。请参阅“禁用备用设备刷新”一些，查阅更详细的信息。

2.10 磁盘错误处理策略

如果某个节点作为 DRBD 的后端磁盘设备出现故障，DRBD 可能把这个 I/O 错误传递给上层（通常是文件系统），或者 DRBD 可能对上层屏蔽了 I/O 错误。

Passing on I/O errors：如果 DRBD 被配置为 pass on I/O 错误，则任何底层设备的错误都会透明地传递给上层 I/O 层。这样，就由上层来处理错误（这会导致文件系统被重新挂载为 read-only）。这个策略不保证服务持续性，并且对大多数用户来说也不推荐。

Masking I/O errors：如果 DRBD 被配置为 detach 底层 I/O 错误，则 DRBD 将分离错误。这个 I/O 错误被 DRBD 对上层屏蔽，并且 DRBD 透明地通过网络从对端节点提取受影响的数据块。在这种情况下，DRBD 被称为运行在 diskless 模式，并处理所有相应的 I/O 操作，读写实际上都是发生在对端（不是本地）。这种 diskless 运行模式会影响性能，但是服务

将继续运行不受影响，并且可以从容地在一个合适的时间迁移到对端节点。（这个方式有点类似 Soft RAID1，当镜像磁盘发生故障时可以确保应用继续运行并提供恢复机会。）

参考配置 I/O 错误处理策略有关配置 I/O 处理策略的信息。

2.11 过期数据处理策略

DRBD 区别对待不一致（inconsistent）和过期（outdated）数据。

不一致数据是指不能被正确存取并且使用的数据，一般例子是在某个节点上正在进行同步的数据，此时在这个节点上的数据是部分陈旧的，部分又是更新的。这种情况下，如果这个设备处理一个文件系统，这个文件系统是不能挂载的，甚至不能通过自动的文件系统检查。

过 期数据，相反，是指在第二个节点上的数据是一致的，但是长时间没有和主节点进行同步。这种情况发生在复制链路中断，不论是临时还是永久。数据是过期的，而 且断开的第 2 个节点被认为是干净的，只是状态反映的是过去的某个时间。为了避免服务使用过期数据，DRBD 不允许创建一个过期数据状态的资源。

DRBD 有一个接口在一个网络中断时允许一个扩展的应用程序来处理第二节点的过期数据。DRBD 然后会拒绝该节点切换为主节点，这样可以保护应用程序不使用过期数据。一个完整实现这个功能的方案是 Heartbeat cluster management framework（Heartbeat 使用了从 DRBD 复制链路分离的一个通讯通道）。然而，这个接口是通用的，并且可能适合任何集群管理应用程序。

当一个过期资源重新连接了复制链路，则过期状态将自动清除。一个后台同步过程将自动开始。

the DRBD outdate-peer daemon (dopd)介绍了 DRBD/Heartbeat 配置保护的案例。

2.12 三路复制

从 DRBD 版本 8.3.0 开始提供三路复制。

当使用三路复制，DRBD 对一个已存在的 2 节点集群增加了一个第三节点，并复制数据到这个第三节点上，使用这个第三节点用于备份和灾难恢复。

三路复制通过在一个已经存在的处理生产数据的堆栈上增加另一个，基于堆栈的 DRBD 资源，类似以下图示：

这个堆积起来资源是通过异步复制（DRBD protocol A）结合同步复制（DRBD protocol C）来实现的。

三路复制可以永久使用，此时第三节点从生产集群持续更新数据。并且，也可以在需要是建立连接，此时生产集群通常和备份站点断开，周期性地启动 site-to-site 同步（例如在每天晚上 cron 任务）。

这个构架适合远程灾备，对于远程镜像可以采用异步方式。由于生产系统不需要等待远程节点完成就可以继续读写，远程站点可以在后续逐步同步好存储系统。

2.13 使用 DRBD Proxy 实现远距离复制

DRBD 版本 8.2.7 以上支持 DRBD Proxy。

DRBD protocol A 是异步实现的，不过写应用程序仍然会在 socket 输出缓存耗尽后阻塞住（查看 drbd.conf 的 sndbuf-size 选项）。此时，写应用程序将不得不等待直到一些等待数据被缓慢地写入远程系统。

写带宽受限于网络带宽，写入峰值只有在填入有限的 socket 输出缓存才能有效处理。

可以通过 DRBD proxy 的缓存机制来解决这个问题。DRBD Proxy 会吸收所有从主节点发出的 DRBD 数据到它的缓存中。DRBD Proxy 的缓存大小是任意配置的，只受限于地址空间大小和可用的物理内存。

DRBD Proxy 还可以配置在转发数据时是否采用数据压缩和解压缩。DRBD 的数据包压缩和解压缩可能可以轻微减少延迟。

建议在多内核 SMP 系统上使用数据压缩和解压缩。

实际上，多数数据块 I/O 数据压缩是非常好的，可以有效减轻带宽压力，甚至可以使用 DRBD protocol B 和 protocol C。

备忘

DRBD proxy 属于 DRBD 产品线，并且不是以开源 licence 发布，需要购买。

2.14 基于复制的传输

Trunk based replication，称为 disk shipping，是预处理远程站点的数据服务，通过物理递送方式把硬盘（存储）运送到远程站点。通常适合：

需要复制的数据非常巨大（超过数百 G 数据）

数据的变化相对较小

站点间的网络带宽有限

在这种情况下如果不采用物理传递硬盘数据的方法，则 DRBD 初始化同步时间过长，而通过物理传递数据盘，可以大大减少初始化同步时间。

参考 Using trunk based replication 有关使用案例。

2.15 浮动的对等节点

从 DRBD 8.3.2 开始提供浮动对端功能。

一些情况下 DRBD 需要使用浮动对端配置。在浮动对端配置是，DRBD 的对端不是特定命名的主机，而是可以浮动的一系列主机。在这种情况下，DRBD 通过 IP 地址标识对端，而不是主机名。

参考在两个基于 SAN 后端的 Pacemaker 集群间配置 DRBD 复制

第二篇　　安装和配置

目录

3. DRBD 的编译，安装和配置

3.1 由 LINBIT 提供的软件包

LINBIT 是赞助 DRBD 项目的公司，为商业用户提供 DRBD 的二进制包。这些包在 http://www.linbit.com/support/中可以找到，被认为是“官方”DRBD 的基础。

它的发行版本如下：

这些支持以下发行版本：

红帽企业的 linux（RHEL），版本 4 和 5

SUSE Linux 企业操作系统（SLES），版本 9、10 和 11

Debian GUN/linux 操作系统，版本 4.0（etch）和 5.0（lenny）

Ubuntu Server Edition LTS，版本 6.06（Dapper Drake）和 8.04（Hardy Heron）

LINBIT 发布所有 DRBD 的新源码包。

基于 RPM 包（SLES、RHEL）的安装可以通过简单的调用 rpm -i （新安装）或者是 rpm –U （升级）对 DRBD 进行安装。

对于基于 Debian 的系统（GUN/Linux，Ubuntu），Drbd8-utils 和 Drbd8-module 包的安装使用 dpkg -i 或者是 gdebi（如果可用的话）。

3.2 由发行商提供的软件包

相当数量的发布包包括 DRBD，包括二进制的预编译包。支持这些编译，如果有的话就是由发行商提供相关发布，他们释放的周期往往落后于 DRBD 源码的发行。

3.2.1SUSE Linux 企业服务器（SLES）

在 SLES7 和 10 中包括 DRBD0.7，在 SLES11 高可用扩展中包含 DRBD8.2。

在 SLES，DRBD 通常是通过安装软件的安装组件 YaST2,捆绑在“高可用”软件包中供选择。

用户通常使用简单的命令行安装：

yast -i drbd 或者 rug install drbd

3.2.2Debian GUN/Linux

自动 5.0（Lenny）中包含 DRBD8，从 Debian 3.1（sarge）中包含 DRBD0.7。

在 Lenny 中（现在已将 DRBD 预编译在内核中，不在需要使用 module-assistan），可以通过如下安装 DRBD：

apt-get install drbd8-utils drbd8-module

在 Debian3.1 和 4.0，则必须通过如下命令安装：

apt-get install drbd0.7-utils drbd0.7-module-source build-essential module-assistant

module-assistant auto-install drbd0.7

请参阅“创建 DRBD 的 Debian 包”一节，有关涉及安装以及 module-assistant 的详细资料。

3.2.3 在 CentOS

CentOS 5 中包含 DRBD8，CentOS 4 中包含 DRBD0.7

DRBD 可以通过 yum 命令进行安装（需要注意的是提要支持软件仓库）

yum install drbd kmod-drbd

3.2.4 在 Ubuntu

Ubuntu7.10（gusty gibbon）包含 DRBD8，Ubuntu6.06（Dapper Drake）中包含 DRBD0.7。可以通过激活 Ubuntu 中的镜像/etc/apt/sources.list，执行如下命令获取 DRBD：

apt-get update

apt-get install drbd8-utils drbd8-module-source build-essential module-assistant

module-assistant auto-install drbd8

警告

Ubuntu 6.10（Edgy EFT）和 7.04（Feisty Fawn）都包含 DRBD8 的预发行版，这是从未在生产系统中正式使用的版本。Ubuntu 也包含 DRBD0.7，但是生产可适当的用（但是版本已经过期）。

4 源码编译和安装 DRBD

4.1 下载 DRBD 源码

DRBD 的历史版本和当前版本的压缩包都可以从 http://oss.linbit.com/drbd/上下载。按照惯例，DRBD 的源码包以 drbd-x.y.z 的方式命名，其中想，x,y,z 对应的为主、辅和错误的修正版本号。

DRBD 压缩包的大小不超过 0.5M。要下载并解压到当前工作目录，可使用如下命令：

wget http://oss.linbit.com/drbd/8.3/drbd-8.3.4.tar.gz

tar -xzf drbd-8.3.4.tar.gz

备忘

上面 wget 下载的源码只是一个例子，当然你可以下载你喜欢的版本。

建议将 DRBD 放到存放源代码的普通目录中，如/usr/src 或者是/usr/local/src，本指南中假定在/usr/src 目录中。

4.2 从 DRBD 源码库中签出源码

DRBD 的源码保存在一个公共的 Git 库中，可以在 http://git.drbd.org/上浏览。要从库中签出 DRBD 的源码，就必须将 DRBD 释放到库中。这本例中签出的为 DRBD8.3：

git clone git://git.drbd.org/drbd-8.3.git

如果您的防火墙不允许 TCP 连接到 9418 端口，您也可以通过 HTTP 签出（请注意，使用 http 协议要比 Git 签出的方式慢的多，因此建议尽量的使用 Git 签出）。

git clone http://git.drbd.org/drbd-8.3.git

先创建一个命名为 drbd-8.3 的 Git 的签出目录，也就是将命令的文件夹的名字和签出的源代码的名字一致，可使用以下命令：

cd drbd-8.3

git checkout drbd-8.3.x

这里的 X 就代表你想签出的 DRBD 的建立点。

签出的目录是和解压后的指定版本的源代码一样的，也就就是使用 DRBd 的源代码了。

备忘

实际上源代码解压和 git 签出两个来源之间还是有些细微的差别的：

git 签出包含一个 debian/subdirectoy，而源代码解压却没有。这是因为 debian 版本的维护者要求加入的。

源代码的 tarball 包含帮助手册，而 git 不包含。虽然 docbook 不是源码必须的，但是 git 签出需要一个完整的 docbook 手册的地址。

4.3 从源码编译 DRBD

4.3.1 检查编译的条件

在编译 DRBD 的源码前，机器必须符合一下条件：

make，gcc：glibc 的开发包，是 flex scanner generator 必须安装的。

备忘

必须保证使用的 gcc 编译模块运行在内核上，如果系统中有多个版本的 gcc，需要选择特

定版本的 gcc。

如果直接从 git 签出的文件中编译，guncutoconf 也是必须的。这个如果从源码编译这是不需要的。

如果选择运行一个 stock kernel，就需要安装一个和之相匹配的预编译的内核头文件包。这些通常命名为 kernel-dev、kernel-headers、linux-headers 或者类似的名字。这种情况下，可以跳过“准备内核源码树”一节，而进行“准备 DRBD 的编译树”一节。

如果选择的不是 stock kernel 的（比如：系统运行在一个从源码编译配置的内核上），那么内核的源文件则必须安装，则可编译安装和内核源代码版本匹配或者是相似的软件包。

备忘

在以 rpm 为基础的系统上，这些软件包命名类似于 kernel-source-version.rpm 的形式，这就很容易和 kernel-version.src.rpm 相混淆。而前者则是正确的 DRBD 编译安装包。

在 kernel.org 上“Vanlilla”内核包命名的格式为 linux-version-tar.bz2 并应该被解压到 /usr/src/linux-version 中，使用连接的方式指向/usr/src/linux 的文件夹。

在这种情况下，编译 DRBD 对应的内核源码，则必要进行“准备内核源码树”一节。

4.3.2 准备内核源码树

必须在解压后的内核源码所在的目录准备建立 DRBD 的源代码树，通常情况下，这个目录为/usr/src/linux-version，或者是就是一个连接名为/usr/src/linux 的目录。

```
cd /usr/src/linux
```

接下来的为建议的步骤而不是必须的。务必复制现有的.config 文件到一个安全的位置，在这一步的基础上初始化内核源码树，消除之前编译和配置的影响：

```
make mrproper
```

接下来就可以复制当前运行的配置文件到内核源码树，下面有些选项需要选择：

通过/proc 将最近内核的合理配置输出到当前正在运行的内核配置文件中，可以进行复制，如下：

zcat /proc/config.gz > .config

在 SUSE 的内核上包含一个 cloneconfig target，所在这这类系统上，操作如下：

make cloneconfig

一些安装将内核配置文件安装到/boot 下，这就需要：

cp /boot/config-`uname -r` .config

最后，你可以将已经配置好的文件拷贝到当前运行的内核中运行。

4.3.3 准备 DRBD 的编译树

任何 DRBD 的编译都需要先配置 DRBD 的源码树和脚本：

备忘

本节中所提到的适用于 DRBD8.3.6 及以上的版本。一直到 DRBD8.3.5 还没有配置脚本。

当编译 git 签出的源码时，配置脚本也是不存在的，因此必须先签出它才行。

可以使用—help 查看配置帮助文件列表。在下面的表格中总结了些比较重要的选项：

表 4.1 DRBD 所支持的配置选项

选项	描述	默认	备注
--prefix	安装路径的选项	/usr/local	这是默认的文件系统标准的无包装软件的安装路径，包装软件一般覆盖/usr
--localstatedir	目录状态定位路径	/usr/local/var	即使有默认的 prefix，但大多数用户还是想重写/var
--sysconfdir	系统配置路径	/usr/local/etc	即使有默认的 prefix，但大多数用户还是想在/etc 下
--with-km	编译 DRBD 的内核	无	在编译 DRBD 内核模块时开启此

选项	描述	默认	备注
	模块		选项
--with-utils	编译 DRBD 集成 userland 工具	yes	当编译一个新的内核版本的 DRBD，而不同时升级 DRBD，禁用次选项
--with-heartbeat	编译 DRBD 集成 Heartbeat	yes	如果不适用 DRBD 的 heartbeat v1 资源代理或 dopd，则可禁用该选项
--with-pacemaker	编译 DRBD 集成 Pacemaker	yes	如不打算启用 Pacemaker 功能，则可禁用该选项
--with-rgmanager	编译 DRBD 集成红帽集群套件	no	如想适用 DRBD 整合红帽几圈套件的 rgmanager，这需要启用该选项
--with-xen	编译 DRBD 集成 Xen	yes (在 X86 平台上)	如果不打算集成 Xen 模块，则可禁用该选项
--with-bashcompletion	编译可编程的 bash**drbdadm**	yes	如果使用其他的 bash 的 shell，或者不想利用可编程的 drbdadm，这可禁用该选项
--enable-spec	创建一个 rpm spec 文件	no	仅对包的编译者：如果想创建一个 rpm spec，可启用该选项。见“编译 DRBD 的 RPM”一节

配置脚本将建立使用符合需求的DRBD。它通过自动检测所调用的参数对DRBD进行配置，因此在配置相应的参数时，要小心重写默认的选项。

配置脚本在被调用的目录中创建一个日志文件名为 config.log。在记录编译时的问题时，也比较智能话的记录整个编译的过程。

4.4DRBD 的 rpm 包的创建

备忘

本节中所提到的适用于 DRBD8.3.6 及以上的版本。一直到 DRBD8.3.5 还是用不同的 rpm

安装的方法。

DRBD 的安装包含一个直接以安装 rpm 出 DRBD 的源代码树体系。为创建 rpm 包，这部分叫做“检查创建的前提条件”适用于以上同样 make 安装的方式，除非你还需要 rpm 的创建工具。

另外，如果你是创建与内核中可用的预编译头的方式，可以参见“准备内核源代码树”一节。

有两种方案创建 rpm 包。简单的方法是简单在顶层 makefile 调用 rpm 目标文件：

$ ./configure

$ make rpm

$ make km-rpm

这种方法是将自动从预先定义的模板生成 spec 文件，然后利用这些模板文件，建立二进制的 rpm 包。

生成 Rpm 的方式生成大量的 rpm 包：

表 4.2。 DRBD 技术的 userland RPM 包

包名	描述	依赖关系	备注
drbd	DRBD 元软件包	其他所有的 drbd-* 软件包	顶层的虚拟包。安装时，在所有其他的 userland 作为依赖包中提取。
drbd-utils	二进制管理使用程序	DRBD 启用需要	
drbd-udev	udev 的集成设施	drbd-utils, udev	启用 udev 来管理用户友好连结到 DRBD 设备
drbd-xen	Xen DRBD helper scripts	drbd-utils, xen	Enables xend to auto-manage DRBD resources
drbd-heartbeat	DRBD Heartbeat	drbd-utils,	Enables DRBD

包名	描述	依赖关系	备注
	integration scripts	heartbeat	management by legacy v1-style Heartbeat clusters
drbd-pacemaker	DRBD Pacemaker integration scripts	drbd-utils, pacemaker	Enables DRBD management by Pacemaker clusters
drbd-rgmanager	DRBD Red Hat Cluster Suite integration scripts	drbd-utils, rgmanager	Enables DRBD management by rgmanager, the Red Hat Cluster Suite resource manager
drbd-bashcompletion	Progammable bash completion	drbd-utils, bash-completion	Enables Programmable bash completion for the drbdadm utility

另外一种是更灵活的方法是配置并生成 spec 文件，你可以进行任何你所需要的更改，然后使用 rpmbuild 命令生成：

```
$ ./configure --enable-spec

$ make tgz

$ cp drbd*.tar.gz `rpm -E _sourcedir`

$ rpmbuild -bb drbd.spec
```

如果你要建立 drbd 的公用用户空间和内核模块的 rpm，使用：

```
$ ./configure --enable-spec --with-km

$ make tgz

$ cp drbd*.tar.gz `rpm -E _sourcedir`

$ rpmbuild -bb drbd.spec

$ rpmbuild -bb drbd-km.spec
```

这种方式将被创建在 rpm 配置（或个人的 rpmmacros 配置）的地方

当创建这些软件包后，你可以安装、升级和卸载系统中的任何其他的 rpm 包。

请注意任何内核的升级都需要生成心的 drbd-km 包以匹配新的内核。

当升级 drbd 到一个新版本时，只需要创建新的 drbd userland 包。然而当升级新的内核和新的 drbd 版本时，则需要同时升级包。

4.5 创建 DRBD 的 debian 包

DRBD 创建系统包含一个直接从 DRBD 源代码树建立出的 Debian 包。可参见“检查建立的前提条件”一节建立 debian 软件包，除了需要 dpkg-dev 包含 debian 软件包工具和 fakeroot 包（如果想非 root 用户建立 DRBD—强烈推荐），基本上和建立并使用 make 安装的方式是一样的。

另外，如果你是创建与内核中可用的预编译头的方式，可参见“准备内核源代码树”一节。

DRBD 源代码书包含一个 debian 软件包所需文件的 debian 子目录。但是这个子目录并不在 DRBD 的 tar 包内，因此如果需要时则需要创建一个特定的 git，将其签出。

可以使用一下命令签出 DRBD 的 debian 软件包：

dpkg-buildpackage -rfakeroot -b -uc

备忘

如编译 drbd 的软件包使用非 root 用户（-rfakeroot）以二进制的方式（-b），并禁用变更文件的签名。其他的编译选项可以参见 dpkg-buildpackage 手册。

这个创建的过程将创建两个 debian 的软件包：

1. 命名为 drbd8-utils_x.y.z-BUILD_ARCH.deb，drbd 的用户控件工具

2. 命名为 drbd8-module-source_x.y.z-BUILD_all。Deb，源代码匹配包。

当你创建这些软件包后，就可以安装、升级或者是卸载他们，就像其他的 Debian 软件包一样。

通过 debian 的模块儿辅助功能，从源码包中编译和安装 drbd 的实际的内核模块儿是比较容易的：

module-assistant auto-install drbd8

也可以使用上面命令的简写模式：

m-a a-i drbd8

需要注意的是任何内核的升级都需要重新编译其内核模块儿（如模块辅助）来匹配新的内核。相比之下 Drbd8-utils 和 drbd8-module-source 源码包，则只需要升级到新的 drbd 版本。无论是升级到新的内核还是新的 drbd 版本，都需要升级以上两个软件包。

5 配置 drbd

5.1 准备底层存储

安装过 drbd 之后，就必须在集群的两个节点上预留一个大小一致的存储区域。这将是 drbd 的底层的设备。可以使用系统中任何型号的块设备达到次目的。典型的例子包括：

一个磁盘分区（或一个完成的物理磁盘）

软 RAID 设备

LVM 逻辑卷或者其他有 linux 的 device-mapper 基础块设备

EVMS

服务器上的其他任何类型的块设备。DRBD8.3 及以上的版本，也可以使用资源堆放，也就意味着可用一个 drbd 设备为其他的一个底层的设备，一些特殊的也可以考虑资源堆放，可参见“创建一个三节点设置”了解详细的配置介绍。

备忘

DRBD 虽然也可以使用 loop 设备作为底层的设置，但是因为其有死锁的问题，因此并不推荐。

创建 drbd 资源前并不要求该存储区是空的。事实上，它是一种常见从以前的两个非冗余单节点的服务器，使用 drbd 创建两个节点的集群。（如打算这样做的话，请参阅“DRBD 源数据”一节，了解一些注意事项。）

在本指南中，我们假设为一个很简单的设置：

两台主机都有空闲的分区/dev/sda7

我们正在使用内部元数据

5.2 准备网络配置

建议运行 DRBD 复制在专用链接环境下，虽然没有严格的要求。本指南建议最合理的选择为直接的、千兆 G 太网络连接。如果你通过交换里连接 DRBD，建议使用冗余组建和 linux 绑定却东程序（主动备份模式）。

一般情况下不建议通过路由器运行 drbd 复制，因为通过路由器的方式表现出明显的缺点（影响吞吐量并造成延迟）。

考虑本地防火墙的因素，drbd 建议采用单独的、可配置 TCP7788 以上的端口，但不变的是 TCP 端口上侦听的 TCP 资源。DRBD 使用两个单独的 TCP 连接（任何方向之一）对资源进行配置。对于正确的使用 DRBD 功能，必须让防火墙运行这些连接的进行。

比防火墙更安全的方式为强制访问控制（MAC），如启动 selinux 或者是 appmrmor。因此需要调整本地的安全策略来适应 DRBD 功能的运行。

自然需要保证 DRBD 所使用的端口没有被其他的程序所占用。

备忘

DRBD 资源不可能支持多个 TCP 连接。如果想让 DRBD 连接实现负载均衡或者是冗余，建议在比较容易实现的以太网级别进行（再次使用 bongding 设备）。

在本指南中，我们假设为一个很简单的设置：

两个 DRBD 主机分别有一个未启用的网络接口 eth1，给他他们分别配置 ip 地址 10.1.1.31 和 10.1.1.32.

在两台主机上都没有其他程序占用 TCP 端口 7788 和 7799

本地防火墙配置允许两台机器通过这些 TCP 端口的进和出

5.3 资源的配置

DRBD 的所有的控制都是在配置文件/etc/drbd.conf 中。通常情况下配置文件包含如下内容：

include "/etc/drbd.d/global_common.conf";

include "/etc/drbd.d/*.res";

通常情况下，/etc/drbd.d/global_common.conf 包含 global 和 common 的 DRBD 配置部分，而.res 文件都包含一个资源的部分。

在一个单独的 drbd.conf 文件中配置全部是可以实现的，但是占用的配置很快就会变得混乱，变得难以管理，这也是为什么多文件管理作为首选的原因之一。

无论采用哪种方式，需必须保持在各个集群节点的 drbd.conf 以及其他的文件完全相同。

在 drbd 的源代码包的脚本子目录中包含配置文件的样例。如二进制安装包直接将其安装在/etc 或者是包中特定的样例目录/usr/share/doc/packages/drbd 中。

备忘

DRBD 资源不可能支持多个 TCP 连接。如果想让 DRBD 连接实现负载均衡或者是冗余，建议在比较容易实现的以太网级别进行（再次使用 bongding 设备）。

本节只介绍这些为了保证 drbd 运行的必须了解的配置文件相关的方面。配置文件的语法和内容都相继的记录在 drbd.conf（5）中。

5.3.1 配置样例

在本指南中，我们假设为一个很简单的设置：

global {

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

该配置样例做一下详解：

选择参与 drbd 的使用情况统计（见下文）

资源配饰使用完全同步复制协议（Protocol C），除非另有明确指定。

集群包含两个节点。Alice 和 Bob

资源命名为 r0，并使用底层设备/dev/sda7，并与内部原数据配置资源

资源使用网络 TCP 连接端口 7789，并分别绑定在 ip 地址 10.1.1.31 和 10.1.1.32 上

5.3.2global 部分

在配置文件中这部分只允许出现一次，他通常在/etc/drbd.d/global_common.conf 文件中。在单一的配置文件中，他应该置顶。在本节提供的几个选项中，只有一个是大多数用户相关。

usage-count.DRBD 用于统计应用各个版本的信息。当新的版本的 drbd 被安装就会和 http server 进行联系。当然也可以禁用该选项，默认情况下是启用该选项的。

备忘 **5.3.3Common 部分**

DRBD 使用统计数据是公开的，可以参见 http://usage.drbd.org 该本分提供一个快捷的方式来定义配置每个可继承的资源，它通常在/etc/drbd.d/gloabal_common.conf 中。可以在定义的每一个资源的基础上定义任意选项。

如果使用多个资源的话，该部分并不是严格要求但是是强烈建议的。否则的话配置文件很快就会变的让人难以理解了。

在上面的例子中，我们使用 Protocol C，所以在每一个资源配置（包括 r0）基础该选项，除非你明确配置使用另外的协议选项。对其他的同步协议选项，可以参见“复制模式”一节。

5.3.4Resource 部分

每个资源的配置通常都命名为/etc/drbd.d/resource.res。所有的 drbd 资源必须通过指定的配置方式进行命名，可以使用 US-ASCii 内的除空格外的任意的标示符。

每一个资源配置都必须有两个主机上的分节（一个节点上一个）。

所以的配置设置都继承共同的部分（如果存在的话），或者是从 DRBD 默认设置中继承。

事实上，你也可以使用主机分节上资源部分的一个速记符号，也可以指定每个选项的值在量主机上一样。这样，和上面的例子比我们可以进一步压缩该部分：

```
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;
  }
}
```

这个符号在 drbd8.2.1 及以上版本可用。

5.4 首次启用资源

如上一节所述完成对资源的配置后即可启用资源。

备忘

以下步骤必须在两个节点上完成

1. 创建元数据设备。这步只有先创建设置然后才能初始化设备：

```
drbdadm create-md resource
v08 Magic number not found
Writing meta data...
initialising activity log
NOT initialized bitmap
```

New drbd meta data block sucessfully created.

success

2. 附加到备份设置。这步将 drbd 资源和后端设备连接

drbdadm attach resource

3. 设置同步参数。这步设置 drbd 资源的同步参数

drbdadm syncer resource

4. 4 连接对等节点。这一步连接对等节点的资源

drbdadm connect resource

提示

必须注意操作的步骤为 drbdadm attach、drbdadm syncer 、drbdadm connect，不然可能导致 drbd 崩溃。

5. 查看/proc/drbd。/proc/drbd 为 Drbd 的虚拟状态文件，包含如下类似的信息：

cat /proc/drbd

version: 8.3.0 (api:88/proto:86-89)

GIT-hash: 9ba8b93e24d842f0dd3fb1f9b90e8348ddb95829 build by buildsystem@linbit, 2008-12-18 16:02:26

1: cs:Connected ro:Secondary/Secondary ds:Inconsistent/Inconsistent C r---

ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:200768

这两个节点上出现了 Inconsistent/Inconsistent（不一致/不一致）的状态。

到这里，drbd 已经成功的分配好磁盘和网络资源，为运行做好了准备。但是他还不知道使用那个节点作为初始化设备的同步源。

5.5 初始化设备同步

Drbd 的运行还需要以下两个以下的步骤：

1. 选择一个初始同步源。如果是新初始化的或者是空盘时，这个选择是任意的。但是如果其中的一个节点已经在使用并包含有用的数据，那么选择哪个节点同步源就是至关重要的。如果选择了错误的初始化同步方向，就会造成数据丢失，因此需要十分的小心。

2. 启动初始化化完全同步。这一步只能在初始化资源配置的一个节点上进行，并作为同步源选择的节点上。发出如下命令支持此步骤：

drbdadm -- --overwrite-data-of-peer primary resource

执行此命令后，初始化同步开始进行。可以通过查看/proc/drbd 检测同步的进展情况。同步的时间和设备的大小有一定的关系。

到现在为止，drbd 的设置已经全面投入使用，甚至是在初始化同步完毕之前（这期间性能可能有所降低）。现在可以创建一个文件系统作为 raw 的存储块设备，然后挂载它，之后就可执行任何的其他的操作了。

现在可进行第 6 章，了解常见的任务管理方面的知识，这部分将描述常见的资源的管理任务。

5.6 基于复制的传输

为跳过初始化设备同步，为远程节点预置数据并保持同步，请参照以下步骤进行操作。

备忘

假设本地节点已经配置，但是主节点的 drbd 资源断开了连接。也就是说设备配置已经完成，相同的配置文件 drbd.conf 存在在两个对等的节点上，也发出了进行初始化本地节点的命令，但是远程节点尚未连接。

1．在本地节点发出如下命令：

drbdadm new-current-uuid --clear-bitmap <resource>

2．创建一个一致的、逐字拷贝的数据的资源以及其元数据。例如，从 raid1 中删除一个支持热插拔的磁盘，然后将其更换为一个新的磁盘，并重建 raid 以保证持续的冗余。被删除未一个可以撤掉的逐字复制的设备。如果本地的磁盘支持快照副本（如基于 LVM 的 DRBD），你可以使用 dd 创建一个逐位复制的快照。

3．在本地节点的问题：

drbdadm new-current-uuid resource

请注意，这里没有调用 --clear-bitmap 选项

4．副本物理传输到远程对等节点。

5．新增副本到远程节点。这可能会再次造成物理磁盘的堵塞，逐位拷贝磁盘现有的数据到远程节点上。

一定要保证恢复或者是拷贝的数据不仅仅是复制的数据，而且还需要相关的 drbd 的元数据。如果没有这样做的话，磁盘的传输是没有意义的。

6．启用远程节点的资源：

drbdadm up resource

当两个对等节点连接后，不会启动进行完整的同步，相反，自动同步只包含哪些因为 drbdadm 调用而改变的块-- --clear-bitmap new-current-uuid.

即使自此之后没有发生任何的变化，由于涉及活动日志的更新等，也有可能出现一个简短的同步期。这些可以通过使用基于校验的同步而减轻同步。

提示

无论资源是普通的资源还是堆放的资源，都可以使用相同的过程。对于堆放的 DRBD 资源，需要通过启动参数 –S 或者是 –stacked 选项。

第三篇 利用 drbd 工作

6 常见的管理任务

目录

6.1 检查 DRBD 的状态

6.1.1 通过 drbd-overview 查看 DRBD 的状态

查看 drbd 状态最方便的方式是通过 drbd-overview 查看。

drbd-overview

```
  0:home               Connected Primary/Secondary
  UpToDate/UpToDate C r--- /home         xfs   200G 158G 43G   79%
  1:data               Connected Primary/Secondary
  UpToDate/UpToDate C r--- /mnt/ha1      ext3 9.9G 618M 8.8G 7%
  2:nfs-root           Connected Primary/Secondary
```

UpToDate/UpToDate C r--- /mnt/netboot ext3 79G 57G 19G 76%

6.1.2 从/proc/drbd 中查看状态信息

/proc/drbd 是一个虚拟的文件，用于显示当前配置的所有 drbd 资源的实时状态，可是使用以下命令查看该文件的内容：

cat /proc/drbd

version: 8.3.0 (api:88/proto:86-89)

GIT-hash: 9ba8b93e24d842f0dd3fb1f9b90e8348ddb95829 build by buildsystem@linbit, 2008-12-18 16:02:26

0: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r---

ns:0 nr:8 dw:8 dr:0 al:0 bm:2 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0

1: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r---

ns:0 nr:12 dw:12 dr:0 al:0 bm:1 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0

2: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r---

ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0

第一行显示的为当前系统使用的 drbd 的版本，第二行包含一些特定的编译信息。

在本例中的其他行为重复显示的每一个 drbd 设备的配置信息，设备的次要编号。在这里是 0，对应的设备则为/dev/drbd0。

/proc/drbd 包含了其他的一些具体的各种资源的信息：

CS（connnection state 连接状态）。网络连接状态，可参见“6.1.3”章节查看有关各种连接状态的详细信息。

RO（role 角色）。节点的角色。本地节点的角色显示在前，斜线后面的为对等节点的角色。详细可参阅“6.1.4”章节有关资源角色的详细信息。

DS（disk stats 磁盘状态）。磁盘状态，斜线前的为本地节点的磁盘状态，斜线后的为对等节点的磁盘状态。详细可参阅“6.1.5”章节有关各种磁盘状态的详细信息。

Replication protocol 复制协议。资源使用的复制协议，不外乎 A/B/C。详细可参阅“2.3”节有关复制协议。

I/O 标示。六个状态标示反映出资源的 I/O 状态。详细可参见“6.1.6”章节了解有关标示的详细信息。

Performance indicators 绩效指标。一些反映资源利用率和性能的计数器和仪表。详细可参见“6.1.7 绩效指标”章节。

6.1.3 连接状态

可以通过查看/proc/drbd 或者是使用命令 drbdadm cstate 查看资源的连接状态：

drbdadm cstate <resource>

Connected

一个资源可能有以下连接状态之一：

StandAlone 独立的：网络配置不可用。资源还没有被连接或者是被管理断开（使用 drbdadm disconnect 命令），或者是由于出现认证失败或者是裂脑的情况。

Disconnecting 断开：断开只是临时状态，下一个状态将是 StandAlone 独立的。

Unconnected 悬空：是尝试连接前的临时状态，可能的下一个状态为 WFconnection 和 WFReportParams。

Timeout 超时：与对等节点连接超时，也是临时状态，下一个状态为 Unconected 悬空。

BrokerPipe：与对等节点连接丢失，也是临时状态，下一个状态为 Unconected 悬空。

NetworkFailure：与对等节点失去连接后的临时状态，下一个状态为 Unconected 悬空。

ProtocolError. 与对等节点失去连接后的临时状态，下一个状态为 Unconected 悬空。

TearDown 拆解：临时状态，对等节点连接关闭，下一个状态为 Unconected 悬空。

WFConnection.等待和对等节点建立网络连接。

WFReportParams：已经建立 TCP 连接，本节点等待从对等节点传来的第一个网络包。

Connected 连接：Drbd 已经建立连接，数据镜像现在可用，节点处于正常状态。

StartingSyncS：完全同步，有管理员发起的刚刚开始同步。未来可能的状态为 SyncSource 或 PausedSyncS。

StartingSyncT：完全同步，有管理员发起的刚刚开始同步，下一状态为 WFSyncUUID。

WFBitMapS：部分同步刚刚开始，下一步可能的状态：SyncSource 或 PausedSyncS。

WFBitMapT：部分同步刚刚开始，下一步可能的状态：WFSyncUUID。

WFSyncUUID：同步即将开始，下一步可能的状态：SyncTarget 或 PausedSyncT。

SyncSource：以本节点为同步源的同步正在进行。

SyncTarget：以本节点为同步目标的同步正在进行。

PausedSyncS：以本地节点是一个持续同步的源，但是目前同步已经暂停。可能是因为另外一个同步正在进行或者是使用命令 drbdadm pause-sync 暂停了同步。

PausedSyncT：以本地节点为持续的同步目标，但是目前同步已经暂停，这可能是因为另外一个同步正在进行或者是使用命令 drbdadm pause-sync 暂停了同步。

VerifyS：以本地节点为验证源的线上设备验证正在执行。

VerifyT：以本地节点为目标源的线上设备验证正在执行。

6.1.4 资源角色

可以通过查看/proc/drbd 或者是使用命令 drbdadm role 查看资源角色：

drbdadm role <resource>

Primary/Secondary

本地节点一般显示在前，对等节点资源显示在后。

资源的角色一般为如下之一：

Primary 主：资源目前为主，并且可能正在被读取或者写入。若不是双主模式被激活，这种角色只可能出现在两节点中的一个上。

Secondary 次：资源目前为次。正常接收对等节点的更新（除非运行在断开模式下才不是），但是它既不能被读取也不能被写入。这种角色只能是两节点中的一个。

Unknown 未知：资源角色目前未知。本地资源不会出现这种状态，只有对等节点在断开模式下才会出现在这种情况。

6.1.5 磁盘状态

可以通过查看/proc/drbd 或者是使用命令 drbdadm role 查看资源的磁盘状态：

drbdadm dstate <resource>

UpToDate/UpToDate

本地节点一般显示在前，对等节点资源显示在后。

本地和对等节点的磁盘状态都有可能是一下状态之一：

Diskless 无盘：本地没有块设备分配给 DRBD 使用，这就意味着没有可用的后背设备，或者使用 drbdadm 命令手工分离或者是底层的 I/O 错误导致自动分离。

Attaching：读取元数据时候的瞬间状态。

Failed 失败：本地块设备报告 I/O 错误的下一个状态。其下一个状态为 Diskless 无盘。

Negotiating：在已经连接的 DRBD 设备进行 Attach 读取元数据前的瞬间状态。

Inconsistent：数据是不一致的，在两个节点上（初始的完全同步前）这种状态出现后立即创建一个新的资源。此外，在同步期间（同步目标）在一个节点上出现这种状态。

Outdated：数据资源是一致的，但是已经过时。

DUnknown：当对等节点网络连接不可用时出现这种状态。

Consistent：一个没有连接的节点数据一致。当建立连接时，它决定数据是 UpToDate 或者是 Outdated。

UpToDate：一致的最新的数据状态，这个状态是正常的状态。

6.1.6I / O 的状态标志

I/O 状态标志包含在/proc/drbd 中的有关 I/O 操作的当前资源的信息。I/O 状态表示一共六种，具体信息如下：

1. I/O suspension，r 表示运行 I/O，s 表示 suspended I/O，正常情况下为 r

2. Serial resynchronization 串行同步：当资源等待同步时却推迟了重新同步，这个标志变成 A，正常情况为 –

3. Peer-initiated sync suspension 对等节点发起的同步暂停：当资源正在等待重新同步时，对等节点因为一些原因发起暂停同步。这个标志变成 p。正常为 –

4. Locally initiated sync suspension 本地发起的同步暂停：当资源等待重新同步时，在本地节点上用户发起同步暂停，这个标志变成 u。正常为 –

5. Locally blocked I/O 。正常为 -。可能会是一下标志：

d：如因为一个短暂的磁盘状态导致的 drbd 内部出现 I/O 阻塞

b：备用设备的 I/O 阻塞

n：网络 socket 出现阻塞

a：设备的 I/O 阻塞和网络阻塞的组合

6. Activity Log update suspension 活动日志更新暂停：当活动日志更新暂停，这个标志变成 s。正常为 –

6.1.7 绩效指标

/proc/drbd 第二行包含每个资源的计数器和仪表状况的信息：

NS（network send 网络发送）：net 数据以 kibyte 通过网络连接发送到对等节点

NR（network receive 网络接收）：通过网络连接以 kibyte 接收 net 数据

DW（disk write 磁盘写入）：net 数据以 kibyte 写入本地磁盘

DR（disk read 磁盘读取）：net 数据以 kibyte 读取本地磁盘数据

AL（activity log 活动日志）：活动日志区的元数据更新

BM（bit map 位图）：位图区元数据更新

LO（local count 本地计数）：DRBD 请求的开放本地 I/O 子系统的数量

PE（pengding 待定）：发送到对等节点但是尚未节点对方回答的请求的数目

UA（unacknowledged 未确认）：通过网络连接接收到对等节点的请求，但是尚未得到回复

AP（alllication pending 应用程序挂起）：数据块 I/O 请求转发到 DRBD，但是 DRBD 尚未回答

EP（epochs）：一定数量的 epoch 对象，通常为 1，使用阻碍或者是没有排序写的方法时可能增加 I/O 负载。

WO（write order 写入顺序）目前使用的写入顺序的方法：b(barrier 障碍)、f（flush 刷新）、d(drain 漏)或者是 n（none 无）

OOS（out of sync）：以 kububyte 同步当前存储

6.2 启用和禁用资源

6.2.1 启用资源

通常情况下，所有配置的 DRBD 资源自动启用。

在其自由裁量权的范围内管理集群资源或者是根据集群的配置管理

在系统启动时使用/etc/init.d/drbd 脚本

不管什么理由，如果你要手动启用资源，你可以通过如下名下进行操作：

drbdadm up <resource>

当然你也可使用 all 这个关键字，而不是特定的资源名称来启动/etc/drbd.conf 定义的所有资源。

6.2.2 禁用资源

可以通过命令暂时禁用特定的资源：

drbdadm down <resource>

这里同样可以使用 all 这个关键字暂时停用所有在/etc/drbd.conf 配置的所有资源。

6.3 重新配置资源

DRBD 运行在操作的过程中重新配置资源。

要保证对资源的任何必要的修改都配置在/etc/drbd.conf 中

在两个节点中同步/etc/drbd.conf

在量节点上都需要使用 drbdadm adjust <resource>

Drbdadm adjust 后需要使用 drbdsetup 配置进行必要的调整。当然你可以通过运行 drbdadm -d(dry-run)选项检查 drbdsetup 的调用。

备忘

当调整/etc/drbd.conf 中常见的部分，可以使用命令 drbdadm 调整所有运行中的资源。

6.4 升级和降级资源

可以通过手工发出如下命令更改资源的角色，或者从次切换为主或者是从主到次：

drbdadm primary <resource>

drbdadm secondary <resource>

在单主模式下（DRBD 默认），两个节点同时处于连接状态下，任何一个节点都可以在特定的时间内变成主。当在一个节点上运行 drbdadm primary <resource>后再另外一个节点又运行 drbdadm primary <resource>，这样就会出现错误。

但是在双主模式下，可以将两个节点上的资源的角色都可以是主。

6.5 启用双主模式

双主模式下允许一个资源同事承担两个节点上的主。这样做可能是出于永久或者临时的基础上考虑。

备忘

双主模式要求资源配置为同步复制（协议 C）

6.5.1 永久双主模式

如若启用双主模式，则需要在资源配置中指定 allow-two-primaries 为 yes，如下：

```
resource <resource>
  net {
    protocol C;
    allow-two-primaries yes;
  }
  ...
}
```

配置之后不要忘记在两节点间同步配置。在两节点上运行 drbdadm adjust <resource>。

之后就可以在两个节点上同时运行 drbdadm primary <resource>。

6.5.2 临时双主模式

如若在一个单主模式的配置下临时调整启用双主模式，可以使用以下命令：

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

要结束临时双主模式，同样可以运行上面类似的命令，不过参数为 --allow-two-primaries=no（如果需要可以跟上所需要的复制协议）

6.5.3 开机自动启用

当配置资源为双主模式时，也可以配置为在系统开机时自动启动 drbd 并切换为主模式。

```
resource <resource>
  startup {
    become-primary-on both;
  }
  ...
}
```

可利用系统脚本/etc/init.d/drbd 作为系统启动和推广相关资源。

备忘

在 Pacemaker 管理的 drbd 配置中，成为主的方法不要求和不推荐。Pacemaker 配置中，资源的升降级应该有集群进行管理。

6.6 在线设备验证

6.6.1 启用在线设备验证

默认情况下在线设备验证是未启用的，如果要启用它，可以在配置文件/etc/drbd.conf 资源配置部分添加：

```
resource <resource>
  net {
```

```
    verify-alg <algorithm>;
  }
  ...
}
```

<algorithm>算法，系统内核配置系统密码 api 支持任意的信息加密算法。通常情况下，你至少可以从 sha1、md5 和 crc32c 中选择其一。

如果对现有资源进行改更改，同样需要同步 drbd.conf 到对等节点，需要在两个节点上运行 drbda adjust <resource>。

6.6.2 调用在线验证

在启用在线验证后，可以通过如下命令启动验证：

```
drbdadm verify <resource>
```

这样做之后 drbd 就开始对<resource>进行在线设备验证，如果探测到有块未同步，将会对这些块进行标记，并向内核日志中记录一条信息。此时任何应用程序都可正常操作，包括切换资源的角色。

如果在验证运行时 out-of-sync 块被发现，可能需要在验证完毕之后使用如下命令通信同步：

```
drbdadm disconnect <resource>
drbdadm connect <resource>:
```

6.6.3 自动在线验证

大多数用户为轻松完成工作，都希望自动进行在线设备验证。可以创建一个名为 /etc/dron.d/drbd-verify 的文件，输入如下内容：

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

这样在每星期日的午夜的 00:42 都会进行设备的在线验证。

如果要对所有的资源启用在线设备验证（如通过在/etc/drbd.conf 文件的 common 部分添加 verify-alg <algorithm>），也可以使用如下进行调用：

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

6.7 配置同步率

正常情况下，后台同步需要保证（这会导致同步目标出现暂时的不一致）尽可能快的完成，但是后台同步带宽不能占用所有的带宽，不然会影响前端复制，影响应用程序的性能。因此，也就必须根据各自的硬件环境，配置同步的带宽。可以是永久的也可能是临时的。

重要

设置的同辈速率比次节点的最大吞吐量大还大的话，会变得无意义。次节点的写速度必会比它的 I/O 子系统还快，因此它是一个正在进行同步的同步目标设备。
基于同样的理由，同步速率设置的超过网络的最大可用带宽也是没有任何意义的。

6.7.1 固定的同步速率配置

一个资源用于后台同步的最大带宽由资源的速率选项控制。它必须包含在资源配置文件 /etc/drbd.conf 的磁盘部分：

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

请注意，同步的速率是 bytes 字节，而不是 bits/s。

提示

根据经验同步速率比较合理的是可用带宽的30%。这样加入一个I/O子系统能支持180MB/s的吞吐量，而千兆网络可支持110MB/s（网络带宽会成为瓶颈），可以计算出：

图 6.1。以网络带宽 110MB/ s 为例，同步速率：

这样就得出同步速率的推荐值为 33M。

相反，如果 I/O 子系统的支持 80MB/s 的吞吐量，千兆以网络环境（I/O 子系统是瓶颈）的最大吞吐量应该为：

图 6.2。以网络带宽 80MB/ s 为例，同步速率：

这样就得出同步速率的推荐值为 24M。

6.7.2 临时的同步速率配置

有时候需要暂时调整同步的速率。例如，集群节点进行预定的维护后希望加快后台重新同步，或者是当应用程序非常频繁的进行写的时候，希望降低重新同步的速率以保证比较大的带宽用于复制。

例如，千兆以太网将大部分带宽用于重新同步，可以使用如下命令：

drbdadm disk-options --resync-rate=110M <resource>

只需要在一个节点上发出以上命令。

要恢复这个临时设置，并启用/etc/drbd.conf 中配置的同步速率，可以使用如下命令：

drbdadm adjust <resource>

6.7.3 可变同步率配置

当多个 DRBD 资源共用同一个复制/同步的网络时，固定的同步速率可能就不是很合适。在这种情况下，可以配置可变的同步速率。DRBD 使用一个自动控制回路的算法来确定和调

整同步速率。该算法可以保证始终有足够的带宽用于前台的复制，从而大大的减小了后台同步 I/O 对前台的影响。

可变同步速率的优化配置很大程度上取决于可用的网络带宽，应用 I/O 模型和链路的拥塞状况。理想的设置还取决于是不是应用了 DRBD 代理。在优化 DRBD 的配置方面比较明智的是请教专业的 DRBD 人员。下面提供一个配置的样例（假定结合了 drbd 代理）：

```
resource <resource> {
  net {
    c-plan-ahead 200;
    c-max-rate 10M;
    c-fill-target 15M;
  }
}
```

提示

c-fill-target 比较好的起始值为 BDP*3，其中 BDP 为复制链接产生的带宽延迟

6.8 配置基于校验的同步

默认情况下基于校验的同步是未启用的。如果要启用的话可以在配置文件/etc/drbd.conf 中配置：

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

}

<algorithm>算法，系统内核配置系统密码 api 支持任意的信息加密算法。通常情况下，你至少可以从 sha1、md5 和 crc32c 中选择其一。

如果对现有资源进行改更改，同样需要同步 drbd.conf 到对等节点，需要在两个节点上运行 drbda adjust <resource>。

6.9 配置拥堵策略和暂停复制

在复制带宽高度可变的环境下（广域网下的典型），有可能复制就会发生拥堵。使用默认的配置是不可取的，因为这样会导致主节点上 I/O 阻塞。

相反，如果在这种情况下暂停 DRBD 的复制，会导致主节点上的数据在此节点上设置为 *pull ahead*。在这种模式下，drbd 是始终开启着复制通道的-它并没有置换到 disconnected 断开模式，不过在没有足够的带宽可用前，它并不进行复制操作。

下面的一个 drbd proxy 的配置样例：

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

通常同时设置 congestion-fill、congestion-extents 和 pull-ahead 选项一起工作比较好。

congestion-fill 比较合理的值是 90%。

当通过 drbd proxy 复制时，drbd proxy 缓冲内存。

当不设置 drbd proxy 时，通过 tcp 网络发送缓冲。

Congestion-extents 比较合理的值为设置的 al-extents 值得 90%。

6.10 配置 I/O 错误处理策略

DRBD 可通过在资源的磁盘配置文件/etc/drbd.conf 中配置 on-io-error 选项处理底层的 I/O 错误：

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

如果想对所有的资源做一个全局性的 I/O 错误处理策略，也可以在 common 部分设置该选项。

<strategy>策略可能有如下选项之一：

1、 detach 分离：这是默认和推荐的选项。如果在节点上发生底层的磁盘 I/O 错误，它会将设备运行在 diskless 无盘模式下。

2、 pass_on：drbd 会将 I/O 错误报告到上层。在主节点上，它会将其报告给挂载的文件系统，但是在此节点上就往往忽略（因此此节点上没有可以报告的上层）

3、 -local-io-error 调用本地磁盘 I/O 处理程序中定义的命令。这就需要有相应有让 local-io-error 调用的资源处理程序处理错误的命令。这就给管理员留有足够自由的权力使用命令或者是脚本调用 local-io-error 处理 I/O 错误。

备忘

在早起的 DRBD 版本中（8.0 之前）其实包含另外一个选项，panic，它是在本地出现 I/O 错误时强行将发生 I/O 错误的节点从集群中移除。虽然该选项不再可用，但是通过 local-io-error/+ call-local-io-error+接口可实现与之相同的功能。当你对足够了解这种行为的影响时你可以应用这样的该功能。
你可以按照如下的操作重新配置正在运行资源的 I/O 错误处理策略：

编辑资源配置文件/etc/drbd.d/<resource>.res

将配置文件拷贝到对等节点

在两个节点上运行 drbdadm adjust <resource>

6.11 配置复制传输完整性验证

默认情况下资源的复制传输完整性验证是不开启的。可以在/etc/drbd.conf 文件中添加如下内容启用复制传输完整性验证：

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

<algorithm>算法，系统内核配置系统密码 api 支持任意的信息加密算法。通常情况下，你至少可以从 sha1、md5 和 crc32c 中选择其一。

如果对现有资源进行改更改，同样需要同步 drbd.conf 到对等节点，需要在两个节点上运行 drbda adjust <resource>。

6.12 重新调整资源

6.12.1 在线增长

如果后端块设备支持（在线）操作，那也就可能支持在调整时在线调整 DRBD 设备的大小。如要这样做，必须满足两个条件：

1. 受影响的后端设备必须是一个逻辑卷管理子系统，如 LVM。

2. 资源目前必须出在连接状态。

如果后端块设备在两个节点上都支持增长，就需要保证仅有一个节点处于主状态。然后在一个节点上输入：

drbdadm resize <resource>

新的部分将触发同步。同步从主节点到次节点上。

6.12.2 离线增长

当两个节点的后端块设备增长时，DRBD 是不活动，且 drbd 利用外部元数据，那么新的大小会自动识别。任何管理的干预都是不必要的。Drbd 设备的新大小激活，并建立网络连接。

如果 drbd 资源配置的是使用内部元数据，那么这个元数据必须在增长的设备变成可用前移动进去。可以采用以下步骤完成操作：

警告 **取消配置的 DRBD 资源**

这是一个超级进程，你可以自己觉得如何使用。 drbdadm down <resource>

保存萎缩之前的元数据到文本

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

必须在两个节点上都这样操作，为每个节点都单独的转储文件。千万不要在一个节点导出数据然后拷贝到其他的节点上，这样会导致 drbd 不能工作。

两个节点上的后端块设备都支持增长

可在两个节点上，根据文件/tmp/metadata 调整大小信息（la-size-sect）。请注意参数 la-size-sect 必须在 sectors 中指定。

重新初始化元数据区域：

drbdadm create-md <resource>

在两个节点上重新导入正确的元数据：

drbdmeta_cmd=$(drbdadm -d dump-md test-disk)

${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

备忘

这个例子使用 shell 变量代替。它可能或不能在其他的 shell 中工作。检查环境变量看是不是没有正确使用。

重新启用 drbd 资源：

drbdadm up <resource>

在一个节点上提升 drbd 的资源：

drbdadm primary <resource>

最后，增长的文件系统填补扩展的 drbd 的设备的大小。

6.12.3 在线收缩

在收缩 drbd 设备前，必须先缩小 drbd 的上层，例如，通常的文件系统系统。由于 drbd 不能获取到文件系统实际使用多少空间，为了不丢失数据操作一定要小心。

备忘

能不能在线收缩取决于正在使用的文件系统。大多数的文件系统是不支持在线收缩的。如 xfs 根本不支持收缩。
当使用内部元数据时一定要考虑元数据使用的磁盘空间。提供给 drbdadm resize 的大小是文件系统的大小。在使用内部元数据的情况下，drbd 所需求的大小要高（详见 17.1.3 计算元数据大小 一节）。

要在线收缩 drbd，可以在收缩文件系统之后使用如下命令进行操作：

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

通常可以使用乘数后缀为<new-size>(K/M/G 等)，在收缩 drbd 之后，还可以收缩包含的块设备（如果支持收缩的话）。

6.12.4 离线收缩

如果收缩不活动的 drbd 块设备，在尝试连接时，drbd 就会拒绝连接到块设备，因为目前它太小(使用外部元数据的情况下),或者它不能找到内部元数据(使用内部元数据的情况下)。为解决该问题，使用该程序（如果不能使用在线收缩）：

警告 **要在一个节点上收缩文件系统，依然要配置 drbd**

这是一个超级进程，你可以自己觉得如何使用。

取消配置的 drbd 资源：

drbdadm down <resource>

保存萎缩之前的元数据到文本

drbdadm dump-md <resource> >+ /tmp/metadata

必须在两个节点上都这样操作，为每个节点都单独的转储文件。千万不要在一个节点导出数据然后拷贝到其他的节点上，这样会导致 drbd 不能工作。

在两个节点上收缩后端块设备

可在两个节点上，根据文件/tmp/metadata 调整大小信息（la-size-sect）。请注意参数 la-size-sect 必须在 sectors 中指定。

只有当你使用内部元数据（此时可能由于收缩进行丢失），重新初始化元数据区域：

drbdadm create-md <resource>

在两个节点上重新导入正确的元数据：

drbdmeta_cmd=$(drbdadm -d dump-md test-disk)

${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

备忘

这个例子使用 shell 变量代替。它可能或不能在其他的 shell 中工作。检查环境变量看是不是没有正确使用。

重新启用 drbd 资源：

drbdadm up <resource>

6.13 禁用后端设备刷新

注意

当有耐性的 drbd 使用 battery-backed 高速写缓存（BBWC）时，应该只禁用设备的刷新。当 battery 池耗尽时大多数的存储控制器允许自动禁用写缓存，当 battery 死时切换到写模式。强烈建议使用该功能。

当运行无 BBWC 或者是 BBWC 池耗尽时，禁用 DRBD 缓存很可能会导致数据丢失，因此不应该尝试这样做。

DRBD 允许分别启用和禁用后端设备刷新对复制数据集和 DRBD 拥有自身的元数据。这写选项默认都是开启的。如果要禁用一个（或者是两个），就需要设置 drbd 配置文件中的 /etc/drbd.conf 中的 disk 部分。

要禁用复制数据集的磁盘刷新，就需要配置如下：

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

要禁用 drbd 元数据的磁盘刷新，需要配置如下：

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

在核对资源配置后（同样需要同步/etc/drbd.conf 到两个节点间），可以在两个节点运行如下命令：

```
drbdadm adjust <resource>
```

6.14 配置裂脑行为

6.14.1 裂脑通知

如果配置 DRBD 会调用裂脑处理程序，裂脑发生时就会被探测到。要配置这个程序，需要对资源进行如下配置：

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

<handler>可能是目前系统中一个可执行的文件。

Drbd 包含一个裂脑处理程序脚本/usr/lob/drbd/notify-split-brain.sh。它这是通过电子邮件的方式发送到指定的地址。要配合程序发送信息到 root@localhost（这假设是设置的系统管理员的邮件地址），配置如下：

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

当配置已经在资源上进行修改（同步到两个节点上），就不需要添加其他的处理就可以启动处理程序。Drbd 会在下一次出现裂脑时候就简单的调用该处理程序。

6.14.2 裂脑自动修复策略

为了能够启用和配置 drbd 的自动裂脑恢复策略，就必须理解 drbd 提供的几种配置选项。Drbd 提供基于同时几个几点处于主角色探测的裂脑恢复程序。为此，drbd 资源的 net 配置部分有以下关键字：

after-sb-0pri：裂脑已经被探测到，但是现在没有节点处于主角色，对于这个选项，drbd 有以下关键字：

disconnect:不需要自动恢复，仅仅是调用裂脑处理程序的脚本（如果配置了），断开连接并出在断开模式。

discard-younger-primary:放弃和回滚最后成为主的上面所做的修改。

discard-least-changes:放弃和回滚，变动比较少的主机上的修改。

discard-zero-changes:如果任何节点都没有发生任何变化，仅仅申请在一个节点上做出继续修改即可。

after-sb-1pri：裂脑已经被探测到，现有有一个节点处于主角色，对于这个选项，drbd 有以下关键字：

disconnect:和 after-sb-0pri 一样，调用裂脑处理程序的脚本（如果配置了），断开连接并出在断开模式。

consensus:和 after-sb-0pri 中同样的修复策略。如果利用这些策略裂脑危害能选择，那就能自动解决。否则，同样断开指定的动作。

call-pri-lost-after-sb:和 after-sb-0pri 中同样的修复策略。如果利用这些策略裂脑危害能选择，就在受危害的节点上调用 pri-lost-after-sb 程序。这个程序必须确认在 handlers 中配置，并考虑到从集群中移除该节点。

discard-secondary:不管哪个主机只要处于次角色，都是裂脑的危害者。

after-sb-2pri：在两个节点都处于主角色时，裂脑被发现。次选项使用和 after-sb-1pri 同样的关键字，丢弃次节点并达成共识。

备忘

DRBD 的这三个选项，往往被忽略，因为很少用到。可以参见 drbd.cong(5)查看裂脑恢复的详细关键字，这里不再细说。
例如，在双主模式下，使用 GFS 或者 OCFS2 文件系统作为资源的块设备，可以使用如下方式定义恢复策略：

```
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;
    ...
  }
  ...
}
```

6.15 创建三节点设置

三节点的设置涉及 DRBD 设备之上另一个堆叠。

6.15.1 设备堆叠的注意事项

这种类型的设置需要注意一下事项：

堆叠的设备是活动的设备。假若已经配置了 **drbd** 设备**/dev/drbd0**，并且在堆叠设备上配置的为**/dev/drbd10**，那么**/dev/drbd10** 这就是挂载和使用的设备。

设备的元数据将被保持两次，分别为底层的 **drbd** 设备和堆叠的设备上。在堆叠的设备上，必须使用内部元数据，这就意味着堆叠的设备比不堆叠的设备有效可用的存储区域要小。

如要上层的堆叠设备运行，那么底层的设备必须是主。

为了保证能够同步到备份节点，堆叠设备必须是活动并且是主角色。

6.15.2 配置堆叠的资源

在下面的例子中，节点分别命名为 alice、bob 和 charlie，而 alice 和 bob 形成一个两节点的集群，而 charlie 则是备份的节点。

```
resource r0 {
  net {
    protocol C;
  }

  on alice {
    device      /dev/drbd0;
    disk        /dev/sda6;
    address     10.0.0.1:7788;
    meta-disk internal;
  }

  on bob {
```

```
    device     /dev/drbd0;
    disk       /dev/sda6;
    address    10.0.0.2:7788;
    meta-disk internal;
  }
}

resource r0-U {
  net {
    protocol A;
  }

  stacked-on-top-of r0 {
    device      /dev/drbd10;
    address     192.168.42.1:7788;
  }

  on charlie {
    device      /dev/drbd10;
    disk        /dev/hda6;
    address     192.168.42.2:7788; # Public IP of the backup node
```

```
    meta-disk   internal;
  }
}
```

在有三个节点的情况下，配置文件 drbd.conf 必须在每一个集群的节点中都有。注意以下关键字在不堆叠的资源配置中是没有的：

stacked-on-top-of：该选项表示 drbd 所使用的资源为一个堆叠的资源。它替换普通的资源配置的部分。在底层的资源中不能使用 stacked-on-top-of 选项。

备忘

DRBD 的这三个选项，往往被忽略，因为很少用到。可以参见 drbd.cong(5)查看裂脑恢复的详细关键字，这里不再细说。
对堆叠的资源来说 Protocol A 不是必须的。你可以根据需求，选择任何 drbd 的复制协议。

6.15.3 启用堆叠资源

为了启用堆叠资源，首先必须先启用底层的资源并推广：

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

而对不堆叠的资源，则在必须堆叠的资源上创建 drbd 元数据。可是使用如下命令：

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

然后你就可以启用堆叠的资源了：

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

在此之后，你可以在备用节点上启动资源，开启三节点复制：

```
drbdadm create-md r0-U
```

drbdadm up r0-U

为了自动化管理堆叠的资源，必须在集群管理配置中整合堆叠的资源。参见第 8.4 节“Pacemaker 集群中使用堆叠的 DRBD 资源”了解在 Pacemaker 集群中管理集群的一些详细信息。

6.16 使用 DRBD proxy

6.16.1DRBD proxy 部署注意事项

DRBD proxy 进程即可直接放在 DRBD 自身的服务器上，也可以放在其他的单独的专用服务器上。DRBD proxy 实例可作为多个 drbd 设备分布在多个节点上的 proxy 代理。

Drbd proxy 对 drbd 来说是透明的。通常情况下会有大量的数据报进行传输，这样就有相当大的活动日志。这就可能导致时间比较长的同步在主节点崩溃后重新运行，因此建议启用 drbd 的 csums-alg 设置。

6.16.2 安装

请联系 linbit 销售商获取 drbd proxy。除非有特殊需求，要不就请使用最新的 drbd proxy。

在 drbian 和 debian-based 系统上安装 drbd proxy，使用 dpkg 工具操作如下(使用的 drbd proxy 的版本要和目标架构的版本相匹配)：

dpkg -i drbd-proxy_1.0.16_i386.deb

在基于 rpm 的系统（如 sles 和 rhel）上安装 drbd proxy，使用 rpm 工程操作如下(使用的 drbd proxy 的版本要和目标架构的版本相匹配)：

rpm -i drbd-proxy-1.0.16-1.i386.rpm

要配置 drbd proxy 还需要安装 drbd 管理工具 drbdadm。

以二进制文件安装 drbd proxy 通常会安装到/etc/init,d 的 init 脚本中。请使用 init 脚本 start/stop drbd proxy 服务，它相应需要使用 drbdadm 工具配置 drbd proxy。

6.16.3License 文件

当从 linbit 获取一个 license 时，drbd proxy 的运行也是需要 license 文件的。这个文件名为 drbd-proxy.license，必须在目标机器上拷贝到/etc 目录下：

cp drbd-proxy.license /etc

6.16.4 配置

Drbd proxy 在 drbd 的主被配置文件中配置。它是在主机部分中的 proxy 和 additional proxy 的 additional 选项部分进行配置。

下面为在 drbd 节点上直接配置运行 drbd proxy 的例子：

```
resource r0 {
        net {
          protocol A;
        }
        device      minor 0;
        disk        /dev/sdb1;
        meta-disk   /dev/sdb2;


        proxy {
             compression on;
             memlimit 100M;
        }


        on alice {
```

```
            address 127.0.0.1:7789;
            proxy on alice {
                inside 127.0.0.1:7788;
                outside 192.168.23.1:7788;
            }
        }

        on bob {
            address 127.0.0.1:7789;
            proxy on bob {
                inside 127.0.0.1:7788;
                outside 192.168.23.2:7788;
            }
        }
}
```

Insert ip 地址为 drbd 和 drbd proxy 之间沟通使用，而 outside ip 地址为 drbd proxy 之间沟通使用。

6.16.5 控制 DRBD proxy

Drbdadm 提供 proxy-up 和 proxy-down 子命令配置或删除连接到本地 drbd 资源的 drbd proxy。可使用/etc/init.d/drbdproxy 命令进行启动和停止的操作。

Drbd proxy 有一个焦作 drbd-proxy-ctl 的底层的配置工具。当没有其他的选项时它可在交互模式下调用。帮助信息命令显示如下：

add connection <name> <ip-listen1>:<port> <ip-connect1>:<port>

<ip-listen2>:<port> <ip-connect2>:<port>

Creates a communication path between two DRBD instances.

set memlimit <name> <memlimit-in-bytes>

Sets memlimit for connection <name>

del connection <name>

Deletes communication path named name.

show

Shows currently configured communication paths.

show memusage

Shows memory usage of each connection.

list [h]subconnections

Shows currently established individual connections

together with some stats. With h outputs bytes in human

readable format.

list [h]connections

Shows currently configured connections and their states

With h outputs bytes in human readable format.

list details

Shows currently established individual connections with

counters for each DRBD packet type.

quit

Exits the client program (closes control connection).

shutdown

Shuts down the drbd-proxy program. Attention: this

unconditionally terminates any DRBD connections running.

6.16.6 故障排除

Drbd proxy 日志通过 syslog 使用 LOG_DAEMON 功能。通常情况下，可在 /var/log/daemon.log 查看 drbd proxy 信息。

例如，如果 drbd 连接失败，因为本节点不能连接到领导的一端，就会记录一些类似拒绝连接的信息。在这种情况下酒需要检查 DRBD 是不是云翔在两个节点上（不是 standalone 模式），并且 drbd proxy 在运行。同时也需要自己检查配置。

7 故障排除和错误

目录

本章介绍在硬件或系统故障的情况下执行的任务

7.1 磁盘故障的处理

如何处理磁盘故障取决于 drbd 配置的磁盘 I/O 错误处理策略（参阅 2.11 节“磁盘错误处理策略”），和元数据类型的配置（参阅 17.1 节“DRBD 元数据”）。

备注

在大多数情况下，这里所描述的步骤仅仅使用 DRBD 直接运行在磁盘上的时候。一般情况下，对 DRBD 运行在顶层时不适用。

MD 软 raid 设置（在这种情况下使用 mdadm 管理磁盘的更换），

设备映射 raid（使用 dmraid）

硬 raid 设备（可根据供应商的指示处理磁盘故障）

一些非标准设备映射的虚拟块设备（参阅设备映射的文档）

7.1.1 从磁盘中手工分离出 DRBD

如果 drbd 配置通过 I/O 错误（不推荐），就必须先分离 drbd 的资源，即，分离出后端存储：

drbdadm detach <resource>

可以运行 drbdadm dstate 命令，查看资源是不是运行在 diskless 无盘模式：

drbdadm dstate <resource>

Diskless/UpToDate

如果主节点发生磁盘故障，可以结果这一步完成切换操作。

7.1.2 自动分离 I / O 错误

如果 drbd 配置为自动分离 I/O 错误（推荐选项），DRBD 就可以从后端存储上自动分离资源，而无需人工干预。你可以通过 drbdadm dstate 命令核对资源是不是运行在 diskless 无盘模式下。

7.1.3 使用内部元数据时更换故障磁盘

如果使用内部元数据，就足以将 drbd 设备绑定到新的磁盘上。如果新的磁盘名被其他的 linxu 设备或者是故障磁盘占用，就需要修改 drbd 的配置文件进行修改。

这个过程包括新建元数据集，然后从新连接资源：

drbdadm create-md <resource>

v08 Magic number not found

Writing meta data...

initialising activity log

NOT initializing bitmap

New drbd meta data block sucessfully created.

drbdadm attach <resource>

新磁盘的完全同步是瞬间和自动进行的。可以通过查看/proc/drbd 检测后台同步的进度信息。

7.1.4 使用外部元数据时更换故障磁盘

使用外部元数据时，程序基本上是一样的。然而，DRBD 不能识别单独的磁盘更换，因此需要一个额外的操作步骤：

drbdadm create-md <resource>

v08 Magic number not found

Writing meta data...

initialising activity log

NOT initializing bitmap

New drbd meta data block sucessfully created.

drbdadm attach <resource>

drbdadm invalidate <resource>

这里 drbdadm 无效命令触发同步。同样可以通过查看/proc/drbd 观察同步的进展情况。

7.2 处理节点故障

当 drbd 探测到对等节点宕机（无论是真正的硬件故障还是人工干预），drbd 都将其连接状态有 Connected 变为 WFConnection，直到对等节点恢复。Drbd 资源会在此时运行 disconnected 模式下。在 disconnected 模式下，资源以及和棋相关的块设备都是可用的，

并可以在必要时进行升降级，但是并不能将修改复制到对等节点。相反，drbd 在块设备上存储的内部信息在 disconnected 时仍会被修改。

7.2.1 处理次节点临时故障

如果一个处于次角色的节点资源暂时出现故障（例如因为内存问题导致的需要更换内存），进一步的干预是不需要的—除了明显的必须修复的故障节点并恢复回来。当这种情况发生后，两个节点会重新建立连接并启用，drbd 会复制在此期间所有进行的修改到次节点上。

重要

由于 drbd 资源重新同步算法的影响，此时此节点上的资源会出现短暂的不一致 inconsistent。在这个短暂的时间内，如果对等节点出现不可用，此时次节点是不能升级到主节点的。因此，在集群不冗余的期间，这个时间为次节点停机的时间加上重新同步所需要的时间。

7.2.2 处理主节点临时故障

从 drbd 的角度上看，主节点的故障和次节点的故障基本类似。存活的节点发现对等阶段故障，就会将其置为 disconnected 模式。Drbd 并不升级存活的节点到主，需要集群管理程序重要做。

当失败的节点被修复并回到集群中，在此节点也是如此，因此，无需进行人工干预。再次，drbd 并没有改回资源的角色，它是由几圈管理器控制的（如果配置了的话）。

Drbd 通过一个特殊的机制保证在主节点出现故障的情况下块设备的一致。详细可以参阅 17.3 节“活动日志”。

7.2.3 节点永久故障的处理

如果一个节点遇到不可恢复的问题或者是永久性破坏，需按照以下步骤进行操作：

将出现故障的硬件替换为与之类似性能和容量的磁盘。

备注

替换也可以替换性能差点，但是不推荐。替换为磁盘容量比较小的是不可能的，这样会导致 drbd 拒绝连接被替换的节点。

安装基本系统和应用程序。

安装 drbd 并从幸存的节点上拷贝/etc/drbd.conf 和所有的/etc/drbd.d

按照第五章的步骤配置 drbd，但是不进行 5.5 节的“初始化设备同步”

此时没有必要手工进行一个完整的设备同步，它会在开始时自动连接到存活的主节点上。

7.3 手工进行裂脑修复

DRBD 检测到裂脑恢复连接并变成可用，和对等节点达成初步的握手。如果 drbd 检测到两个节点（也可能是两个节点断开时）都是主角色，它就连接关闭复制的连接。这个可在系统日志中发现如下信息：

Split-Brain detected, dropping connection!

当裂脑被探测到时，一个节点将保持始终以 StandAlone 状态连接资源，另外一个节点也可能处于 StandAlone 状态（如果两个节点被探测到同时处于裂脑状态），也可能是 WFConnection 状态（如果对等接在还没有来得及探测到裂脑就 down 掉的话）。

此时，必须手工干预选择丢失一个节点的修改被丢失（这个节点被称为裂脑受害者），除非配置 drbd 的裂脑自动修复。这种干预需运行如下命令：

drbdadm secondary <resource>

drbdadm connect --discard-my-data <resource>

在其他节点上（裂脑幸存者），如果它的状态也为 StandAlone 状态，可输入以下命令：

drbdadm connect <resource>

如果节点已经处于 WFConnection 状态，可是省略这一步，因为它会自动进行重新连接。

如果裂脑影响的是堆叠的资源，需要使用 drbdadm –stacked 来代替 drbdadm。

当连接时裂脑受害者将立即改变连接状态为 SyncTarget，并被主节点覆盖其余节点的修改。

备注

裂脑受害者不是一个完整的设备同步，相反，还有可能执行本地修改的回滚操作，而将裂脑幸存者的所做的修改完整的传输到裂脑受害者。
当重新完成同步后，就认为裂脑问题已经解决，两个节点的数据再次达成一致，形成一个冗余复制的存储系统。

第四篇 应用 DRBD 功能

8 DRBD 集成 Pacemaker 集群

目录

在实际应用中 drbd 结合 Pacemaker 集群一起应用是比较常见的。Pacemaker 也是 DRBD 众多应用中比较强大的一个应用。

8.1 Pacemaker 引子

在 linux 平台上 Pacemaker 是一个复杂的、功能丰富并且广泛应用和部署的集群。它配备有丰富的文档集。为了了解这一章，强烈建议阅读以下文件：

“从头开始，一步一步教你配置高可用性集群” –“Clusters From Scratch, a step-by-step guide to configuring high availability clusters;”

CRM CLI（命令行界面）工具，CRM shell 手册，Pacemaker 的简单直观的命令行界面—“CRM CLI (command line interface) tool, a manual for the CRM shell, a simple and intuitive command line interface bundled with Pacemaker;”

Pacemaker 配置说明。解释 Pacemaker 的设计概念和原理的参考文档。

8.2 集群配置添加 DRBD-backed 服务

这部分主要介绍如何启用 Pacemaker 集群的 DRBD-backed 服务。

备注

如果采用了 drbd ocf 资源代理，建议推迟 drbd 的启用、关闭、推广和升降级专门的 OCF 资源代理。也就是说需要禁用 drbd 的 init 脚本（自启动脚本）：
chkconfig drbd off

ocf：linbit：drbd OCF 代理提供主/从功能，允许 Pacemaker 启动和监控、推广和升降级多个节点的 drbd 资源。同时也必须了解 drbd 的 RA 断开和分离所有的在 Pacemaker 管理 drbd 资源的关闭，也可为另外一个节点启用待机模式。

重要

OCF 资源和 drbd 都属于 linbit 提供商，因此安装为/usr/lib/ocf/resource.d/linbit/drbd。OCF 资源代理作为传统的资源代理，提供心跳检测，并安装到 /usr/lib/ocf/resource.d/heartbeat/drbd。传统的 OCF RA 已经被废弃并不再使用。

当 Pacemaker CRM 集群中使用 OCF 资源代理时为了启用 mysql 数据库的 drbd-backed 配置，为保证服务预先启动提升 drbd 的资源，必须同时创建必要的资源和 Pacemaker 限制。可如下例中使用 crm shell：

Drbd-backed 的 mysql 服务的 Pacemaker 配置

```
crm configure

crm(live)configure# primitive drbd_mysql ocf:linbit:drbd \

        params drbd_resource="mysql" \

        op monitor interval="15s"

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" \

          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，配置应该被启用。Pacemaker 现在选择一个节点升级 drbd 的资源，然后再找个节点上启用 drbd-backed 资源。

8.3 在 Pacemaker 集群中使用 resource-level fencing

本节概述为防止 Pacemaker 从促进 drbd 的主/备资源时，drbd 复制链接被中断的必要采取的步骤。这使 Pacemaker 开启服务时使用过时的数据，从而导致不必要的“time warp”。

为了保证 drbd 的 resource-level fencing，必须添加以下几行到资源的配置中：

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

由于集群基础设施的使用，也需要对 handlers 部分进行修改：

基于心跳的 Pacemaker 集群可参看“8.3.1 Resource-level fencing 和 dopd”的配置概述。

Corosync 和基于心跳的集群都可使用该功能，参加“8.3.2 Resource-level fencing 使用集群信息库（CIB）”

重要

绝对至关重要的是至少配置两个独立的集群通信通道此功能正常工作。基于心跳 Pacemaker 集群应在 ha.cf 配置文件中定义至少两个集群通信链路。Corosync 集群中至少在 corosync.conf 有两个冗余环。

8.3.1 Resource-level fencing 和 dopd

在基于心跳的 Pacemaker 集群中，drbd 可使用一个叫做 outdate-peer 服务的 resources-level fencing 功能，简称 dopd。

8.3.1.1 dopd 的 Heartbeat 配置

为启用 dopd，必须在文件/etc/ha.d/ha.cf file 中添加如下行：

```
respawn hacluster /usr/lib/heartbeat/dopd

apiauth dopd gid=haclient uid=hacluster
```

可根据首选的分布调整 dopd 的路径。在某些分布和结构中，正确的路径为 /usr/lib64/heartbeat/dopd。

在进行修改并拷贝 ha.cf 文件到对等节点后，将 Pacemaker 置为维护模式，并运行 /etc/init.d/heartbeat 重新加载配置文件。之后，需要确认 dopd 为一个正在运行的进程。

备注

可使用 ps ax | grep dopd 或者是发出 killall -0 dopd.检查这个进行。

8.3.1.2 Dopd 的 drbd 配置

一旦 dopd 运行，请为 drbd 的资源添加如下配置：

```
resource <resource> {

    handlers {

        fence-peer "/usr/lib/heartbeat/drbd-peer-outdater -t 5";

        ...

    }
```

```
    disk {
        fencing resource-only;
        ...
    }
    ...
}
```

对于 dopd 而言分布可能将 drbd-peer-outdater binary 根据系统架构的情况放置到 /usr/lib64/heartbeat。

最后，复制 drbd.conf 到对等节点，并使用 drbdadm 命令重新调整资源并反映出变化调整。

8.3.1.3 测试 dopd 功能

为了测试 dopd 是不是运行正常，在心跳服务正常运行时中断配置的复制连接和连接的资源。也可以通过简单的拔出物理链路，当然这是相当有侵略性的。当然也可以通过插入一条临时的 iptables 规则，丢弃 drbd 的传输到资源使用的 tcp 端口的信息。

再次之后，可以观察到资源的连接状态从 connection 转变为 WFConnection。几秒之后，就可以发现磁盘的状态变成了 Outdated/DUnknown。而这正是 dopd 所负责的。

再次之后的任何企图切换过时资源到主角色的操作都会失败。

当重新建立网络连接（不管是插入物理链路还是将 iptables 的临时规则移除），连接状态都会改变为 connection，并自动同步到目标（如果主节点在网络中断期间发生了变化）。然后就观察到一个简短的同步期间，最后，以前过时的资源则再次被标记为 UpToDate。

8.3.2 Resource-level fencing 使用集群信息库（CIB）

需在 drbd.conf 中设置两个选项才能启用 Pacemaker 的 resource-level fencing：

```
resource <resource> {
  disk {
```

```
    fencing resource-only;
    ...
  }
  handlers {
    fence-peer "/usr/lib/drbd/crm-fence-peer.sh";
    after-resync-target "/usr/lib/drbd/crm-unfence-peer.sh";
    ...
  }
  ...
}
```

因此，如果 drbd 复制连接变成 disconnected，脚本 crm-fence-peer.sh 会联系几圈管理器，探测 Pacemaker 的 Master/Slave 资源相关的 drbd 资源，并确保 Master/Slave 资源不再在任一节点上被提升为当前活跃的。相反，当连接重新建立 drbd 完成同步后，这个约束就会被删除，并且集群管理器也可自由的在任何节点上升级资源。

8.4 Pacemaker 集群只用堆叠的 drbd 资源

Stacked 堆叠资源允许 drbd 被用在多点集群的多级冗余，或建立异地灾备的功能。本节将介绍如何在泽中配置中配置 drbd 和 Pacemaker。

8.4.1 添加站点灾难恢复和 Pacemaker 集群

在这种配置的情况下，会在一个站点两个节点的高可用性集群，外加一个被安置异地处理单独的节点。第三个节点作为一个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一概念。

图 8.1。在 Pacemaker 集群中的 DRBD 资源堆积

在这个例子中，alice 和 bob 为两个 Pacemaker 集群节点，而 charlie 则为一个不被 Pacemaker 管理的场外节点。

要建立这样的一个配置，首先需要配置和初始化 6.15 节“创建三节点的设置”中所述的 drbd 的资源。之后，根据 CRM 的配置配置 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
```

如果创建一个名为/tmp/crm.txt 的临时文件，可以使用如下命令导入活动集群的配置：

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

该配置将确保在 alice/ bob 集群的以下动作按照正确的顺序发生：

1、 Pacemaker 在两个集群节点上启动 drbd 资源 r0，并升级其中一个节点为主角色。

2、 Pacemaker 之后启动 ip 地址 192.168.42.1，用于复制堆叠的资源到第三节点上。先这样做的节点的 drbd 资源 r0 晋升为主节点。

3、 在 r0 的主节点上复制 r0-U 的 ip 地址，现在 Pacemaker 启动 r0-U 的用于连接和复制到异地节点的 drbd 资源。

4、 然后 Pacemaker 提升 r0-U 的资源为主角色，它可被一个应用程序使用。

因此，Pacemaker 的配置确保群集节点之间不仅有完整的数据冗余，而且第三，异地节点也有。

备注

8.4.2 使用堆叠的资源，实现 4 路冗余的 Pacemaker 集群

这个类型的设置通常和 drbd proxy 部署在一起。

在此配置中共有三个 drbd 资源（两个不堆叠的，一个堆叠的），用于实现 4 路存储冗余。这就意味着一个 4 节点的集群，即使三个节点同时出现故障，也能保证提供正常的服务。

使用下图来解释这个概念。

图 8.2 Pacemaker 集群中的 DRBD 堆叠资源

在这个例子中，alice, bob, charlie,和 daisy 组成两个量节点的 Pacemaker 集群。Alice 和 bob 组成的集群名为左，并复制两点之间的 drbd 资源，而 charlie 和 daisy 在另外一个独立的单元中用途是一样的，名字为 right。而堆叠的 drbd 资源则连接两个集群。

备注

由于 Pacemaker 集群管理器的限制，需要是 Pacemaker v1.0.5，这是不可能创建单个 4 节点集群而不启用 CIB 功能，这是一个超级进程，不建议一般用途使用。这个可能会在新发布的 Pacemaker 中解决。
为建立这样的配置，首先需要按照 6.15 节中“创建三节点的设置”配置和初始化资源（除了远程一般的 drbd 配置为堆叠，而不是本地集群）。之后配置 Pacemaker 使用下面的 CRM 配置，并启动 left 集群：

```
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_left:start

order o_drbd_left_before_stacked_left \

inf: ms_drbd_left:promote ms_drbd_stacked_left:start

如果创建一个名为/tmp/crm.txt 的临时文件，可以使用如下命令导入活动集群的配置：

crm configure < /tmp/crm.txt

在加入这些配置到 CIB 后，Pacemaker 会执行以下动作：

1、 在 alice 和 bob 间启动 drbd 资源 left 复制并提升资源角色为主。

2、 启动 IP 地址 10.9.9.100（Alice 或 Bob，取决于这些持有的 left 资源的主角色）。

3、 启动 drbd 资源堆叠在刚刚配置 IP 地址的同一节点上。

4、 提升堆叠的 drbd 资源为主角色。

现在继续配置集群 right，通过创建以下配置：

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_right:start

order o_drbd_right_before_stacked_right \
```

inf: ms_drbd_right:promote ms_drbd_stacked_right:start

在加入这些配置到 CIB 后，Pacemaker 会执行以下动作：

1、 在 charlie 和 daisy 间启动 drbd 资源 right 复制并提升资源角色为主。

2、 启动 IP 地址 10.9.9.100（charlie 或 daisy，取决于这些持有的 right 资源的主角色）。

3、 启动 drbd 资源堆叠在刚刚配置 IP 地址的同一节点上。

4、 保留堆叠 stacked drbd 资源在次角色（因为 target-role="Slave"）

8.5 配置 drbd 在两个 SAN-backed 进行复制的 Pacemaker 集群

这个一个在 split-site 中经常采用的一个高级设置。它涉及到两个独立的 Pacemaker 集群，每个集群访问一个单独的网络存储区域（SAN）。Drbd 通过一个两点 site 间的 IP link 使用存在 SAN 上的复制的数据。

使用下图来解释这个概念。

图 8.3 基于 SAN 集群的 drbd 复制

在每个 site 的单个节点目前作为 drbd 对等节点是没有明确定义的--drbd 对等节点是浮动的，也就是说，drbd 绑定在虚拟 ip 地址上而不是特定的无理机上。

备注

这种类型的设置通常是和 drbd proxy 或者是基于传输的复制结合在一起的。由于这种类型的设置处理共享存储，为保证它能正常的工作，配置和测试 STONITH 是绝对至关重要的。

8.5.1 drbd 的资源配置

需要在 drbd.conf 中配置以下方式使 drbd 资源浮动：

resource <resource> {

...

```
  device /dev/drbd0;
  disk /dev/sda1;
  meta-disk internal;
  floating 10.9.9.100:7788;
  floating 10.9.10.101:7788;
}
```

Floating 这个关键字通常置于资源配中的<host>部分。在这种模式下，drbd 表示 ip 地址和 tcp 端口，而不是对等节点的名字。需要注意的是指定的地址必须为虚拟的集群的 IP 地址，而不是物理节点的 IP 地址，对于浮动的节点这点很重要。在这个例子中，在 split-site 配置两个浮动的地址属于两个独立的 IP 网络是必须的。要特别注意要允许路由器或防火墙在 drbd 节点之间进行复制和传输。

8.5.2 Pacemaker 资源的配置

一个 drbd 浮动节点的设置，在 Pacemaker 配置方面，包括如下项目（在两个参与的每一个 Pacemaker 集群）：

一个虚拟的集群 IP

一个主/从的 drbd 资源（使用 drbd OCF 资源代理）

确保 Pacemaker 以正确的顺序在正确的节点上上启动

要在一个浮动的对等节点的 2 集群节点上配置一个名为 mysql 的资源，使用复制 ip 地址 10.9.9.100，使用一下 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
```

在加入这些配置到 CIB 后，Pacemaker 会执行以下动作：

1、 启用 ip 地址 10.9.9.100（在 alice 或者是 Bob 上）

2、 根据所配置的 ip 地址调用 drbd 资源。

3、 提升其中一个 drbd 资源为主角色。

接着，为了创建与集群相配的配置，使用以下命令配置 Pacemaker：

```
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
```

在加入这些配置到 CIB 后，Pacemaker 会执行以下动作：

1、 启用 ip 地址 10.9.9.101（在 charlie 或者是 daisy 上）

2、 根据所配置的 ip 地址调用 drbd 资源。

3、 保留 drbd 资源为次角色（是由 target-role="Slave"确定的）

8.5.3 Site 的故障转移

在 split-site 配置中服务从一个 site 转移到另外一个是非常必要的。这可能是一个预定的过渡也可能是一个有灾难性后果的事件。如果是过渡期则是非常正常的，可以预期的时间，我们所期望也当然是这样：

在连接到集群的 site 上关于放弃资源，改变影响资源的 DRBD target-role，性要从主到从。这将关闭任何依赖的主要角色 DRBD 资源，降级并继续运行，并准备从新的主上接受更新。

连接到集群 site 上的接替资源，改变受影响的 DRBD 资源的目标角色的属性从到主。这将升级 DRBD 资源，启动任何其他 Pacemaker 的资源，根据 DRBD 资源的主角色，并复制到远程 site 的更新。

要故障切换回来，就是过程的反过来操作。

如果在活动的 site 上发生灾难性停电，site 显而易见的会置于 offline，而不在复制到备份的 site 上去。在这种情况下：

连接到集群上仍在运行的 site 资源，并改变受影响的 DRBD 资源的目标角色的属性从到主。这将升级 DRBD 资源，并根据 DRBD 资源主角色启动任何其他 Pacemaker 的资源。

当原来的 site 被恢复或重建，可能会再次连接的 DRBD 资源，并随后故障恢复使用相反的过程。

9 红帽集群集成 drbd

目录

本章介绍了如何在红帽集群高可用性集群中使用 DRBD 的复制存储。

备注

本指南使用非官方的红帽集群，是指其在历史上的多个正式的产品名称，包括红帽集群套件，红帽企业 Linux 的高可用性附加产品。

9.1 红帽集群的背景资料

9.1.1. Fencing

红帽集群，最初设计是为了共享存储集群，凭借节点的 fencing 放置并发、不协调的连接访问共享资源。红帽集群的 fencing 功能凭借 fencing 服务 fenced 和 fencing 代理，以脚本的形式实现的。

即使基于 DRBD 集群没有利用共同的存储资源，从而从 drbd 角度来看 fencing 不是必须的，红帽集群套房甚至仍然需要 fencing 基于 drbd 的配置。

9.1.2 资源组管理

资源组管理（rgmanager 或 clurgmgr）类似于 Pacemaker。它是集群管理套件和配置管理的应用程序的主要接口。

9.1.2.1 红帽集群资源

在红帽集群术语中单个的高可用应用程序、文件系统、IP 地址和类似的资源。

当资源上相互依赖，例如，NFS 导出取决于正在安装的文件系统-它们形成一个资源树，这是一种嵌套树资源中的另一个。内嵌套层资源可以承受参数从资源在外层嵌套层。在 Pacemaker 中不存在资源树的概念。

9.1.2.2 红帽集群服务

资源形成一个相互依存的集合，该集合成为服务。这不同于 Pacemaker，这样的集合成为一个资源组。

9.1.2.3 rgmanager 资源代理

资源代理被 rgmanager 调用，类似于在 Pacemaker 上的使用，在这个意义上，它们使用基于同一个在开放集群框架（OCF）所界定的 shell api，虽然 Pacemaker 采用了没有被定义

的框架的扩展。因此从理论上讲，资源代理和 Pacemaker 在一定程度是可以互换的—然而，在两个集群管理实际使用不同的但功能和任务相似资源代理。

红帽集群资源代理安装在/usr/share/cluster 下。和 Pacemaker OCF 资源代理的自己包含本身不同，一些红帽集群资源代理被分割到包含 a.sh 文件的 shell 代码和 a.metadata 文件包含 xml 资源代理元数据的文件。

Drbd 包含红帽集群资源代理。它以 drbd.sh 和 drbd.metada 的形式安装到传统的目录里面。

9.2 红帽集群配置

本节概述获得红帽集群运行所需的必要的配置步骤。准备集群配置是比较简单的，所有基于 drbd 的红帽集群都需两个节点（即为在红帽文档中所称的集群成员）和 fencing 设备。

备注

欲了解更多有关配置红帽集群的信息，请参阅红帽的红帽集群和 GFS 的文档。

9.2.1 cluster.conf 文件

RHEL 集群配置保存在一个单独的配置文件/etc/cluster/cluster.conf 中，可通过以下方面管理集群配置：

直接编辑配置文件。这是最简单的的方法。不需要出文本编辑之外的其他先决条件。

使用 GUI 系统配置集群。这是写在 Python 中使用 Glade 的一个 GUI 应用程序。它需要 X 显示（无论是直接在服务器控制台，或通过 SSH 隧道）的可用性。

使用基于 web 管理基础设施 conga。Conga 的汲取设施包含一个节点代理与本地集群管理器、集群资源管理器以及集群 LVM 守护进程、页面管理应用程序（luci）。可用一个简单的 web 浏览器来配置集群的通信。

9.3 在红帽 fail-over 集群中使用 drbd

备注

本节专门对红帽集群 fail-over 设置 drbd，而不涉及 GFS。GFS（和 GFS2）的配置，参见第 11 章“使用 GFS 整合 drbd”

这部分类似于低 8 章，整合 drbd 和 Pacemaker 集群，假定使用如下配置参数配置高可用 mysql 数据库：

drbd 的资源被用为名为 mysql 的数据库存储，并管理设备/dev/drbd0.

drbd 设备拥有一个被挂载/var/lib/mysql 下的 ext3 文件系统（默认为 mysql 的数据目录）

mysql 数据库利用文件系统，并监听在集群 IP 地址 192.168.42.1 上。

9.3.1 设置群集配置

要配置高可用 mysql 数据库，创建或者是修改包含以下配置项的/etc/cluster/cluster.conf 文件。

打开/etc/cluster/cluster.conf 文件，然后再资源中配置下列项目：

```
<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="10.9.9.180" monitor_link="1"/>
    <mysql config_file="/etc/my.cnf"
```

```
        listen_address="10.9.9.180"

        name="mysqld"/>

  </service>

</rm>
```

备注　　内嵌套资源引用另外一个<service/>是红帽集群的表达方式

资源依赖关系

例子假设为一个单卷的资源。

务必增加 config_version 属性，在配置完成之后，在 root 中找到 <cluster>元素。然后发出以下命令修改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，注意需要将<version>替换为你配置的最新的版本号。

备注

无论是 system-config-cluster GUI 配置还是 conga 基于 web 集群管理的设施会控告在 cluster.conf 文件中涉及 drbd 资源代理的集群配置。这是因为这个两个应用程序并不支持第三方扩展到集群基础设施提供的集群管理的 Python 管理包。

这样，当利用集群配置的 drbd 资源代理时，它是不建议使用 system-config-cluster 和 Conga 集群配置的。使用这些工具只监控集群的状态，以保证它的正常工作。

10 使用 drbd 结合 LVM

目录

本章涉及与管理 DRBD 技术结合的 LVM2。特别是涵盖：

使用 LVM 逻辑卷作为 drbd 的后端设备；

使用 LVM 物理卷的 drbd 设备；

结合这些概念，使用 drbd 实现一个使用 LVM 分层方法。

如果不熟悉这些，从 10.1 节“LVM 引子“可作为 LVM 的起点，同时也鼓励通过自己的方式了解除本节提供的以外的 LVM 的知识。

10.1 LVM 引子

LVM2 是逻辑卷管理的 linux 设备映射框架的背景下实施的。与原来的 LVM 出了名称和首字母缩写意外是没有任何的共同之处的。老的（现在追溯命名为“LVM1”）是过时了的，这在本节中未涉及。

使用 LVM 时，重要的是要了解其最基本的概念：

物理卷（PV）。一个物理卷（PV）是一个完全由 LVM 管理的底层块设备。PV 可是单个磁盘或者是单个分区。常见是在磁盘上创建一个分区专门被 linux 的 LVM 使用。

备注

这个分区类型“Linux LVM”（签名 0x8e）可用于识别单独使用的 LVM 分区。然而，这不是必须的—LVM 认可物理卷设备初始化后的书面签名的方式。

卷组（VG）。一个卷组（VG）是 LVM 的基本管理单元。一个 VG 可能包括一个或多个几的 PV。每个 VG 都有一个唯一的名称。VG 可以在运行时添加额外的 PV，或扩大现有的 PV。

逻辑卷（LV）。LV 可在运行期间的 VG 内创建，并且对内核的其他部分可视为常规块设备。因此，它们可用来保存一个文件系统，或者用于其他任何目的的块设备。LV 可以在线调整大小，也可以从一个 PV 转移到另外一个上（只要 PV 属于相同的 VG）。

快照逻辑卷（SLV）。快照是暂时的 LV 的时间点副本。创建快照几乎是瞬间就完成的，即使原来的 LV（原点卷）有大小数百 GiByte。通常情况下，快照需要的空间大大低于原来的 LV。

图 10.1。 LVM 的概述

10.2 使用逻辑卷（LV）作为 drbd 的后端设备

在 linux 上由于逻辑卷（LV）是一个简单的块设备，因此它可作为 drbd 设备使用。要以这种方式使用 LV，只需要创建然后像其他的设备一样进行初始化即可。

在这个例子中假设卷组（vg）名为 foo 已经在存在在两个 LVM 系统的节点上，就可以在该 VG 上使用 LVM 创建一个名为 r0 的 drbd 资源。

首先，创建逻辑卷：

```
lvcreate --name bar --size 10G foo

 Logical volume "bar" created
```

当然，需要在两个 drbd 集群的节点上都运行以上命令，之后在两个节点上都应该会产生一个名为/dev/foo/bar 的块设备。

然后，可以在资源中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;
    ...
  }
}
```

现在可以像没有 LVM 块设备一样继续启动资源。

10.3 DRBD 的同步过程中使用自动化 LVM 快照

尽管 drbd 是同步的，但是 SyncTarget 同步目标的状态会直到同步完成前都会是 Inconsistent 状态。如果在这种情况下 SyncSource 同步源发生故障（无法修复），则将是非常不幸的。有好数据的节点是死的，而有坏数据的节点是存活的。

drbd 使用 LVM 逻辑卷可通过创建一个自动化的快照启动同步，并删除成功同步后的相同的快照，从而减轻这个问题带来的损失。

为了使用快照自动同步，需要在资源中添加如下配置：

DRBD 自动化同步前的快照。

```
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";
  }
```

}

两个脚本解析$DRBD_RESOURCE$环境变量，drbd 将之传递给任何它调用的处理程序。snapshot-resync-target-lvm.sh 脚本在同步开始前创建一个包含所有资源的 LVM 的快照，在这种情况下，如果脚本失败，则不进行同步。

一旦同步完成，unsnapshot-resync-target-lvm.sh 脚本自动删除不需要快照，如果删除快照失败，快照将继续存在。

重要

任何时候出现 SyncSource 同步源不能修复的故障，并且你决定恢复到最新的对等节点的最近的快照，就可使用使用的 lvconvert –M 命令。

应该尽快审查摇摆不定的快照。一个完整的快照会导致快照本身和来源的失败。

10.4 使用物理卷 PV 配置 drbd 资源

为了准备一个使用物理卷作为 DRBD 资源，必须在 DRBD 设备上创建一个 PV 签名。为了做到这一点，在主角色的节点上运行一下命令：

pvcreate /dev/drbdX

或者是

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

备注

本例中假设是一个单卷的资源。

目前 LVM 扫描包括该设备在内的 PV 签名是必需的。为了重要做，需要编辑 LVM 的配置文件，其通常为 /etc/lvm/lvm.conf、找到包括过滤关键字在内的设备部分进行在线编辑，如果所有的 PV 都存储在 drbd 设备上，下面则为一个合适的过滤选项：

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

这种过滤器表示接受任何在 DRBD 设备上找到的签名，而拒绝其他的。

备注

默认情况下，LVM 扫描所有在/dev 的 PV 签名的所有块设备。这相当于 filter = ["a|.*|"].

如果想使用堆叠的资源为 LVM 的 PV，那就需要一个明确的过滤器配置。需要确保 LVM 在堆叠的资源上探测到 PV 签名，而忽略与之对应的地产的资源和后端块设备。本例假设底层的 drbd 资源使用设备从 0 到 9，而在堆叠的资源上则是 10 以上的：

filter = ["a|drbd1[0-9]|", "r|.*|"]

这种过滤器表示只接受在 drbd 设备/dev/drbd10 到/dev/drbd19 以内的 PV 签名，而忽略其他的。

在修改 lvm.conf 文件后，必须运行 vgscan 命令使 LVM 丢弃他的配置缓存并重新扫描设备的 PV 签名。

当然也可以使用不同的过滤器配置，以满足特定系统的配置。但是千万要注意：

接受（含）你希望作为 PV 的 drbd 设备

拒绝（不包含）相应底层的设备，以避免 LVM 发现重复的 PV 签名。

此外，通过设置禁用 LVM 缓存：

write_cache_state = 0

急用 LVM 缓存后，确保要删除/etc/lvm/cache/.cache.中任何过时的缓存。

必须对等节点上重复上述步骤。

当配置新的 PV 后，可以继续将它添加到卷组，或从它创建一个新的卷组。 在主节点上注意操作 drbd 资源是必须的。

vgcreate <name> /dev/drbdX

备注

虽然可在一个卷组中混用 DRBD 和非 DRBD 卷组，但是这不推荐并且也不具有任何实用价值。
当创建 VG 后，必须使用 lvcreate 命令操作逻辑卷（与非基于 drbd 的卷组）

10.5 在现有卷组中添加一个新的 DRBD 卷

有时可能想在 VG 中添加一个新的 drbd 支持的物理卷，每当这样做时，一个新的卷就应该添加到现有的资源配置中。这将保留所有的 VG 中 PV 的写保真度。

重要

如果 LVM 卷组是由 Pacemaker 管理的，可参见 10.7 节“高可用 LVM Pacemaker”。它必须在维护模式改变前对 drbd 进行配置更改。
在本例中，扩展包括额外卷的资源的配置：

```
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 配置在每个节点上都是相同，然后执行如下命令：

drbdadm new-minor r0 1

这将保证资源 r0 为新的第一卷。一旦新的卷被添加到复制流，就需要初始化然后将它添加到卷组中：

pvcreate /dev/drbd/by-res/<resource>/1

lvextend <name> /dev/drbd/by-res/<resource>/1

这将新增 PV /dev/drbd/by-res/<resource>/1 到<name>VG，维护整个 VG 的写保真度。

10.6 嵌套的 LVM 配置与 DRBD

如果有可能，稍微先进点可即使用 LV 作为 drbd 的后端设备，又同时使用 drbd 设备本身作为 PV。为提供一个例子可参见如下配置：

有两个分别名为/dev/sda1 和/dev/sda2 准备作为物理卷使用的分区

两个 PV 都可作为本地卷组的一部分

意图咋该 VG 中创建一个名为 r0 的 10Gib 的 LV

这个 LV 将成为 drbd 资源的本地也命名为 r0 的备份设备，对应设备未/dev/drbd0

该设备将是对另一个名为 replicated 的 VG 的唯一 PV

这个 VG 包含两个 LV，分别名为 foo（4GiB）he bar(6GiB)

为了使此配置，请按照下列步骤操作：

在/etc/ LVM/lvm.conf 中设置一个适当的过滤器选项：

indexterm:[LVM]indexterm:[filter expression (LVM)]

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

该过滤器表示接受任何 scsi 和 drbd 设备上的 PV 签名，而拒绝其他所有的。

在修改 lvm.conf 文件后，必须运行 vgscan 命令使 LVM 丢弃缓存的配置并重新扫描设备的 PV 签名。

通过设置禁用 LVM 缓存：

write_cache_state = 0

急用 LVM 缓存后，确保要删除/etc/lvm/cache/.cache.中任何过时的缓存。

必须对等节点上重复上述步骤。

现在就可初始化两个 scsi 的分区为 PV：

pvcreate /dev/sda1

Physical volume "/dev/sda1" successfully created

pvcreate /dev/sdb1

Physical volume "/dev/sdb1" successfully created

下一步是创建本地底层的 VG，包括刚才初始化的两个 PV：

vgcreate local /dev/sda1 /dev/sda2

Volume group "local" successfully created

现在可以创建 LV 作为 drbd 的后端设备：

lvcreate --name r0 --size 10G local

Logical volume "r0" created

在这里复制所有的步骤到对等节点上

然后编辑/etc/drbd.conf 创建一个名为 r0 的资源：

```
resource r0 {
  device /dev/drbd0;
  disk /dev/local/r0;
  meta-disk internal;
  on <host> { address <address>:<port>; }
  on <host> { address <address>:<port>; }
}
```

在创建好新的资源配置后千万要注意将 drbd.conf 复制到对等节点。

在此之后，可参见 5.4 节“首次启用资源”初始化资源（在两个节点上）。

之后在一个节点上晋升资源：

```
drbdadm primary r0
```

现在，在刚晋升的资源节点上，将 drbd 设备初始化为一个新的物理卷：

```
pvcreate /dev/drbd0
Physical volume "/dev/drbd0" successfully created
```

创建名为 replicated 的 VG，在同一节点上初始化 PV：

```
vgcreate replicated /dev/drbd0
Volume group "replicated" successfully created
```

最后，在新创建的 VG 上创建新的 LV：

```
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 将以/dev/replicated/foo 和 /dev/replicated/bar 的形式出现在本地节点上：

vgchange -a n replicated

0 logical volume(s) in volume group "replicated" now active

drbdadm secondary r0

然后，对等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

在此之后，块设备/dev/replicated/foo 和 /dev/replicated/bar 将在对等节点可用。

10.7 高可用 LVM Pacemaker

对等节点间传输卷组，并提供相应的逻辑卷的过程实现自动化。Pacemaker LVM 资源代理正是以此为目的而设计的。

为了管理存在的支持 drbd 卷组的管理下，在 shell 中运行以下命令：

DRBD 支持 LVM 卷组的 Pacemaker 配置

```
primitive p_drbd_r0 ocf:linbit:drbd \
  params drbd_resource="r0" \
  op monitor interval="15s"
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
```

当完成该设置后，Pacemaker 将自动使 r0 卷组在任何 drbd 资源当前为主角色节点上可用。

11 使用 drbd 结合 GFS

目录

本章概述以块设备共享全局文件系统（GFS）建立一个 drbd 资源的必要步骤。它涵盖了 GFS 和 GFS2。

如果在 drbd 上使用 GFS，必须配置 drbd 为双主模式。

11.1 GFS 引子

红帽全局文件系统（GFS）是红帽执行并发访问共享存储文件系统的。因为任何这样的文件系统，GFS 允许多个基点访问相同的存储设备，在读写方面不用担心数据损坏。它是通过分布式锁管理器（DLM）管理集群成员的并发访问的。

GFS 从一开始涉及的目的就是像传统的共享存储设备使用。因此 drbd 是完全可能在双主模式下将 GFS 作为复制的存储设备的。应用程序可能会受益于减少 drbd 整读写到本地存储，而不是通常配置运行的 SAN 设备 GFS。此外 drbd 对每个 GFS 文件系统增加一个额外的物理考虑，从而保证了冗余。

GFS 使用 LVM 的集群感知的变体，称作集群逻辑卷管理器或 CLVM。因此，一些并发存在在使用 drbd 作为 GFS 的数据存储和使用 drbd 作为传统 LVM 的 PV 之间。

通常 GFS 文件是集成在红帽的集群管理的框架内的，即红帽集群。本章介绍在红帽集群中使用 DRBD 和 GFS。

GFS、CLVM 和红帽集群在 RHEL 以及从它所衍生出来的 CentOS 中是可用的。同样对于 Debian GNU/Linux 也有相应的安装包。本章加上在红帽企业 Linux(RHEL)中运行 GFS。

11.2 创建适用于 GFS 的 drbd 资源

由于 GFS 是一个集群文件系统，期望从所有的集群节点进行并发的读和写，任何用于存储 GFS 文件系统的 drbd 资源都必须配置为双主模式。此外，它建议使用 drbd 的特性自动处理裂脑的自动恢复。将资源在启动后立即切换到主角色是非常必要的。为了做到这一些，如下在资源中配置如下行：

```
resource <resource> {

  startup {

    become-primary-on both;
```

```
    ...
  }
  net {
    allow-two-primaries yes;
    after-sb-0pri discard-zero-changes;
    after-sb-1pri discard-secondary;
    after-sb-2pri disconnect;
    ...
  }
  ...
}
```

一旦添加了上面的这些新选项，就需要像往常一样初始化资源。由于资源的 allow-two-primaries 选项设置的为 yes，因此能同时晋升两个节点的资源为主角色。

11.3 配置 LVM 认识的 drbd 资源

GFS 使用 CLVM，即 VLM 的集群变种版本，管理 GFS 使用的块设备。为了 drbd 使用 CLVM，需要配置 LVM。

使用集群锁。可在/etc/lvm/lvm.conf 中配置如下选项：

```
locking_type = 3
```

扫描 drbd 设备，使之从新识别基于 drbd 的物理卷（PV）。这同样适用于传统的（非集群）的 LVM。参见 10.4 节“使用物理卷 PV 配置 drbd 资源”了解详细内容。

11.4 配置支持 GFS 的集群

在创建新的 drbd 资源并完成初始化配置，在 GFS 集群的两个节点必须确保并启动一下系统服务：

cman（它同时启动 ccsd 和 fencend）

clvmd.

11.5 创建一个 GFS 文件系统

为了在双主的 drbd 资源上创建一个 GFS 文件系统，就必要先初始化它作为 LVM 的逻辑卷（LV）。

与传统的相反，非集群感知 LVM 配置，由于 CLVM 的集群感知性必须只在一个节点上完成下列步骤：

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

备注　　CLVM 会立即通知对等节点这些变化；对等节点发出的 LVS（或 lvdisplay）将列出新创建的逻辑卷。

这个例子假设一个单卷的资源。

现在可以创建实际的文件系统：

mkfs -t gfs -p lock_dlm -j 2 /dev/<vg-name>/<lv-name>

或者是一个 GFS2 文件系统：

mkfs -t gfs2 -p lock_dlm -j 2 -t <cluster>:<name>

/dev/<vg-name>/<lv-name>

在此命令中-j 选项是指为 GFS 所保持的 journals 的数量。这必须和 GFS 集群中节点数量相同的，由于 drbd 不支持对于两个的节点，因此这里设置的值为 2.

-t 选项只适用于 GFS2 文件系统，定义所表的名称。其格式如下<cluster>:<name>,这里<cluster>必须匹配在/etc/cluster/cluster.conf 中定义的集群名。因此，只有该集群成员被允许使用文件系统。相反<name>是集群中的独特的任意一个文件系统的名称。

11.6 使用 GFS 文件系统

在创建文件紫铜后，必须将之添加到/etc/fstab：

/dev/<vg-name>/<lv-name> <mountpoint> gfs defaults 0 0

而对于 GFS2 文件系统，只需要更改定义的文件系统的类型即可：

/dev/<vg-name>/<lv-name> <mountpoint> gfs2 defaults 0 0

千万不要忘记以上操作需要在两个节点上进行的。

在此之后，就可以启动 GFS 服务挂载新的文件系统（在两个节点上）：

service gfs start

从此时起，只要系统启动时 drbd 配置自动在 RHCS 服务器和 GFS 服务前，就可使用 GFS 文件系统就像在普通的共享存储上设置一样。

12 使用 OCFS2 整合 DRBD

目录

12.1 OCFS2 引子

Oracle 集群文件系统 2 版本（OCFS2）是有甲骨文公司开发的并发访问共享存储的文件系统。不同于其前任 OCFS 是专门设计并只适应 Oracle 数据库的有效载荷，OCFS2 是一个通用的实现了大多 POSIX 的标准的文件系统。ORACLE 真正应用集群（RAC）是 OCFS2 的比较常见的用例，但是 OCFS2 同样可以使用在负载均衡的 NFS 集群。

尽管最初设计是使用传统的共享存储设备，OCFS2 也同样是适用于部署双主 drbd 的。从文件系统读取的应用程序可能会受益于降低读延迟，因为 DRBD 从本地存储读和写，而不是正常运行的 SAN 设备 OCFS2 上。另外，DRBD 以增加格外的每个文件系统镜像增加 OCFS2 的冗余，而不是单一的文件系统镜像（也即共享）。

和其他的共享集群文件系统如 GFS，OCFS2 允许多个节点在读写模式下访问相同的存储设备，而不用担心数据损坏。同时使用分布式锁管理器（DLM）管理集群节点的并发访问。DLM 本身的使用一个虚拟文件系统(ocfs2_dlmfs)，它独立于实际 OCFS2 文件系统存在于系统。

OCFS2 即可以一种内在的集群通信层也可以挂载和卸载的文件系统进行操作来管理集群成员，或者是推迟 Pacemaker 集群执行这些任务。

OCFS2 是在 SUSE Linux 企业服务器(它主要是支持共享的集群文件系统)，centos、Debian GNU/Linux, 和 Ubuntu Server Edition 可用。Oracle 还同样为红帽企业版 linux（RHEL）提供了包。本章假定 OCFS2 是运行在 SUSE linux 企业服务器系统上的。

12.2 创建一个适用于 OCFS2 的 DRBD 资源

由于 OCFS2 是一个共享集群文件系统期望于并发读写在所有的节点上，所有的 drbd 资源都必须配置为双主模式用于存储 OCFS 文件系统。此外，建议使用 drbd 的一些特性，配置裂脑自动修复。而且启动之后立即切换到主角色也是必要的。要做到这些可用按照以下方式配置资源：

```
resource <resource> {
  startup {
    become-primary-on both;
    ...
  }
  net {
    # allow-two-primaries yes;
    after-sb-0pri discard-zero-changes;
    after-sb-1pri discard-secondary;
    after-sb-2pri disconnect;
    ...
  }
  ...
}
```

在初始化配置前是不建议设置 allow-two-primaries 为 yes 的，而是应该在初始化资源同步完成后这样做。

一旦你添加了这些选项用于刷新资源的配置，就可正常的初始化资源。当设置该资源的 allow-two-primaries 选项为 yes 后，就可在两个节点上都晋升资源为主角色。

12.3 创建 OCFS2 文件系统

现在使用 OCFS2 的 mkfs 命令创建文件系统：

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

这将在两个节点上的/dev/drbd0 上创建一个 OCFS2 文件系统，并设置文件系统的标签为 ocfs2_drbd0。也可以在 mkfs 时指定其他选项，请参阅 mkfs.ocfs2 系统手册了解详细内容。

12.4 Pacemaker OCFS2 管理

12.4.1 给 Pacemaker 增加一个双主模式的资源

添加一个存在的双主模式的 drbd 资源为 Pacemaker 资源资源，使用如下 drm 配置：

```
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
```

重要

注意 master-max=2 变量，对 Pacemaker 的主从设置启用双主模式。要求 drbd 的资源配置 allow-two-primaries 选项需要设置为 yes。否则，Pacemaker 会在资源校验时报配置错误。

12.4.2 Pacemaker 增加对 OCFS2 的管理能力

为了管理 OCFS2 和内核分布锁管理（DLM），Pacemaker 使用一下三种不同的资源代理：

ocf:pacemaker:controld---Pacemaker 的 DLM 接口

ocf:ocfs2:o2cb---Pacemaker 对 OCFS2 集群管理的接口

ocf:heartbeat:Filesystem---当配置 Pacemaker 克隆支持集群文件系统的通用文件系统管理的资源代理。

使用一下 crm 配置，在所有的 Pacemaker OCFS2 集群管理的节点上创建一克隆组的资源：

```
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

一旦完成这个配置，Pacemaker 会在集群所有的节点上启动 controld 和 o2cb 资源类型的实例。

12.4.3 给 Pacemaker 添加 OCFS2 文件系统

虽然在 clone 模式下 Pacemaker 使用传统的 ocf:heartbeat:Filesystem 资源代理管理 OCFS2 文件系统。为使 OCFS2 文件系统在 Pacemaker 的管理下，使用一下 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

注意

12.4.4 增加要求的 Pacemaker 约束管理

这个例子假设一个单卷的资源 **OCFS2 文件系统**

为了将所有的 OCFS2 相关的资源和 clone 的资源配合在一起，对 Pacemaker 配置添加一下约束：

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

12.5 传统的 OCFS2 管理（不含 Pacemaker）

重要

在本节中介绍的信息适用于传统系统，在 Pacemaker 中 OCFS2 的 DLM 不可用。它被保存在这里仅供参考。新装置应始终使用 Pacemaker 的方法。

12.5.1 配置集群支持 OCFS2

12.5.1.1 创建配置文件

OCFS2 使用中央配置文件/etc/ocfs2/cluster.conf。

当创建 OCFS2 集群，一定要添加 hosts 到集群配置中。集群相互通信默认端口（7777）通常是可选择的。如果你选择了任何其他的端口，一定要注意选择的 drbd 端口不能和其他的相冲突。

如果感觉直接编辑 clust.conf 不方便，也可以使用 ocfs2console 图像化配置工具进行配置。无论选择那种方法，your /etc/ocfs2/cluster.conf 文件都会大致如下：

```
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

当配置好集群的配置，使用 scp 将其拷贝到集群的其他的两个节点。

12.5.1.1 配置 O2CB 驱动程序

======的 SUSE Linux Enterprise 系统

在 SLES 中，可能会利用 O2CB init 脚本的配置选项：

/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

======的 Debian GNU/ Linux 系统 Debian 上，配置选项/ etc/init.d/o2cb 是不可用的。相反，重新配置 OCFS2 工具包的驱动：

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

12.5.2 使用 OCFS2 文件系统

当完成集群配置并创建文件系统，就可像其他的文件系统一样挂载：

mount -t ocfs2 /dev/drbd0 /shared

内核日志（使用 dmesg 命令）应包含类似的如下行：

ocfs2: Mounting device (147,0) on (node 0, slot 0) with ordered data mode.

在此之前，在读写模式下，应该同时在两个节点上挂载 OCFS2 文件系统。

13 使用 Xen 整合 drbd

目录

本章概括了在 XEN 虚拟环境中使用 drbd 作为虚拟块设备。

13.1 Xen 引子

Xen 是最初是由英国剑桥大学开发，后由 XenSource 公司（现在是 Citrix 的一部分）维护的一个虚拟化的框架。它包含在大多数的 linux 发行版，如 Debian GUN/Linux（从第四版后）、SUSE 企业 linux 服务器（从第十版后）、红帽企业 linux（从第五版之后）等等。

Xen 使用准虚拟化技术---虚拟化的方法涉及虚拟主机和客户虚拟机之间的高度合作---比传统的虚拟化解决方案（如典型的基于硬件的仿真）提高了选定的客户操作系统的性能。Xen 同时还支持适当虚拟化扩展 CPU，支持完整的硬件仿真，用 Xen 的说法，这就是众所周知的 HVM（硬件辅助虚拟机）。

注意

在写操作时，Xen 支持 HVM 的 CPU 扩展是 Intel 的虚拟化技术（VT，以前称为 Vanderpool）和 AMD 的 安全虚拟机（SVM，以前称为 Pacifica）。
Xen 支持实时迁移，这是指从一个屋里主机迁移到来宾操作系统中，而不造成中断。

当 drbd 资源作为 Xen 的复制虚拟块设备时，它可用在两台服务器上，然后配置为自动故障转移到一个包含全部内容的 domU 虚拟磁盘。在这种方式下，DRBD 不仅仅在 linux 服务器下（在非虚拟化的 drbd 技术的部署方案），而且在其他的任何操作系统都可使用 Xen 虚拟化技术实现冗余。在本质上，它包括 32 位和 64 位 intel 兼容的框架。

13.2 Xen 下设置 drbd 的模块参数

对于 Xen Domain-0 内核，建议加载 drbd 模块设置为 1.创建或者是打开 /etc/modprobe.d/drbd.conf 文件，然后输入如下行：

```
options drbd disable_sendpage=1
```

13.3 创建适用于 Xen VBD 的 DRBD 资源

将 drbd 资源配置为 Xen 虚拟化块设备是比较简单的。事实上，典型的 drbd 配置可用于任何目的。但是，如果想使 guest 实例实时迁移的话，就需要启用资源的双主模式。

```
resource <resource> {
  net {
    allow-two-primaries yes;
    ...
  }
  ...
}
```

对 Xen 来说是必须启用双主模式的，在启动迁移前，检查所有 VBD 资源配置的来源和目的地主机的写权限。

13.4 使用 DRBD 的 VBD

为使用虚拟块设备做 drbd 资源，必须在 en domU 的配置中添加如下行：

disk = ['drbd:<resource>,xvda,w']

这个配置例子中，命名为 resource 的 drbd 资源/dev/xvda 在读写模式下对 DomU 可用。

当然也可是一个单一的 domU 私用多个 drbd 资源。这种情况下，需要添加刚那个多的条目，并以分号将磁盘选项隔开。

注意 **正在配置一个全虚拟化（HVM）的 DomU**

有以下三种情况不能使用这种方法： **正使用图形化安装工具安装 DomU 并且高图形化安装不支持 drbd 的语法**

正配置一个 DomU，无 kernel、initrd 和额外的选项，以引导和 bootloader_args 使用不支持 drbd 的语法 Xen pseudo-bootloader,pseudo-bootloader。

pygrub+（之前的 Xen3.3）和 domUloader.py（使用 Xen 的 SUSE Linux Enterprise Server 10 的运）两个伪引导程序不支持 DRBD 的虚拟块设备的配置语法的例子。

pygrub 从 Xen3.3 开始，domUloader.py 版本 SLES 11 的附带支持这个语法。

在这种情况下，必须使用传统的 PHY 设备语法和 DRBD 设备的名称和资源，而不是资源的名称相关联的。然而，这需要管理 DRBD 之外的 Xen，这是一个 DRBD 资源类型提供的状态转换灵活的方式。

13.5 启动，停止和迁移 DRBD 支持的 domU

启动 domU。当配置 DRBD 支持 domU，就可以像其他的任何 DomU 一样的方式启动：

xm create <domU>

Using config file "+/etc/xen/<domU>+".

Started domain <domU>

在这个过程中配置为 VBD 的 drbd 资源将被晋升为主角色，并能被 Xen 访问。

停止 domU。操作简单如下：

xm shutdown -w <domU>

Domain <domU> terminated.

在 domU 成功关闭后，Drbd 资源会转变为次角色：

迁移 domU。通常使用 Xen 工具：

xm migrate --live <domU> <destination-host>

这种情况下，自动快速并连续的采取若干管理步骤。晋升目标主机的资源为主角色—本地主机上的实时迁移 domU 启动---当完成到目标主机的迁移本地主机的资源变成次角色。

事实上资源必须在两个主机上都处于主角色，这也是将资源配置为双主模式的首要原因。

13.6 DRBD/Xen 的内部整合

Xen 本身支持两种虚拟块设备类型：

Phy。此设备类型用于交互物理块设备，用于主机环境中，以实际透明的方式关闭 guest domU。

File。次设备类型用于提供给 guest domU 基于文件的块设备镜像。它的工作原理是创建一个循环块设备，从原始图像文件取出，然后交给该块设备，并以和 PHY 大致相同的方式关闭 domU。

如果在 domU 的 disk 选项中配置一个虚拟块设备使用任何前缀，而不是 phy:, file:,或者是没有前缀（这种情况下 Xen 默认使用 PHY 设备类型），在名为 Xen 的 scripts 目录中可找到比较常用的脚本/etc/xen/scripts。

DRBD 分布给 drbd 设备类型提供一个名为/etc/xen/scripts/block-drbd 的脚本。这个脚本处理本章之前所描述的那些必要的 drbd 的资源状态转换。

13.7 集成 Xen 和 Pacemaker

为了充分利用基于 DRBD 的 Xen VBD，推荐心跳管理相关的 domU 心跳的资源。

可以配置 Xen domU 为 Pacemaker 资源，并自动化资源的故障转移。需要使用 Xen OCF 资源代理实现它。如果已经使用本章所述的 DRBD Xen 相关的设备类型，将不需要配置单

独的 drbd 资源给 Xen 集群资源使用。相反，block-drbd 辅助脚本会帮你完成所有的必要的资源转换。

当连接时裂脑受害者将立即改变连接状态为 SyncTarget，并被主节点覆盖其余节点的修改。

第五篇 优化 DRBD 性能

14 测试块设备的性能

目录

14.1 测量吞吐量

测试 DRBD 对系统的 I/O 吞吐量时，对而系统而言绝对吞吐量是无关紧要的。更意思的是 drbd 对 I/O 性能的影响是相对的。

注意

在本节中所描述的测试是侵入式的，通过覆盖数据导致 drbd 之间不同步。很重要的是测试要在可以在临时的测试完成后可以丢失的卷上进行。
I/O 吞吐量的测试是通过写入大块数据到一个设备上，测试系统完成写操作所消耗的时间。可以使用比较实用的工具 dd，它在新版本中有相应的内置吞吐量的估计。

下面是一个基于 dd 吞吐量的简单标准，假设对一个当前连接并在两个节点都处于次角色的名为 test 临时资源：

```
TEST_RESOURCE=test

TEST_DEVICE=$(drbdadm sh-dev $TEST_RESOURCE)

TEST_LL_DEVICE=$(drbdadm sh-ll-dev $TEST_RESOURCE)

drbdadm primary $TEST_RESOURCE

for i in $(seq 5); do
```

```
  dd if=/dev/zero of=$TEST_DEVICE bs=512M count=1 oflag=direct
done
drbdadm down $TEST_RESOURCE
for i in $(seq 5); do
  dd if=/dev/zero of=$TEST_LL_DEVICE bs=512M count=1 oflag=direct
done
```

本测试只是写一个大小为 512M 的块到 drbd 设备，然后进行对比和测试。每个测试都进行 5 次，然后取平均值。相关结果都是通过 dd 进行吞吐量的测试的。

备注

对新启用的 drbd 设备，在第一次运行 dd 时刻明显发现器性能有所下降，这主要是因为并不引人注意的“cold”日志引起的。

14.2 测量延迟

延迟量和吞吐量是基于完全不同的标准的：在 I/O 延迟测试中，写一个非常小的数据块（理想情况下，为该系统能处理的最小数据块），并观察完成写入所需的时间。该过程通常反复数次使之在正常的范围波动。

正如吞吐量测试，I/O 延迟测试也可使用 dd 进行测试，尽管有不同的设置以及完全不同的观察重点。

下面是一个基于 dd 吞吐量的简单标准，假设对一个当前连接并在两个节点都处于次角色的名为 test 临时资源：

```
TEST_RESOURCE=test
TEST_DEVICE=$(drbdadm sh-dev $TEST_RESOURCE)
TEST_LL_DEVICE=$(drbdadm sh-ll-dev $TEST_RESOURCE)
drbdadm primary $TEST_RESOURCE
```

dd if=/dev/zero of=$TEST_DEVICE bs=512 count=1000 oflag=direct

drbdadm down $TEST_RESOURCE

dd if=/dev/zero of=$TEST_LL_DEVICE bs=512 count=1000 oflag=direct

在 drbd 设备上测试写 1000 个 512 字节的数据块，之后对后端的设备进行对比。512 字节的块大小是 linux 所支持的做小的块（S390 以外的所有架构）。

重要的是要了解到吞吐量产生的测试和 dd 测试是完全不相干的。重要的进行 1000 次写所耗费的时间。这段时间除以 1000 即是单一文件的延迟写时间。

15 优化 drbd 的吞吐量

目录

本章涉及优化 DRBD 的吞吐量。为实现该目的，本文结合硬件的注意事项提出调整优化和细节吞吐量的建议。

15.1 硬件注意事项

Drbd 的吞吐量收带宽和底层 I/O 子系统（磁盘、控制器和相应的缓存），以及复制网络的带宽影响。

I/O 子系统的吞吐量。I/O 子系统的吞吐量是在一定程度上确定的，为以并行写入磁盘的数量。通常单一的、接近合理的、scsi 或 sas 磁盘以 40MB/s 的流写入单个磁盘。当部署一个条带化配置时，I/O 子系统以并行方式写磁盘，有效的多条带化配置则为单个磁盘吞吐量的乘积。因此，40MB/s 的磁盘支撑在 raid0 或者是 raid-1+0 三条带中 120MB/s，或者是 5 条带的 200MB/s。

备注

对新启用的 drbd 设备，在第一次运行 dd 时刻明显发现器性能有所下降，这主要是因为并不引人注意的“cold”日志引起的。
磁盘镜像（RAID-1）对吞吐量的影响对应比较小。校验（RAID- 5）的条带化的磁盘对吞吐量的影响，则往往是不利的。

网络吞吐量。网络吞吐量通常是目前网络中以路由/交换机等设置所支持的流量。尽管 drbd 允许通过专用链接传输，但是这还是主要的 drbd 复制环节。因此，网络吞吐量改善可通过切换到更好的网络协议（如 10G 太网）或者通过聚合多个网络的连接绑定到指定的 linux 的网络驱动设备上。

15.2 期望的吞吐量开销

当估算吞吐量和 drbd 相关的开销时，重要是考虑一下自然的限制：

drbd 的吞吐量受原始 I/O 子系统的限制。

drbd 的吞吐量受可用网络带宽的限制。

两个之间的最小值就是 drbd 可支持的最大吞吐量。Drbd 减少额外的吞吐量开销大约小于最大吞吐量的 3%。

例如，考虑两个集群节点包含 I/O 子系统能力为 200MB/s，之间可以通过一个千兆以太网的链路。千兆以太网可以预计将产生 110MB/s 的 tcp 连接的网络吞吐量，从而在这个配置中的瓶颈就是所预计 drbd 的最大吞吐量为 107MB/s。

相反，如果 I/O 子系统的写入能力只有 100MB/s，那么它就成为了瓶颈，因此可以预计 drbd 的最大吞吐量为 97MB/s。

15.3 调整建议

Drbd 提供的大量的配置选项，这些都可能对系统的吞吐量产生影响。本节中对吞吐量的调整提出一些建议。然后，由于吞吐量在很大程度上依赖于硬件，在这里所描述影响可能对不同的系统有极大的不同。需要了解到这里的建议不是“灵丹妙药”，不能解决所有的吞吐量的瓶颈。

15.3.1 设置 max-buffers 和 max-epoch-size

这些选项会影响次节点上写的性能。max-buffers 是 drbd 数据写入到磁盘分配的缓冲区的最大数，而 max-epoch-size 的大小是两个写瓶颈的最大允许写的数量。在大多数情况下，应该同时设置两个选项，并设置为相同的值。二者默认为 2048，将其设置在 8000 应该是最合理的利用高性能硬件 raid 控制。

```
resource <resource> {
  net {
    max-buffers 8000;
    max-epoch-size 8000;
    ...
  }
  ...
}
```

15.3.2 调整 I/O unplug watermark

在正常操作期间 I/O unplug watermark 是可调整的，它影响 I/O 子系统控制器多久是“kicked”（被迫进程等待 I/O 请求）。对该选项的设置没有统一的建议，它的设置很大程度上是依赖于硬件的。

一些控制器在“kecked”频繁时表现比较好，所以这些控制器设置的较低是比较有意义的，甚至于低于 drbd 允许的最低值（16）。当单独离开时其他表现的比较好，对于这些控制器设置 max-buffers 高点是可取的。

```
resource <resource> {
  net {
    unplug-watermark 16;
    ...
  }
  ...
}
```

15.3.3 调整 TCP 发送缓冲区的大小

TCP 发送缓冲区是内存缓冲区传出的 TCP 的流量。在默认情况下它设置大小为 128KiB。适用于高吞吐量网络使用（如专用的千兆以太网或者是负载均衡的连接），它增大到 512KiB 或者更多是比较有意义的。不推荐发送缓冲区超过 2MiB 的大小（也不可能带来任何吞吐量的提高）。

```
resource <resource> {
  net {
    sndbuf-size 512k;
    ...
  }
  ...
}
```

DRBD 还支持发送缓冲区的自动调节。启动该功能后，drbd 会动态的选择一个合适的 TCP 发送缓冲区的大小。TCP 发送缓冲区自动调节可以将将缓冲区的大小设置为 0.

```
resource <resource> {
  net {
    sndbuf-size 0;
    ...
  }
  ...
}
```

15.3.4 调整活动日志的大小

如果使用 drbd 的应用程序写操作比较频繁的写到一个设备上，它通常最好使用一个相当大的活动日志。否则，频繁的元数据更新会影响写性能。

```
resource <resource> {
  disk {
    al-extents 3389;
    ...
  }
  ...
}
```

15.3.5 禁用 barriers 和磁盘刷新

警告

本节中所述的建议只适用于非易失性（battery backed）控制器缓冲系统。 系统配置 battery backed write cache 是都带内置的手段保护数据以面对停电。这种情况下，处于同样的目的它允许禁用 drbd 自己的保护措施。这有利于吞吐量的提升。

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

16 优化 DRBD 的延迟

目录

本章涉及优化 DRBD 的延迟。为实现该目的，本文结合硬件的注意事项提出调整优化和细节延迟的建议。

16.1 硬件注意事项

DRBD 的延迟收 I/O 子系统延迟（磁盘、控制器和对应的缓存）和复制网络延迟的影响。

I/O 子系统延迟。I/O 子系统的延迟主要体现在磁盘的转速上，因此利用高转速的磁盘可有效减少 I/O 子系统的延迟。

同样，使用支持高速写缓存池（BBWC）减少写完成的时间，也能降低写入延迟。最合理的存储子系统与某种形式的高度缓存池，并允许管理员配置词缓存的那一部分用于读取和写入的操作。推荐的方法是禁用磁盘的完全读缓存并对磁盘写缓存都使用内存缓存。

网络延迟。网络延迟本质上是数据包在主机之间往返的时间。它是由多项因素影响，其中大多数是专用的、使用与 drbd 复制连接无关的紧密的网络连接。因此，在千兆以太网间的 100 到 200 微秒（μs）RTT 秩序报的延迟是可以接受的。

网络延迟可能只能使用低延迟的网络协议降低延迟，如通过 Dolphin 的 SuperSockets 在 Dolphin Express 运行 drbd。

16.2 期望的延迟开销

当估算结合 drbd 的延迟的开销时，对吞吐量有一些重要的自然的限制需要考虑：

drbd 延迟受原始 I/O 子系统的限制。

drbd 延迟受可用网络带宽的延迟的限制。

两者之后为 drbd 的理论的最低延迟。Drbd 预计会带来不到 1%的额外延迟开销。

加入本地磁盘子系统写延迟为 3ms 而网络延迟为 0.2ms，那么 drbd 的延迟以及就为 3.2ms 或者比仅仅写本地磁盘有大约一个 7%的延迟。

备注

延迟还会受到一些其他的因素的影响，包括 CPU 高速缓存为命中、上下文切换等。

16.3 调整建议

16.3.1 设置 DRBD 的 CPU 掩码

DRBD 允许其内核线程设置一个明确的 CPU 掩码，这对程序是特别有利的，否则会与 drbd 的 CPU 周期进行竞争。

CPU 掩码是其在二进制的的最低有效位表示第一个 CPU、第二低表示第二个等等。一个位掩码意味着 drbd 对应的可用的 CPU，否则该位置就必须别清空。例如 1（00000001）表示 drbd 只能使用第一个 CPU，而 12（00001100）则表示 drbd 可使用第三和第第四个 CPU。

例如，CPU 资源的掩码配置如下：

```
resource <resource> {
  options {
    cpu-mask 2;
    ...
  }
  ...
}
```

重要

当然为了减少 drbd 与其他程序的 CPU 竞争，必须配置其他应用程序不使用 drbd 使用的 CPU。
某些应用程序也可能提供像 drbd 这样配置的配置的文件。而其他的应用程序的 init 脚本中

包括 taskset 命令。

16.3.2 修改网络的 MTU

当基于块的（而不是基于 Extent 的）文件系统出在 drbd 的上层，可能改变复制网络的最大传输单元（MTU）大小是有益的，其默认为 1500 字节。通俗的将这就是“启用 Jumbo 帧”。

备注

基于块的文件系统，包括 ext3、ReiserFS（版本 3）以及 GFS。基于 Extent 的文件系统，

如 XFS、Lustre 和 OCFS2。基于 Extent 的文件系统只有当拥有大量的文件时启用 jubmbo 帧就会比较有益。
使用下面的命令可以改变的 MTU：

ifconfig <interface> mtu <size>

或者

ip link set <interface> mtu <size>

<interface>是指用于 DRBD 的复制到网络接口。<SIZE>典型值为 9000（字节）。

16.3.3 启用 I / O 期限调度

当结合高性能使用时，回写启用硬件的 raid 控制器，drbd 的延迟可能会因使用 I/O 调度的期限而大大减少，而不是使用 CFQ 的调度。后者通常在最新的内核配置中（2.6.18 之后）默认是启用的。

通过 sysfs 虚拟文件系统修改 I/O 调度配置，挂载在/sys 下。调度程序配置在 /sys/block/<device>中，这里<device>是使用的后端设备。

可通过如下命令启用最后期限调度：

`echo deadline > /sys/block/<device>/queue/scheduler`

然后还可以通过设置下面的提供额外的延迟效益的值：

禁用前合并：

echo 0 > /sys/block/<device>/queue/iosched/front_merges

减少读取 I / O 期限为 150 毫秒（默认为 500ms）：

echo 150 > /sys/block/<device>/queue/iosched/read_expire

减少写入 I / O 的截止日期为 1500 毫秒（默认为 3000ms）：

echo 1500 > /sys/block/<device>/queue/iosched/write_expire

如果这些对延迟有所改进，那么要想永久使用的话，也可在系统启动时自动设置。Debian 和 Ubuntu 系统通过 sysfsutils 和配置文件/etc/sysfs.conf 提供该功能。

也可通过内部命令设置一个全局性的 I/O 调度。编辑引导装载程序配置（如果使用 grub 引导的 haunted，通常在/boot/grub/menu.lst 中），同时添加 elevator=deadline 选项设置内核启动选项。

第六篇 更多有关 drbd 的知识

17 DRBD 的内部

目录

17.5 对等节点的 fencing 接口

本章提供一些关于 drbd 内部算法和结构的背景资料。目的是给那些对背景知识感兴趣的用户提供 drbd 的背景资料。对 drbd 的开发不做深入 drbd 的内部运作做深入的参考。因此请参阅 18.6 节“出版物”中所累出的文件，其对 drbd 的源代码进行了注释。

17.1 Drbd 元数据

一个专门存储 drbd 各种相关信息的区域。此元数据包括：

drbd 设备的大小

代标识（GI，详细信息参见第 17.2 节“代标识”）

活动日志（AL,详细介绍参见第 17.3 节“活动日志”）

快速同步位图（详细参见 17.4 节“快速同步位图”）

此元数据可以存储在内部和外部，使用暗中方法是在每个资源的基础上进行配置的。

17.1.1 内部元数据

配置资源使用内部元数据意味着 drbd 存在和实际生成数据相同的底层物理设备上。预留一个区域明确用户存储元数据。

优点。由于元数据和实际的数据有着千丝万缕的联系，如果硬盘出现故障管理员没有特殊的操作需求。这样元数据和实际都将一起丢失或者是一起恢复。

缺点。如果底层设备是一单个的物理硬盘（没有 raid 设置），内部元数据可能会对写入吞吐量产生负面影响。应用程序的的写请求可能会触发 drbd 元数据的更新，如在统一磁盘存储元数据，写操作可能会导致磁盘的读/写有额外开销。

重要

如果计划使用内部元数据结合现有的已有数据的底层的块设备，就必须考虑 drbd 的元数据

所需的空间。

否则，drbd 的资源创建后，新创的元数据将覆盖底层设备的其他元数据，并破坏进程中存在的问题。为了避免这种情况，如下做下面的事情：

扩展底层设备。任何逻辑卷管理设施（如 LVM）只要有可用的空闲空间提供相应卷的 grouplt，就可能被硬件存储解决方案所支持。

在底层设备上收缩现有的文件系统。这可能会也可能不会被文件系统所支持。

如果两者都不能的话，就使用外部元数据替代。

估算扩展底层设备以及收缩文件系统的大小，可参阅 17.1.3 节“元数据大小估算”。

17.1.2 外部元数据

外部元数据是单独存储、专用的块设备存储的生产数据。

优点。使用外部元数据对于写操作的延迟将有所改善。

缺点。元数据与实际的生产数据联系不紧密。这意味着，在硬件发生故障，破坏了生产数据（而不是 drbd 的元数据），需认为干预从幸存节点到被替换节点的数据同步。

如果出现以下情况，外部元数据是唯一可选择的：

正在使用的 drbd 复制的设备已经包含了有用的数据。

现有设备不支持扩展。

设备上的现有文件系统不支持收缩。

估算扩展底层设备以及收缩文件系统的大小，可参阅 17.1.3 节“元数据大小估算”。

17.1.3 元数据大小估算

可使用下面的公式精确估算出 drbd 元数据对空间的需求：

图 17.1。计算 DRBD 技术元数据的大小（完全）

Cs 是扇区中的数据设备的大小。

备注

可以检索设备的大小使用 blockdev --getsz <device>命令结果 Ms 也用扇区表示。要处以 2048（一个 512 字节的扇区大小，这是在 S390 外所有的 linux 平台上默认的）转换才 MB。

在实践中，可使用下面给出的一个很近似、合理的方式计算。注意在公式中，单位是兆字节，而不是扇区。

图 17.2。估计 DRBD 技术元数据尺寸（大约）

17.2 代标识

Drbd 使用代表示（GI）确定复制数据的“代”。

这是 drbd 的内部机制，用于：

确定两个节点是不是属于同一集群的事实（而不是意外连接的两个节点）

确定重新同步时同步的方向（如果必要的话）

确定是完全重新同步或者是部分重新同步是否满足需求

确定裂脑

17.2.1 数据的代

发送下列情况标志着 drbd 产生一个新的数据代：

最初设备的完全同步

一个断开连接的资源切换到住角色

主角色的资源断开连接

由此可以总结出，只要资源处于连接状态，两个节点的磁盘状态为 UpToDate，目前两个节点上数据生成是相同的。反过来也是如此。

每个新的数据生成的标识都由一个 8 字节的、全局唯一的标识符（UUID）标志。

17.2.2 代标识的元组

Drbd 保持四个当前的和历史数据代在当地资源元数据的信息：

当前的 UUID。从本地节点的角度看，这是当前数据代的代标识符。当一个资源连接并完全同步时，目前两节点之间的 UUID 是相同的。

位图的 UUID。这是磁盘的同步位图是跟踪更改的代的 UUID。只有在断开模式磁盘上的同步位图本身才与该标识符有关。如果资源是连通的，此 UUID 总是空（零）。

两个历史 UUID。这是前几代的两个数据的标识符。

总的来说，这四个项目被称为一代标识元组，或简称 GI tuple。

17.2.3 生成标识符如何改变

17.2.3.1 开始一个新的数据代

当一个节点失去与对等节点的连接（无论是网络故障还是人为干预），drbd 通过以下方式修改本地的代标识：

图 17.3 GI 元组的变化在一个新的数据代开始

1. 给新的数据代创建一个新的 UUID。这将成为当前主节点新的 UUID。

2. 之前的 UUID 现在指代的位图跟踪到变化，它在主节点变成新的位图 UUID。

3. 在辅助节点上，GI 元组保持不变。

17.2.3.2 开始重新同步

DRBD 重新同步启动后，在本地生成标识符执行这些修改：

图 17.4 开始重新同步的 GI 元组的变化

1. 当前同步源的 UUID 保持不变。

2. 同步源位图的 UUID 是旋转的第一个历史 UUID。

3. 新的 Bitmap（位图） UUID 生成同步源。

4. 此 UUID 将成为新的当前同步目标的 UUID。

5. 位图和历史的 UUID 同步目标保持不变。

17.2.3.3 完成重新同步

重新同步结束时，进行以下更改：

图 17.5 重新同步完成 GI 元组的变化

1. 当前同步源的 UUID 保持不变。

2. 在同步源上的位图 UUID 翻转为第一个历史 UUID，那个 uuid 移动为第二历史项（任何现有的第二个历史条项将被丢弃）。

3. UUID 同步源位图，然后清空（归零）。

4. 同步的目标采用同步源的整个 GI tuple 元组。

17.2.4 如何 DRBD 使用代标识符

当节点间建立连接，两节点就交换目前可用的代标识，并进行相应的处理。存在一些可能的如下结果：

在两个节点上的当前 UUID 为空。本地节点检测到其当前的 UUID 和对等节点当前的 UUID 是空的。这是一个新配置的资源，没有进行完整初始同步。没有同步发生，必须手动启动。

目前的 UUID 在一个节点上空。本地节点的检测对等节点的当前 UUID 为空，而它自己则不是。这是一个新配置的资源，刚启动初始完全同步的正常情况，本地节点被选中作为初始同步源。 DRBD 在磁盘上的同步位图（这意味着它进行整个设备同步）现在设置的所有位，并开始作为一个同步源进行同步。（当地当前 UUID 为空，对等节点非空），相反情况下，DRBD 的执行相同的步骤，除非本地节点成为同步的目标。

当前 UUID 相等。本地节点检测到其当前的 UUID 和同行的当前的 UUID 非空且相等。当它处于次角色时发生了断开的情况，两个断开的节点都没有提升资源。没有同步发生，因次是没有必要。

位图的 UUID 与对等节点的当前的 UUID 匹配。本地节点检测到它的位图的 UUID 匹配对等节点当前的 UUID，同时对等节点的的位图的 UUID 为空。这是当本地节点为主角色次要节点故障后的正常和预期发生的。这意味着在此期间对等节点从来没有成为主角色并在相同的数据生成的基础上都单独工作。现在 DRBD 正常启动，本地成为同步源节点，后台重新同步。反过来说，如果本地节点检测到它的位图的 UUID 是空的，同时对等节点的位图与本地节点的当前的 UUID 匹配，那么本地节点的失败后这就是正常的和预期发生。DRBD 现在启动正常当地的节点成为同步目标，后台重新同步。

当前的 UUID 匹配对等节点的历史 UUID。本地节点的检测当前的 UUID 匹配对等节点的的历史 UUID 之一。这意味着，两个数据集同父，而且本地节点是最新的数据，保存在本地节点的位图的信息是过时的和不可用的。因此，一个正常的同步是不够的。本地节点成为同步源，DRBD 现在标识整个设备过时并初始化重新进行同步。在相反的情况下（本地节点的历史的 UUID 匹配对等节点当前的 UUID），DRBD 执行相同的步骤，除非本地节点成为同步的目标。

位图的 UUID 匹配，当前 UUID 不匹配。本地节点检测到当前的 UUID 和对等节点当前的 UUID 不同，而位图的 UUID 的匹配。这就是脑裂，但一个数据代具有相同的父。这意味着如果配置，DRBD 的调用裂脑自动恢复策略。否则，DRBD 断开连接，并等待人工脑裂进行处理。

当前和位图 UUID 都不匹配。本地节点检测到其当前的 UUID 和对等节点的 UUID 不同，同时位图的 UUID 也不匹配。这是脑裂且一个数据代具有不同的父，从而即使配置自动恢复策略，也没有实际意义。 DRBD 断开，并等待人工脑裂进行处理。

没有相匹配的 UUID。最后，DRBD 技术的情况下无法检测到两个节点的 GI tuples 元组一个都不匹配，它记录一个警告一无关的数据并断开。这是 DRBD 意外连接两个节点的集群，彼此之间并不认识。

17.3 活动日志

17.3.1 目的

Drbd 本地写操作支持写在 Drbd 本地块设备，同时也将写操作通过网络发送数据块。对所有的实际目的这两个动作都是同时发生。随机时序可能会导致一个写操作已经完成，但是通过网络的传输却未发生。

如果此时活动的节点故障，故障转移启动，则该数据块在两节点间不同步—它已在崩溃前写入失败的节点，但是尚未完成复制。因此，最终恢复节点时此块必须在随后的同步数据中删除。否则，失败的节点将是“一个超前写”的幸存节点，这将违反“全有全无”的存储原则。其实这不仅限于 drbd，这个问题是在几乎所有复制的存储配置中都存在的问题。许多其他的存储解决方案（就像 drbd 本身，在 0.7 之前的版本），因此需呀 i 在活动的节点故障后进行完全同步，然后再修复。

在 0.7 版本之前的 drbd 是与之不同的。活动日志（AL）存储在元数据区，保持“最近”被写入块轨道。也就是俗称的热程度。

如果暂时失败的节点处于主动模式发生同步故障，只有那些在 AL 突出的热程度，而不是完整的设备需要同步。这大大减少了主动节点崩溃后的同步时间，。

17.3.2 活跃程度

活动日志有配置参数：活跃度。每个活跃度增加 4MiB 数据量用于主崩溃后的传输。这个参数必须被理解为以下方面之间的折中：

大量的活跃度。为调高读写吞吐量，保持一个大型的活动日志。每一个新活跃度被激活时，旧的不活跃的活跃度被复位。这种转变需要元数据区的写操作。如果活跃度高，旧的活跃度的换出则比较少见，减少元数据的写操作可以提高性能。

很少的活跃度。为减少同步时间则保持比较小的活动日志，主节点故障并随之恢复。

17.3.3 选择合适的活动日志的大小

应根据给定同步速率所需的同步时间定义活跃度的值。活跃度可利用如下公式计算：

图 17.6 基于同步率和目标同步时间的活跃程度计算

R 为同步速率，单位为 MB/s，Tsync 为目标同步时间，单位为秒（s）。E 则为活跃度。

提供这样一个例子，假设集群有一个 I/O 子系统吞吐率 90MiByte/s，而配置速率为 30MiBytes/s（R=30），而同时想保持目标同步时间在 4 分钟或 240s（Tsync=240）

图 17.7 活跃度的计算基于同步率和目标同步时间（例如）

精确的计算结果为 1800，但为保持 DRBD hash 功能的 AL 在最好的程度需设置为素数，因此选择 1801。

17.4 快速同步位图

DRBD 使用的内部数据结构为快速同步位图，基于每一个资源之上保持块的同步（在两个节点上相同）或者不同步。在节点处于断开模式时它是唯一相关的资源。

在快速同步位图中，一个位代表磁盘上一个 4KIB 块数据，如果该位被清零，就意味着对等节点的相应的块仍然是同步的。这就意味着从断开以来没有被写入的时间。相反，如果该位置被设置，就意味该块被修改，需要在连接重新可用时进行重新同步。

Drbd 在断开连接的设备上检测写 I/O，并因此在 RAM 中开始设置快速位图同步的位—这样避免了开销较大的同步元数据的 I/O 操作。只有相应的块变冷（即“活动日志”届满），drbd 则对快速同步位图的位进行适当的修改。同样，如果资源被手动关闭，其余接上断开，则 drbd 刷新完整的快速同步位图为永久存储。

当对等节点恢复或重新建立连接，drbd 结合两个节点的位图信息，确定必须重新同步的数据集。同时 drbd 检查生成的标示符以确定同步的方向。

作为同步源的节点传输商定的块到对等节点，清除同步目标确认的修改的位图中同步位。如果重新同步中断（如另一个网络中断），随后将从它离开的地方进行恢复—--当然，与此同时任何额外块的修改被添加到重新同步数据集。

备注

重新同步也可以使用 drbdadm pause-sync 和 drbdadm resume-sync 命令进行暂停和恢复。然后操作并不轻松---中断重新同步会导致次节点的磁盘超过实际需要的不一致。

17.5 对等节点的 fencing 接口

Drbd 有一个明确定义的接口，fences 在对等节点以防止复制连接时被中断。drbd-peer-outdater 帮助，参考绑定 heartbeat 以实现该接口。然而，可很容易的实现对等节点 fencing 的辅助程序。

Fencing 帮助在下列情况相爱调用：

1. fence-peer handler 被定义为资源（或者 common）处理的一部分。

2. 资源的 fencing 选项设置为 resource-only 或者是 resource-and-stonith 。

3. 复制连接中断的时间对 Drbd 检测网络故障足够长。

当程序或者脚本作为 fence-peer handler 调用时，需要设置 DRBD_RESOURCE 和 DRBD_PEER 环境变量可用。它们分别包含受影响 drbd 资源的名字和对等节点的主机名。

任何对等节点的 fencing 帮助程序（或脚本）都必须返回以下退出的代码之一：

表 17.1 fence-peer handler 退出代码

退出代码	含义
3	对等节点的磁盘状态不一致（Inconsistent）
4	对等节点磁盘状态成功设置为 Outdated（或者是 Outdated 开始）
5	对等节点连接失败，对等节点不可达。
6	因为受影响的资源为主角色拒绝了对等节点变为 outdated
7	对等节点集群成功 fenced off。除非 fencing 设置受影响的资源为 resource-and-stonith，否则不可能发生。

18 获取更多信息

目录

18.1 商用 DRBD 支持

drbd 项目的赞助公司 linbit 提供对商用 drbd 支持、咨询和培训服务。

18.2 公开邮件列表

普通的使用问题可以反馈给公开邮箱：drbd-user@lists.linbit.com

可通过 http://lists.linbit.com/drbd-user 订阅邮件。

完整的列表文档：http://lists.linbit.com/pipermail/drbd-user。

18.3 公共 IRC 渠道

一些 Drbd 开发有时可从公开的 IRC 服务 irc.freenode.net，特别是一下渠道获得信息：

- #drbd,
- #linux-ha,
- #linux-cluster.

在 IRC 上可以公开讨论给 drbd 的开发层面提建议。

18.4 博客

本指南的共同作者之一 Florian Haas 的技术的博客

Martin Loschwitz a.k.a. madkiss，另一个 LINBIT 雇员和合作的 Debian DRBD 技术包的维护者，在德国保持一个个人博客。

Planet HA 是一个聚合信息的高可用性开发，技术顾问和用户的数量集中的博客。

18.5 官方 Twitter 账户

LINBIT 有一个官方 Twitter 帐户：linbit。

如果呼叫 DRBD，请注明#drbd hashtag。

18.6 刊物

Drbd 的作者撰写和发表关于 drbd 或者 drbd 的某一方面的论文。下面为一个简单的列表：

Lars Ellenberg. *DRBD v8.0.x and beyond*. 2007. Available athttp://www.drbd.org/fileadmin/drbd/publications/drbd8.linux-conf.eu.2007.pdf

Philipp Reisner. *DRBD v8 - Replicated Storage with Shared Disk Semantics*. 2007. Available athttp://www.drbd.org/fileadmin/drbd/publications/drbd8.pdf.

Philipp Reisner. *Rapid resynchronization for replicated storage*. 2006. Available athttp://www.drbd.org/fileadmin/drbd/publications/drbd-activity-logging_v6.pdf

18.7 其他有用的资源

维基百科有 DRBD 技术的项目 http://en.wikipedia.org/wiki/DRBD。

无论是 Linux - HA 的 wiki http://wiki.linux-ha.org/ 还是 ClusterLabshttp://www.clusterlabs.org/都有一些利用 DRBD 技术在高可用性集群有用的信息。

第七篇 附录

http://www.drbd.org/users-guide-emb/p-appendices.html