DRBD9 ユーザーズガイド

目次

はじめにお読みください

本書はDistributed Replicated Block Device (DRBD)を利用するための完全なリファレンスガイドです。同時にハンドブックとしても活用できるように編集してあります。

本書はDRBDプロジェクトのスポンサーである LINBIT がそのコミュニティ向けに作成しています。そしてコミュニティにとって有益であることを願って無償で公開しています。本ユーザーズガイドは、随時更新しています。DRBDの新しいリリースと同時に、その新機能の説明を追加する予定です。オンラインHTML版は https://links.linbit.com/DRBD9-Users-Guide で公開しています。

このガイドはDRBDの最新バージョンのユーザーを対象にしています。DRBDバージョン8.4をご使用の場合には https://links.linbit.com/DRBD84-Users-Guide から対応するバージョンをご利用ください。

本書に対する改善提案や誤りの指摘は drbd-user メーリングリスト へお寄せください。

本書は次の構成になっています。

- DRBDの紹介ではDRBDの基本的な機能を扱います。Linuxの I/OスタックにおけるDRBDの位置付け、DRBDの基本コンセプトなど、基礎となる事項を取り扱います。また、DRBDのもっとも重要な機能について説明します。
- DRBDのコンパイル、インストールおよび設定 ではDRBDのビルド方法、コンパイル済みのパッケージからのインストール方法、またクラスタシステムでのDRBDの運用方法の概要について説明します。
- DRBDの使い方 ではリソース設定ファイルと一般的なトラブルシューティングについて説明します。
- DRBDとアプリケーションの組み合わせではストレージのレプリケーションの追加やアプリケーションの高可用性のためDRBDを活用する方法を説明します。ここではDRBDとPacemakerクラスタ管理システムとの組み合わせだけでなく、LVMとの高度な組み合わせ、GFSとの組み合わせ、Xenによる仮想環境の可用性向上についても触れます。
- DRBDパフォーマンスの最適化ではパフォーマンスを向上させるDRBDの設定について説明します。
- さらに詳しく知る ではDRBDの内部構造を説明します。読者に有益と思われる他の情報リソースについても紹介します。
- 付録:
    - 最近の変更 はこれまでのDRBDバージョンと比較した、DRBD9.0での変更点の概要です。

DRBDトレーニングやサポートサービスにご興味のある方は sales@linbit.com または sales_us@linbit.com にお問い合せください。

DRBDの紹介

1. DRBDの基礎

Distributed Replicated Block Device (DRBD)は、ストレージのレプリケーション(複製)のためのソフトウェアで、シェアードナッシングを実現します。DRBDはサーバ間でブロックデバイス(ハードディスク、パーティション、論理ボリュームなど)の内容をミラーします。

DRBDによるミラーは次の特徴を持ちます。

- **リアルタイムレプリケーション**. 上位アプリケーションがデバイスのデータを書き換えると、 そのデータをリアルタイムでレプリケートします。
- **アプリケーションから透過的**. アプリケーションは、データが複数のホスト上に格納されていることを意識する必要はありません。
- **同期** または **非同期** の両方に対応同期でミラーリングを行う場合には、 すべてのホストのディスクへの書き込みが完了した後で、アプリケーションが完了通知を受け取ります。 非同期でミラーリングを行う場合には、 ローカルディスクへの書き込みが完了したときに、すぐにアプリケーションが完了通知を受け取ります。 この際、他のホストへの書き込みは後で行われます。

1.1. カーネルモジュール

DRBDのコア機能はLinuxのカーネルモジュールとして実装されています。OSのI/Oスタックの下位の階層でDRBDは仮想的なブロックデバイスを作ります。このために、DRBDは非常に柔軟で応用範囲が広く、さまざまなアプリケーションの可用性を高めるために利用できます。

その定義とLinuxカーネルアーキテクチャとの関連にもとづき、DRBDは上位レイヤに関して一切関知しません。このため、DRBDは上位レイヤに対しては何ら機能を付与できません。たとえば、DRBDはファイルシステムの障害を検出できません。またext3やXFSなどのファイルシステムに対してアクティブ-アクティブクラスタ機能を追加することもできません。

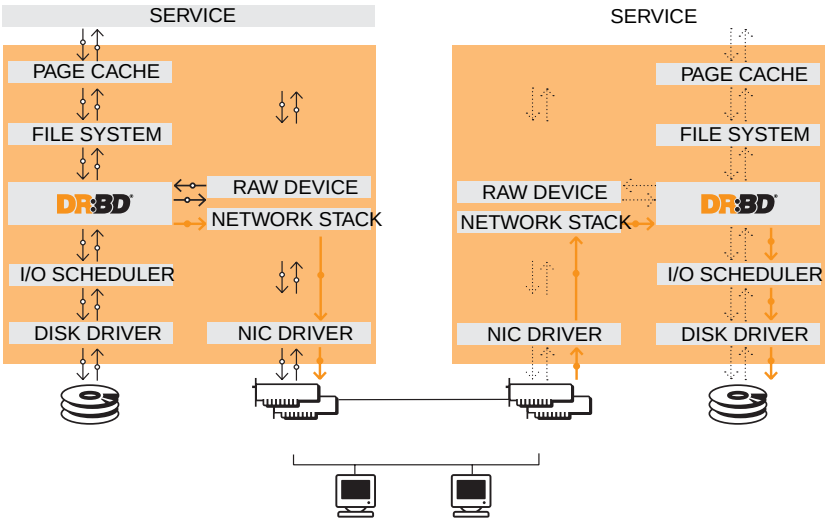


図 1. LinuxのI/OスタックでのDRBDの位置

1.2. ユーザー空間の管理ツール

DRBDにはカーネルモジュールと通信を行う管理ツールがいくつか用意されています。トップレベルのものからボトムレベルの順に以下に説明します。

drbdadm

DRBDプログラム群における高レベル管理ツールです。このコマンドは、DRBDの制御に必要なすべてのパラメータを /etc/drbd.conf から読み込み、 drbdsetup と drbdmeta のフロントエンドとして動作しま

す。drbdadm には -d オプションで起動する dry-run モードがあり、 実際にコマンドを実行することなく、drbdadm から呼び出される drbdsetup や drbdmeta のコマンドを表示します。

drbdsetup

カーネルにロードされたDRBDモジュールを設定します。 drbdsetup の全パラメータはコマンドラインで指定する必要があります。 drbdadm と drbdsetup を分離していることで最大限の柔軟性を確保しています。ほとんどのユーザーは drbdsetup を使う事はないでしょう。

drbdmeta

DRBDメタデータの作成、ダンプ、リストアなどを行うコマンドです。 drbdsetup と同様、ほとんどのユーザーはdrbdmeta を使うことはないでしょう。

1.3. リソース

DRBDでは、レプリケートするデータセットに関するさまざまな属性を総称して リソース と呼びます。リソースは以下の要素で構成されます。

リソース名

個々のリソースを区別するために、ホワイトスペース以外のUS-ASCII文字で表される任意の名前を与えることができます。

ボリューム

どのリソースも、複数のレプリケーションストリームを共有する ボリュームのうちの１つを構成するレプリケーショングループですDRBDは、リソース内のすべてのボリューム間で書き込みの忠実性を保証します。ボリュームは 0 から番号付けされ、1つのリソースにおいて、最大で65,535ボリュームまで可能です。ボリュームにはレプリケートされたデータセットを含み、DRBD内部で使用するメタデータのセットも含みます。

drbdadm コマンドでは、リソース内のボリュームを、リソース名とボリューム名を <resource>/<volume> のように記述して指定します。

DRBDデバイス

DRBDが管理する仮想的なブロックデバイスです。DRBDが管理する仮想的なブロックデバイスで、147のメジャー番号を持ち、minor番号は0から順次割り振られます。各DRBDデバイスは、リソース内の1つのボリュームに該当します。関連付けられたブロックデバイスは通常 /dev/drbdX の形式になり、 X はデバイス番号です。通常 udev は #/dev/drbd/by-res/#resource/vol-nr にあるリソース名とボリューム名を含むシンボリックリンクも作成します。drbdのインストール方法によっては、RPMベースのシステムで `drbd-udev`i パッケージをインストールする必要がある場合があることに注意してください。

初期のバージョンのDRBDは、NBDのデバイスメジャー番号43を勝手に使っていました。現在は147という番号が、DRBDデバイスメジャー番号として、 LANANA-registered に正式に登録されています。

コネクション

コネクション はレプリケートされるデータセットを共有する、2つのホスト間の通信リンクです。DRBD9では、各リソースが複数のホストで設定できますが、この場合、現在のバージョンではホスト間にフルメッシュ型の接続が必要です。つまり、リソース用に各ホストは他のホストへの接続が必要です。

drbdadm では、コネクションはリソース名とコネクション名(デフォルトでは対向のホスト名)で指定されます。

1.4. リソースのロール(役割)

DRBDのすべてのリソースは プライマリ または セカンダリ のどちらかのロール(役割)を持っています。

「プライマリ」と「セカンダリ」という用語は適当に選んだものではないことを理解してください。DRBD開発者は意図的に「アクティブ」と「パッシブ」という用語を避けました。プライマリ と セカンダリ は **ストレージ** の可用性に関する概念です。一方、 アクティブ と パッシブ は **アプリケーション** の可用性に関する概念です。ハイアベイラビリティクラスタ環境では、一般的にアクティブノードのDRBDは プライマリ になりますが、これが例外のないルールだということではありません。

- プライマリロールのDRBDデバイスでは、データの読み込みと書き込みが制約なく行えます。　この状態のストレージに対しては、ファイルシステムの作成やマウントが可能であり、 ブロックデバイスに対する下位デバイスI/Oやダイレクト I/O等も可能です。
- セカンダリ ロールのDRBDデバイスは、対向するノードのすべてのデータの更新を受け取りますが、 他からのアクセスは受け付けません。 つまり自ノードのアプリケーションからのアクセスについても、読み込みと書き込みの両方とも一切受け付けません。　読み込みすら受け付けない理由は、 キャッシュの透過性を保証するためです。 もしもセカンダリリソースが自ノードからのアクセスを受け付けると、この保証ができなくなります。

リソースのロールは、もちろん手動で切り替えできる他に、クラスタ管理アプリケーションの何らかの自動化アルゴリズムによって、または自動プロモーションでも切り替えられます。セカンダリからプライマリへの切り替えを 昇格 と呼びます。一方プライマリからセダンダリの切り替えは 降格 と呼びます。

2. DRBDの機能

本章ではDRBDの有用な機能とその背景にある情報を紹介します。いくつかの機能はほとんどのユーザーにとって重要な機能ですが、別のいくつかの機能については特定の利用目的においてのみ関係します。これらの機能を使うために必要な設定方法については、DRBDの使い方およびトラブルシューティングとエラーからの回復を参照してください。

2.1. シングルプライマリモード

シングルプライマリモードでは、個々のリソース は、任意の時点でクラスタメンバのどれか1台のみプライマリになれます。どれか1台のクラスタノードのみがデータを更新できることが保証されるため、従来の一般的なファイルシステム(ext3、ext4、XFSなど)を操作するのに最適な処理モードと言えます。

一般的なハイアベイラビリティクラスタ(フェイルオーバタイプ)を実現する場合、DRBDをシングルプライマリモードで設定してください。

2.2. デュアルプライマリモード

デュアルプライマリモードでは、すべてのリソースが任意の時点で両方のノード上でプライマリロールになれます。両方のノードから同一のデータに同時にアクセスできるため、分散ロックマネージャを持つ共有クラスタファイルシステムの利用がこのモードには必要です。利用できるファイルシステムにはGFSやOCFS2があります。

2つのノード経由でのデータへの同時アクセスが必要なクラスタシステムの負荷分散をはかりたい場合、デュアルプライマリモードが適しています。例えばライブマイグレーションが必要な仮想化環境などです。 このモードはデフォルトでは無効になっており、DRBD設定ファイルで明示的に有効にする必要があります。

特定のリソースに対して有効にする方法については、デュアルプライマリモードを有効にするを参照してください。

2.3. レプリケーションのモード

DRBDは3種類のレプリケーションモードをサポートしています。

プロトコルA

非同期レプリケーションプロトコルプライマリノードでのディスクへの書き込みは、自機のディスクに書き込んだ上でレプリケーションパケットを自機のTCP送信バッファに送った時点で、完了したと判断されます。システムクラッシュなどの強制的なフェイルオーバが起こると、データを紛失する可能性があります。クラッシュが原因となったフェイルオーバが起こった場合、待機系ノードのデータは整合性があると判断されますが、クラッシュ直前のアップデート内容が反映されない可能性があります。プロトコルAは、遠隔地へのレプリケーションに最も適しています。 DRBD Proxyと組み合わせて使用すると、効果的なディザスタリカバリソリューションとなります。詳しくはDRBD Proxyによる遠距離レプリケーションを参照ください。

プロトコルB

メモリ同期(半非同期)レプリケーションプロトコル。プライマリノードでのディスクへの書き込みは、自機のディスクに書き込んだ上でレプリケーションパケットが他機に届いた時点で、完了したと判断されます。通常、システムクラッシュなどの強制的なフェイルオーバでのデータ紛失は起こりません。しかし、両ノードに同時に電源障害が起こり、プライマリノードのストレージに復旧不可能な障害が起きると、プライマリ側にのみ書き込まれたデータを失う可能性があります。

プロコトルC

同期レプリケーションプロトコルプライマリノードでのディスクへの書き込みは、両ノードのディスクへの書き込みが終わった時点で完了したと判断されます。このため、どちらかのノードでデータを失っても、系全体としてのデータ紛失には直結しません。当然ながら、このプロトコルを採用した場合であっても、両ノードまたはそのストレージサブシステムに復旧できない障害が同時に起こると、データは失われます。

このような特質にもとづき、もっとも一般的に使われているプロトコルはCです。

レプリケーションプロトコルを選択するときに考慮しなければならない要因が2つあります。 データ保護 と レイテンシ遅延 です。一方で、レプリケーションプロトコルの選択は スループット にはほとんど影響しません。

レプリケーションプロトコルの設定例については、リソースの設定を参照してください。

2.4. 2重以上の冗長性

DRBD9では同時に2つ以上のクラスタノードにデータを書き込むことができます。

これはスタッキングを通じても可能でしたが、DRBD9では枠を超えて32ノードまで対応しました。 (実際の環境では、DRBDを通じて3重、4重、または5重の冗長化は、ダウンタイムを招く原因になることがあります。)

スタッキングを用いる場合との最大の違いは、同一の階層でデータのレプリケーションを行うのでパフォーマンスの低下が少ないことです。

2.5. リソースの自動プロモーション

DRBD9以前は、リソースを昇格するときには drbdadm primary コマンドを使用しました。DRBD9では、 auto-promote オプションが有効になっていればDRBDが自動的にリソースをプライマリにして、ボリュームをマウントや書き込みが可能になります。全ボリュームがアンマウントされると、すぐにセカンダリロールに変更されます。

自動プロモーションは、それが可能なクラスタ状態の時にのみ成功します。(つまり drbdadm primary コマンドの実行が成功する場合)そうでなければ、DRBD9以前と同様にマウントやデバイスのオープンは行えません。

2.6. 複数の転送プロトコル

DRBDは複数のネットワークプロトコルに対応しています。現在、TCPとRDMAの2つのトランスポートに対応しています。各トランスポートの実装はOSのカーネルモジュールを使用しています。

2.6.1. TCPトランスポート

DRBDのパッケージファイルには drbd_trasport_tcp.ko が含まれ、これによって実装されています。 その名の通り、TCP/IPプロトコルを使ってマシン間のデータ転送を行います。

DRBDのレプリケーションおよび同期フレームワークのソケットレイヤーは複数のトランスポートプロトコルに対応しています。

TCP over IPv4

標準的かつDRBDのデフォルトのプロトコルです。IPv4が有効なすべてのシステムで利用できます。

TCP over IPv6

レプリケーションと同期用のTCPソケットの設定においては、IPv6もネットワークプロトコルとして使用できます。アドレシング方法が違うだけで、動作上もパフォーマンス上もIPv4と変わりはありません。

SDP

SDPは、InfiniBandなどのRDMAに対応するBSD形式ソケットです。SDPは多くのディストリビューションでOFEDスタックの一部として利用されていましたが、現在は **非推奨** です。SDPはIPv4形式のアドレシングに使用しますインフィニバンドを内部接続に利用すると、SDPによる高スループット、低レイテンシのDRBDレプリケーションネットワークを実現することができます。

SuperSockets

スーパーソケットはTCP/IPスタック部分と置き換え可能なソケット実装で、モノリシック、高効率、RDMA対応などの特徴を持っています。きわめてレイテンシが低いレプリケーションを実現できるプロトコルとして、DRBDはSuperSocketsをサポートしています。現在のところ、SuperSocketsはDolphin Interconnect Solutionsが販売するハードウェアの上でのみ利用できます。

2.6.2. RDMAトランスポート

LINBITの drbd_transport_rdma.ko カーネルモジュールを使用する事もできます。このトランスポートはverbs/RDMA APIを使ってInfiniBand HCAsやiWARPが使えるNIC、またはRoCEが使えるNICでデータ転送をします。TCP/IPで使用するBSDソケットAPIと比較して、verbs/RDMA APIでは非常に低いCPU負荷でデータ転送が行えます。

2.6.3. 転送プロトコルの決定

TCPトランスポートのCPUロード/メモリ帯域が制約要因であれば、高い転送率が可能となります。 適切なハードウェアでRDMAトランスポートを使用すれば高い転送率を実現することができます。

転送プロトコルはリソースのコネクションごとに設定することができます。詳細はトランスポートプロトコルの設定を参照ください。

2.7. 効率的なデータ同期

同期ならびに再同期は、レプリケーションとは区別されます。レプリケーションは、プライマリノードでのデータの書き込みに伴って実施されますが、同期はこれとは無関係です。同期はデバイス全体の状態に関わる機能です。

プライマリノードのダウン、セカンダリノードのダウン、レプリケーション用ネットワークのリンク中断など、さまざまな理由によりレプリケーションが一時中断した場合、同期が必要になります。DRBDの同期は、もともとの書き込み順序ではなくリニアに書き込むロジックを採用しているため、効率的です。

- 何度も書き込みが行われたブロックの場合でも、同期は1回の書き込みですみます。このため、同期は高速です。
- ディスク上のブロックレイアウトを考慮して、わずかなシークですむよう、同期は最適化されています。
- 同期実行中は、スタンバイノードの一部のデータブロックの内容は古く、残りは最新の状態に更新されています。この状態のデータは inconsistent (不一致)と呼びます。

DRBDでは、同期はバックグラウンドで実行されるので、アクティブノードのサービスは同期によって中断されることはありません。

重要:データに不一致箇所が残っているノードは、多くの場合サービスに利用できません。このため、不一致である時間を可能な限り短縮することが求められます。そのため、DRBDは同期直前のLVMスナップショットを自動で作成するLVM統合機能を実装しています。これは同期中であっても対向ノードと_consistent_(一致する)一致するコピーを保証します。この機能の詳細についてはDRBD同期中の自動LVMスナップショットの使用をご参照ください。

2.7.1. 可変レート同期

可変レート同期(8.4以降のデフォルト)の場合、DRBDは同期のネットワーク上で利用可能な帯域幅を検出し、それと、フォアグランドのアプリケーションI/Oからの入力とを比較する、完全自動制御ループに基づいて、最適な同期レートを選択します。

可変レート同期に関する設定の詳細については、可変同期速度設定を参照してください。

2.7.2. 固定レート同期

固定レート同期の場合、同期ホストに対して送信される1秒あたりのデータ量(同期速度)には設定可能な静的な上限があります。この上限に基づき、同期に必要な時間は、次の簡単な式で予測できます。

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

図 2. 同期時間

t_{sync} は同期所要時間の予測値です。 D は同期が必要なデータ量で、リンクが途絶えていた間にアプリケーションによって更新されたデータ量です。 R は設定ファイルに指定した同期速度です。ただし実際の同期速度はネットワークやI/Oサブシステムの性能による制約を受けます。

固定レート同期に関する設定の詳細については同期速度の設定を参照してください。

2.7.3. チェックサムベース同期

DRBDの同期アルゴリズムは、データダイジェスト(チェックサム)を使うことによりさらに効率化されています。チェックサムベースの同期を行うことで、より効率的に同期対象ブロックの書き換えが行われます。DRBDは同期を行う前にブロックを_読み込み_ディスク上のコンテンツのハッシュを計算します。このハッシュと、相手ノードの同じセクタのハッシュを比較して、値が同じであれば、そのブロックを同期での書き換え対象から外します。これにより、DRBDが切断モードから復旧し再同期するときなど、同期時間が劇的に削減されます。

同期に関する設定の詳細はチェックサムベース同期の設定をご参照ください。

2.8. レプリケーションの中断

DRBDが正しく設定されていれば、DRBDはレプリケーションネットワークが輻輳していることを検出することが可能です。その場合にはレプリケーションを 中断 します。この時、プライマリノードはセカンダリとの通信を切断するので一時的に同期しない状態になりますが、セカンダリでは整合性のとれたコピーを保持しています。帯域幅が確保されると、自動で同期が再開し、バックグラウンド同期が行われます。

レプリケーションの中断は、データセンタやクラウドインスタンス間との共有接続で遠隔地レプリケーションを行うような、可変帯域幅での接続の場合に通常利用されます。

輻輳のポリシーとレプリケーションの停止についてほ詳細は輻輳ポリシーと中断したレプリケーションの構成をご参照ください。

2.9. オンライン照合

オンライン照合機能を使うと、2ノードのデータの整合性を、ブロックごとに効率的な方法で確認できます。

ここで 効率的 というのはネットワーク帯域幅を効率的に利用することを意味しています。照合によって冗長性が損なわれることはありません。しかしオンライン照合はCPU使用率やシステム負荷を高めます。この意味では、オンライン照合はリソースを必要とします。

一方のノード(照合ソース)で、低レベルストレージデバイスのブロックごとのダイジェストを計算します。DRBDはダイジェストを他方のノード(照合ターゲット)に転送し、そこでローカルの対応するブロックのダイジェストと照合します。ダイジェストが一致しないブロックはout-of-syncとマークされ、後で同期が行われます。DRBDが転送するのはダイジェストであって、ブロックのデータそのものではありません。このため、オンライン照合はネットワーク帯域幅をきわめて効率的に活用します。

このプロセスは、照合対象のDRBDリソースを利用したまま実行できます。これが_オンライン_の由来です。照合によるパフォーマンス低下は避けられませんが、照合およびその後の同期作業全体を通じてサービスの停止やシステム全体を停止する必要はありません。

オンライン照合は、週または月に1回程度の頻度でcronデーモンから実行するのが妥当です。オンライン照合機能を有効にして実行する方法や、これを自動化する方法については、オンラインデバイス照合の使用をご参照ください。

2.10. レプリケーション用トラフィックの整合性チェック

DRBDは、MD5、SHA-1またはCRD-32Cなどの暗号手法にもとづきノード間のメッセージの整合性チェックができます。

DRBD自身はメッセージダイジェストアルゴリズムは **備えていません** 。Linuxカーネルの暗号APIが提供する機能を単に利用するだけです。したがって、カーネルが備えるアルゴリズムであれば、どのようなものでも利用可能です。

本機能を有効にすると、レプリケート対象のすべてのデータブロックごとのメッセージダイジェストが計算されます。レプリケート先のDRBDは、レプリケーション用パケットの照合にこのメッセージダイジェストを活用します。 データの照合が失敗したら、レプリケート先のDRBDは、失敗したブロックに関するパケットの再送を求めます。 この機能を使うことで、データの損失を起こす可能性がある次のようなさまざまな状況への備えが強化され、DRBDによるレプリーションが保護されます。

- 送信側ノードのメインメモリとネットワークインタフェースの間で生じたビット単位エラー(ビット反転)。 この種のエラーは、多くのシステムにおいてTCPレベルのチェックサムでは検出できません。
- 受信側ノードのネットワークインタフェースとメインメモリの間で生じたビット反転。 TCPチェックサムが役に立たないのは前項と同じです。
- 何らかのリソース競合やネットワークインタフェースまたはそのドライバのバグなどによって生じたデータの損傷。
- ノード間のネットワークコンポーネントが再編成されるときなどに生じるビット反転やデータ損傷。 このエラーの可能性は、ノード間をネットワークケーブルで直結しなかった場合に考慮する必要があります。

レプリケーショントラフィックの整合性チェックを有効にする方法については、レプリケーショントラフィックの整合性チェックを設定をご参照ください。

2.11. スプリットブレインの通知と自動修復

クラスタノード間のすべての通信が一時的に中断され、クラスタ管理ソフトウェアまたは人為的な操作ミスによって両方のノードが プライマリ になった場合に、スプリットブレインの状態に陥ります。それぞれのノードでデータの書き換えが行われることが可能になってしまうため、この状態はきわめて危険です。つまり、2つの分岐したデータセットが作られてしまう軽視できない状況に陥る可能性が高くなります。

クラスタのスプリットブレインは、Heartbeatなどが管理するホスト間の通信がすべて途絶えたときに生じます。これとDRBDのスプリットブレインは区別して考える必要があります。このため、本書では次のような記載方法を使うことにします。

- スプリットブレイン は、DRBDのスプリットブレインと表記します。
- クラスタノード間のすべての通信の断絶のことを クラスタ・パーティション と表記します。

スプリットブレインに陥ったことを検出すると、DRBDは電子メールまたは他の方法によって管理者に自動的に通知できます。この機能を有効にする方法についてはスプリットブレインの通知をご参照ください。

スプリットブレインへの望ましい対処方法は、手動回復 を実施した後、根本原因を取り除くことです。しかし、ときにはこのプロセスを自動化する方がいい場合もあります。自動化のために、DRBDは以下のいくつかのアルゴリズムを提供します。

- **「若い」プライマリ側の変更を切り捨てる方法** ネットワークの接続が回復してスプリットブレインを検出すると、DRBDは 直近で プライマリに切り替わったノードのデータを切り捨てます。

- **「古い」プライマリ側の変更を切り捨てる方法** DRBDは 先に プライマリに切り替わったノードの変更を切り捨てます。
- **変更が少ないプライマリ側の変更を切り捨てる方法** DRBDは2つのノードでどちらが変更が少ないかを調べて、少ない方のノードの すべて を切り捨てます。
- **片ノードに変更がなかった場合の正常回復** もし片ノードにスプリットブレインの間にまったく変更がなかった場合、DRBDは正常に回復し、修復したと判断します。しかし、こういった状況はほとんど考えられません。仮にリードオンリーでファイルシステムをマウントしただけでも、デバイスへの書き換えが起きるためです。

自動修復機能を使うべきかどうかの判断は、個々のアプリケーションに強く依存します。データベースをレプリケーションしている場合を例とすると、変更量が少ないノードのデータを切り捨てるアプローチは、ある種のWebアプリケーションの場合には適しているかもしれません。一方で、金融関連のデータベースアプリケーションでは、 いかなる 変更でも自動的に切り捨てることは受け入れがたく、いかなるスプリットブレインの場合でも手動回復が望ましいでしょう。スプリットブレイン自動修復機能を使う場合、アプリケーションの特性を十分に考慮してください。

DRBDのスプリットブレイン自動修復機能を設定する方法については、スプリットブレインからの自動復旧ポリシーを参照してください。

2.12. ディスクフラッシュのサポート

ローカルディスクやRAID論理ディスクでライトキャッシュが有効な場合、キャッシュにデータが記録された時点でデバイスへの書き込みが完了したと判断されます。このモードは一般にライトバックモードと呼ばれます。このような機能がない場合の書き込みはライトスルーモードです。ライトバックモードで運用中に電源障害が起きると、最後に書き込まれたデータはディスクにコミットされず、データを紛失する可能性があります。

この影響を緩和するために、DRBDはディスクフラッシュを活用します。ディスクフラッシュは書き込みオペレーションのひとつで、対象のデータが安定した(不揮発性の)ストレージに書き込まれるまで完了しません。すなわち、キャッシュへの書き込みではなくディスクへの書き込みを保証します。DRBDは、レプリケートするデータとそのメタデータをディスクに書き込むときに、フラッシュ命令を発行します。実際には、DRBDはアクティビティログの更新時や書き込みに依存性がある場合などにはライトキャッシュへの書き込みを迂回します。このことにより、電源障害の可能性に対する信頼性が高まっています。

しかしDRBDがディスクフラッシュを活用できるのは、直下のディスクデバイスがこの機能をサポートしている場合に限られることに注意してください。最近のカーネルは、ほとんどのSCSIおよびSATAデバイスに対するフラッシュをサポートしています。LinuxソフトウェアRAID (md)は、直下のデバイスがサポートする場合に限り、RAID-1に対してフラッシュをサポートします。デバイスマッパ(LVM2、dm-raid、マルチパス)もフラッシュをサポートしています。

電池でバックアップされた書き込みキャッシュ(BBWC)は、電池からの給電による不揮発性ストレージです。このようなデバイスは、電源障害から回復したときに中断していたディスクへの書き込みをディスクにフラッシュできます。このため、キャッシュへの書き込みは、事実上安定したストレージへの書き込みと同等とみなせます。この種のデバイスが使える場合、DRBDの書き込みパフォーマンスを向上させるためにフラッシュを無効に設定するのがよいかもしれません。詳細は下位デバイスのフラッシュを無効にするをご参照ください。

2.13. Trim/Discardのサポート

Trim と Discard は、ある範囲のデータ領域が既に使用済みで不要になって [1]、他のデータ領域として再利用してよいことをストレージに伝えるコマンドで、どちらも同じ意味を持ちます。これらは、フラッシュストレージで使用されている機能です。フラッシュストレージ(SSD、FusionIOカードなど)では上書きが困難であり、通常は消去してから新しいデータを書き込む必要があります。(これにより多少のレイテンシが発生します)詳細については ここでは割愛します。

DRBDは8.4.3から trim/discard をサポートしています。設定や有効化を行う必要はありません。DRBDはローカル(下位の)ストレージシステムがそれらのコマンドをサポートしていることを検出すると、自動的に利用します。

その効果の例をあげると、大部分または全てのストレージ領域が無効になったとDRBDに伝えることで(DRBDはこれをすべての接続しているノードにリレーします）、比較的最近のmkfs.ext4であれば、初期同期時間を数TBのボリュームでも数秒から数分ほどに短縮することができます。

その後そのノードに接続する後続のリソースは Trim/Discard 要求ではなく、フル同期を行います。カーネルバージョンやファイルシステムによっては fstrim が効果を持つことがあります。

ストレージ自体が Trim/Discard をサポートしていなくても、LVMのシンプロビジョニングボリュームなどの仮想ブロックデバイスでも同様の機能を提供しています。

2.14. ディスクエラー処理ストラテジー

どちらかのノードのDRBD下位ブロックデバイスがI/Oエラーを返したときに、DRBDがそのエラーを上位レイヤ(多くの場合ファイルシステム)に伝えるかどうかを制御できます。

I/Oエラーを伝える

pass_onを設定すると、下位レベルのエラーをDRBDはそのまま上位レイヤに伝えます。したがって、そのようなエラーへの対応(ファイルシステムをリードオンリーでマウントしなおすなど)は上位レイヤに任されます。このモードはサービスの継続性を損ねることがあるので、多くの場合推奨できない設定だといえます。

I/Oエラーを伝えない.

detach を設定すると、最初の下位レイヤでのI/Oエラーに対して、DRBDは自動的にそのレイヤを切り離します。上位レイヤにI/Oエラーは伝えられず、該当ブロックのデータはネットワーク越しに対向ノードに書き込まれます。その後DRBDはディスクレスモードと呼ばれる状態になり、すべてのI/Oは対向ノードに対して読み込んだり、書き込むようになります。このモードでは、パフォーマンスは犠牲になりますが、サービスは途切れることなく継続できます。また、都合のいい任意の時点でサービスを対向ノードに移動させることができます。

I/Oエラー処理方針を設定する方法についてはI/Oエラー処理方針の設定を参照してください。

2.15. 無効データの処理ストラテジー

DRBDはデータの inconsistent(不整合状態) と outdated(無効状態) を区別します。不整合とは、いかなる方法でもアクセスできずしたがって利用できないデータ状態です。たとえば、進行中の同期先のデータが不整合データの例です。この場合、ノードのデータは部分的に古く、部分的に新しくなっており、ノード間の同期は不可能になります。下位デバイスの中にファイルシステムが入っていたら、このファイルシステムは、マウントはもちろんチェックも実行できません。

無効データは、セカンダリノード上のデータで、整合状態にあるもののプライマリ側と同期していない状態のデータをさします。一時的か永続的かを問わず、レプリケーションリンクが途切れたときに、この状態が生じます。リンクが切れている状態でのセカンダリ側の無効データは、クリーンではあるものの、対向ノードのデータ更新が反映されず古いデータ状態になっている可能性があります。サービスが無効データを使ってしまうことを防止するために、DRBDは無効データをプライマリに切り替えることを許可しません。

DRBDにはネットワークの中断時にセカンダリノードのデータを無効に設定するためのインタフェースがあります。DRBDは無効データをアプリケーションが使ってしまうことを防止するために、このノードがプライマリになることを拒絶します。本機能の完全は実装は、DRBDレプリケーションリンクから独立した通信経路を使用するPacemakerクラスタ管理フレームワーク用になされていますが、しかしこのAPIは汎用的なので、他のクラスタ管理アプリケーションでも容易に本機能を利用できます。

レプリケーションリンクが復活すると、無効に設定されたリソースの無効フラグは自動的にクリアされます。そしてバックグラウンド同期が実行されます。

誤って無効データを使ってしまうことを防止するためのDRBD/Heartbeat/Pacemakerの設定例については、DRBD無効化デーモン(dopd)を参照ください。

2.16. 3ノードレプリケーション

この機能はDRBDバージョン8.3.0以上で使用可能ですが、DRBDバージョン9.xでは単一階層で複数ノードが使用可能のため非推奨です。詳細は ネットワークコネクションの定義をご参照ください。

3ノードレプリケーションとは、2ノードクラスタに3番目のノードを追加してDRBDでレプリケーションするものです。この方法は、バックアップやディザスタリカバリのために使われます。このタイプの構成では一般的にDRBD Proxyによる遠距離レプリケーションの内容も関係します。

3ノードレプリケーション既存のDRBDリソースの上にもうひとつのDRBDリソースを スタック(積み重ね) することによって実現されます。次の図を参照してください。

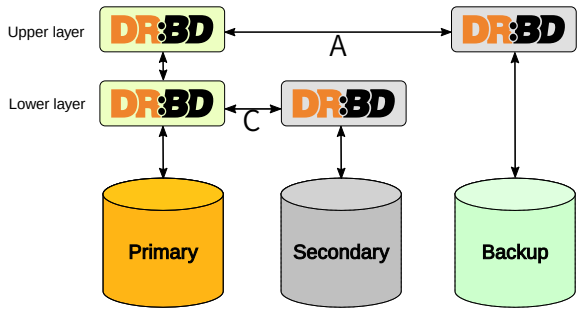


図 3. DRBDリソーススタッキング

下位リソースのレプリケーションには同期モード(DRBDプロトコルC)を使いますが、上位リソースは非同期レプリケーション(DRBDプロトコルA)で動作させます。

3ノードレプリケーションは、常時実行することも、オンデマンドで実行することもできます。常時レプリケーションでは、クラスタ側のデータが更新されると、ただちに3番目のノードにもレプリケートされます。オンデマンドレプリケーションでは、クラスタシステムとバックアップサイトの通信はふだんは停止しておき、cronなどによって定期的に夜間などに同期をはかります。

2.17. DRBD Proxyによる遠距離レプリケーション

DRBDのプロトコルAは非同期モードです。しかし、ソケットの出力バッファが一杯になると(drbd.conf マニュアルページの sndbuf-size を参照ください)、アプリケーションからの書き込みはブロックされてしまいます。帯域幅が狭いネットワークを通じて書き込みデータが対向ノードに送られるまで、そのデータを書き込んだアプリケーションは待たなければなりません。

平均的な書き込み帯域幅は、利用可能なネットワークの帯域幅によって制約されます。ソケットの出力バッファに収まるデータ量までのバースト的な書き込みは、問題なく処理されます。

オプション製品のDRBD Proxyのバッファリング機構を使って、この制約を緩和できます。DRBDプライマリノードからの書き込みデータは、DRBD Proxyのバッファに格納されます。DRBD Proxyのバッファサイズは、アドレス可能空間や搭載メモリの範囲内で自由に設定できます

データ圧縮を行うように設定することも可能です。圧縮と伸長(解凍)は、応答時間をわずかに増やしてしまいます。しかしネットワークの帯域幅が制約要因になっているのなら、転送時間の短縮効果は、圧縮と伸長(解凍)によるオーバヘッドを打ち消します。

圧縮伸長(解凍)機能は複数CPUによるSMPシステムを想定して実装され、複数CPUコアをうまく活用できます。

多くの場合、ブロックI/Oデータの圧縮率は高く、帯域幅の利用効率は向上します。このため、DRBD Proxyを使うことによって、DRBDプロトコルBまたはCを使うことも現実的なものとなります。

DRBD Proxyの設定についてはDRBD Proxyの使用を参照ください。

DRBD Proxyはオープンソースライセンスによらないプロダクトファミリの製品になります。評価や購入については linbit@sios.com へご連絡ください。

2.18. トラック輸送によるレプリケーション

トラック輸送(またはディスク輸送)によるレプリケーションは、ストレージメディアを遠隔サイトに物理的に輸送することによるレプリケーションです。以下の制約がある場合に、この方法はとくに有効です。

- 合計のレプリケート対象データ領域がかなり大きい(数百GB以上)
- 予想されるレプリケートするデータの変更レートがあまり大きくない
- 利用可能なサイト間のネットワーク帯域幅が限られている

このような状況にある場合、トラック輸送を使わなければ、きわめて長期間(数日から数週間またはそれ以上)の初期同期が必要になってしまいます。トラック輸送でデータを遠隔サイトに輸送する場合、初期同期時間を劇的に短縮できます。詳細はトラックベースのレプリケーションの使用を参照ください。

2.19. 動的対向ノード

この記述方法はDRBDバージョン8.3.2以上で使用できます。

DRBDのやや特殊な使用方法に 動的対向ノード があります。動的対向ノードを設定すると、DRBDの対向同士は(通常設定の)特定のホスト名を持つノードには接続せず、いくつかのホスト間を動的に選択して接続するする事ができます。この設定において、DRBDは対向ノードをホスト名ではなくIPアドレスで識別します。

動的対向ノードの設定については2セットのSANベースPacemakerクラスタ間をDRBDでレプリケートを参照ください。

2.20. データ再配置(ストレージの水平スケール)

例えば、会社のポリシーで3重の冗長化が要求される場合、少なくとも3台のサーバが必要になります。

しかし、データ量が増えてくると、サーバ追加の必要性に迫られます。その際には、また新たに3台のサーバを購入する必要はなく、１つのノードだけを追加をしてデータを 再配置 することができます。

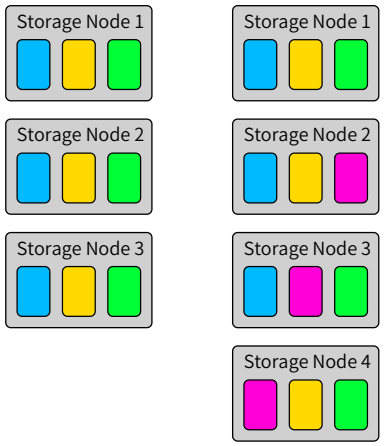


図 4. DRBDデータ再配置

上の図では、3ノードの各々に25TiBのボリュームがある合計75TiBの状態から、4ノードで合計100TiBの状態にしています。

これらはDRBD9ではオンラインで行うことができます。実際の手順についてはデータ再配置ご覧ください。

2.21. DRBDクライアント

DRBDの複数の対向ノード機能に、 DRBDクライアント などの新しいユースケースが追加されました。

基本的にDRBD バックエンド は3〜4、またはそれ以上(冗長化の要件に応じて)で構成できます。しかしDRBD9はそれ以上でも接続することができます。なお1つのビットマップslot [2]が ディスクレスプライマリ (DRBDクライアント)用に予約されます。

プライマリの DRBDクライアント で実行されるすべての書き込み要求は、ストレージを備えたすべてのノードに送られます。読み込み要求は、サーバーノードの1つにのみ送られます。 DRBDクライアント は、使用可能なすべてのサーバーノードに均等に読み読み要求を送ります。

2.22. クォーラム

スプリットブレインまたは複製データの分離を回避するためには、フェンシングを構成する必要があります。しかし、ノードのフェンシングは実際の配備であまり人気がありません。これは計画や配備でミスが発生しやすいからです。

データセットに３つの複製をもつことで、Pacemakerレベルのフェンシングに変わってDRBDのクォーラム実装を使用することができます。Pacemakerはリソースのマスタースコアを通してクォーラムまたはクォーラム喪失の通知を受け取ります。

DRBDのクォーラムはあらゆる種類のLinuxベースのサービスで使用できます。IOエラーによりサービスが終了する瞬間など、クォーラム喪失の動作はとてもよくできています。IOエラーでサービスが終了しないときは、クォーラム喪失したプライマリノードをリブートするようシステもを構成する必要があります。

詳細は クォーラム設定 を参照ください。

2.22.1. タイブレーカー

クォーラムタイブレーカー機能は、DRBDバージョン9.0.18以降で使用できます。

2つのノードクラスタの基本的な問題は、それらが接続性を失うと同時に2つのパーティションを持ち、それらのどちらもクォーラムを持たず、その結果クラスタサービスが停止することです。この問題は、クォーラムタイブレーカーとして機能する3つ目のディスクレスノードをクラスターに追加することで軽減できます。

詳細は タイブレーカーとしてのディスクレスノードを使用 を参照ください。

2.23. DRBDとVCSの統合

Veritas Cluster Server (or Veritas Infoscale Availabilty) はオープンソースであるPacemakerの代替となる商用製品です。DRBDリソースをVSCセットアップとともに使用する場合は github の drbd-utils/scripts/VCS の README を参照ください。

[1] 削除したファイルのデータなど
[2] 後に最適化によって除去される可能性があります。DRBD 9.0.0でもありえるかもしれません。

DRBDのコンパイル、インストールおよび設定

3. コンパイル済みDRBDバイナリパッケージのインストール

3.1. LINBIT社が提供するパッケージ

DRBDプロジェクトのスポンサー企業であるLINBIT社は、商用サポート対象のお客様向けにDRBDバイナリパッケージを提供しています。これらパッケージはリポジトリ (apt, yum,その他用)から利用でき、「公式」DRBDビルドです。

これらのビルドは次のディストリビューションで入手できます。

- Red Hat Enterprise Linux (RHEL)バージョン6 、7、8
- SUSE Linux Enterprise Server (SLES)バージョン11SP4、12、15
- Debian GNU/Linux 9 (stretch)、10 (buster)
- Ubuntu Server Edition LTS 14.04 (Trusty Tahr), LTS 16.04 (Xenial Xerus), LTS 18.04 (Bionic Beaver), and LTS 20.04 (Focal Fossa).

他のディストリビューションのビルドも用意していますが、十分なテストは経ていません。

LINBIT社では、新規のDRBDソースのリリースと並行してバイナリビルドをリリースしています。

RPMベースのシステム(SLES、RHEL)へのパッケージのインストールは`yum install`(新規インストールの場合)または`yum update`(アップグレードの場合)コマンドを呼び出すことで簡単に行えます。

Debianベースのシステム(Debian GNU/Linux、Ubuntu)では、 drbd-utils と drbd-module-`uname -r` パッケージを apt または、 aptitude や synaptic 等のツールを利用してインストールしてください。

3.1.1. セキュアブート/カーネルモジュールの署名

DRBD バージョン 9.0.25 以降、LINBIT はカーネルモジュールオブジェクトファイルに署名します。現在、これは RHEL 8 で使用できます。

公開署名鍵は rpm パッケージに含まれ、 /etc/pki/linbit/SECURE-BOOT-KEY-linbit.com.der にインストールされます。次の方法で登録できます。

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

パスワードは自由に選択できます。再起動後、キーが実際にMOKリストに登録されるときに使用されます。

3.2. LINBIT社が提供する Docker イメージ

LINBITは商用サポートカスタマー向けにDockerレポジトリを提供します。レポジトリはホスト名 'drbd.io' 経由でアクセスします。イメージを取得する前にレポジトリにログインする必要があります。

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

ログインに成功するとイメージを取得できます。ログインしてテストするには以下のコマンド実行してください。

```
# docker pull drbd.io/alpine
# docker run -it --rm drbd.io/alpine # press CTRL-D to exit
```

3.3. ディストリビューションベンダが提供するパッケージ

コンパイル済みバイナリパッケージを含め、いくつかのディストリビューションでDRBDが配布されています。これらのパッケージに対するサポートは、それぞれのディストリビュータが提供します。リリースサイクルは、DRBDソースのリリースより遅れる場合があります。

3.3.1. SUSE Linux Enterprise Server

SLES High Availability Extension (HAE) includes DRBD.

SLESの場合、DRBDは通常はYaST2のソフトウェアインストールコンポーネントによりインストールされます。これは High Availabilityパッケージセレクションに同梱されています。

コマンドラインを使用してインストールする場合は、次のコマンドを実行します。

```
# yast -i drbd
```

または

```
# zypper install drbd
```

3.3.2. CentOS

CentOSのリリース5からDRBD 8が含まれています。DRBD 9はEPEL等から探してください。

DRBDは yum でインストールします。この際には、正しいリポジトリが有効である必要があります。

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

3.3.3. Ubuntu Linux

LINBITはUbuntu LTS用にPPAリポジトリを提供しています。 https://launchpad.net/~linbit/`archive/ubuntu/linbit-drbd9-stack. 詳細は以下をご確認ください。 Adding Launchpad PPA Repositories

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

3.4. ソースからパッケージをコンパイルする

github で git tags によって生成されたリリースはある時点での git レポジトリのスナップショットです。これらはマニュアルページや configure スクリプト、あるいはその他の生成されるファイルが不足しているかもしれません。tarball からビルドするなら、 DRBD Community Download を使用してください。

すべてのプロジェクトは標準のビルドスクリプト (eg, Makefile, configure) を含みます。ディストリビューション毎に固有の情報をメンテナンスすることは手間がかかり、また歴史的にすぐに最新でなくなってしまいまし

た。標準的な方法でソフトウェアをビルドする方法を知らない場合は、LINBITによって供給されるパッケージを使ってください。

DRBDの使い方

4. 一般的な管理作業

この章では一般的なオペレーションでの管理作業を説明します。トラブルシューティングについては扱いません。トラブルシューティングについてはトラブルシューティングとエラーからの回復を参照ください。

4.1. DRBDの設定

4.1.1. 下位レベルストレージの準備

DRBDをインストールしたら、両方のクラスタノードにほぼ同じ容量の記憶領域を用意する必要があります。これがDRBDリソースの 下位レベルデバイス になります。システムの任意のブロックデバイスを下位レベルデバイスとして使用できます。たとえば、次のようなものがあります。

- ハードドライブのパーティション(または物理ハードドライブ全体)
- ソフトウェアRAIDデバイス
- LVM論理ボリュームまたはLinuxデバイスマッパインフラストラクチャによって構成されるその他のブロックデバイス
- システム内のその他のブロックデバイス

リソースを スタッキング(積み重ね) することもできます。つまり、DRBDデバイスを他のDRBDデバイスの下位レベルのデバイスとして利用することができます。リソースの積み重ねにはいくつかの注意点があります。詳しくはスタック3ノード構成の作成を参照ください。

ループデバイスをDRBDの下位レベルデバイスとして使用することもできますが、デッドロックの問題があるためお勧めできません。

DRBDリソースを作成する前に、そのストレージ領域を空にしておく 必要はありません 。DRBDを使用して、非冗長のシングルサーバシステムから、2ノードのクラスタシステムを作成することは一般的なユースケースですが、いくつか重要な注意点があります。(その場合にはDRBDメタデータを参照ください)

本ガイドの説明は、次のようなとてもシンプルな構成を前提としています。

- 両ホストには使用可能な(現在未使用の) /dev/sda7 というパーティションがある。
- 内部メタデータを使用する。

4.1.2. ネットワーク構成の準備

必須要件ではありませんが、DRBDによるレプリケーションの実行には、専用接続を使用することをお勧めします。この書き込みには、ギガビットイーサネット同士をケーブルで直結した接続が最適です。DRBDをスイッチを介して使用する場合には、冗長コンポーネントと bonding ドライバ(active-backup モードで)の使用を推奨します。

一般に、ルータを介してDRBDレプリケーションを行うことはお勧めできません。スループットと待ち時間の両方に悪影響を及ぼし、パフォーマンスが大幅に低下します。

ローカルファイアウォールの要件として重要な点は、通常、DRBDは7788以上のTCPポートを使用し、それぞれのTCPリソースが個別のTCPポート上で待機するということです。DRBDは 2つ のTCP接続を使用します。これらの接続が許可されるようにファイアウォールを設定する必要があります。

SELinuxやAppArmorなどのMAC (Mandatory Access Control)スキーマが有効な場合は、ファイアウォール以外のセキュリティ要件も考慮する場合があります。DRBDが正しく機能するように、 必要に応じてローカルセキュリティポリシーを調整してください。

また、DRBDに使用するTCPポートを別のアプリケーションが使用していないことも確認してください。

現時点では1つのDRBDリソースに複数のTCPコネクション・ペアを設定することはできません。DRBD接続に負荷分散や冗長性が必要な場合は、イーサネットレベルで簡単に実現できます(この場合も bonding ドライバを使用してください)。

本ガイドの説明は、次のようなとてもシンプルな構成を前提としています。

- 2つのDRBDホストそれぞれに、現在使用されていないネットワークインタフェース eth1 が存在する(IPアドレスはそれぞれ 10.1.1.31 と 10.1.1.32)。
- どちらのホストでも他のサービスがTCPポート7788～7799を使用していない。
- ローカルファイアウォール設定は、これらのポートを介したホスト間のインバウンドとアウトバウンドの両方のTCP接続を許可する。

4.1.3. リソースの設定

DRBDのすべての機能は、設定ファイル /etc/drbd.conf で制御されます。通常、この設定ファイルは、次のような内容となっています。

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

通例では、/etc/drbd.d/global_common.conf にはDRBD設定の global と common セクションが含まれます。また、 .res ファイルには各 リソース セクションが含まれます。

drbd.conf に include ステートメントを使用せずにすべての設定を記載することも可能です。しかし、設定の見やすさの観点から、複数のファイルに分割することをお勧めします。

いずれにしても drbd.conf や、その他の設定ファイルは、すべてのクラスタノードで 正確に同じ である必要があります。

DRBDのソースtarファイルの scripts サブディレクトリに、サンプル設定ファイルがあります。バイナリインストールパッケージの場合、サンプル設定ファイルは直接 /etc にインストールされるか、/usr/share/doc/packages/drbd などのパッケージ固有の文書ディレクトリにインストールされます。

このセクションは、DRBDを稼働させるために理解しておく必要のある設定ファイルの項目についての説明です。設定ファイルの構文と内容の詳細については drbd.conf マニュアルページを参照ください。

設定例

本ガイドでの説明は、前章であげた例をもとにする最小限の構成を前提にしています。

Listing 1. シンプルなDRBD構成例 (/etc/drbd.d/global_common.conf)

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

Listing 2. シンプルなDRBDリソースの構成例 (/etc/drbd.d/r0.res)

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

この例では、DRBDが次のように設定されます。

- DRBDの使用状況の統計をオプトインとして含める(usage-count参照)。
- 特に他の指定がない限り完全に同期したレプリケーションを使用するようにリソースを設定する(プロトコルC)。
- クラスタには2つのノード alice と bob がある。
- r0 という名前のリソース(名前は自由に設定可能)があり /dev/sda7 を下位レベルデバイスとして使用し、また、内部メタデータを構成する。
- リソースはネットワーク接続にTCPポート7789を使用し、それぞれIPアドレス10.1.1.31と10.1.1.32にバインドされる(これが暗黙的に使用するネットワークコネクションを定義する)。

暗黙的に、上記の設定はリソースの1つのボリュームを作成し、番号 ゼロ(0) が付与されます。1つのリソースに複数のボリュームを設定する場合には、次のようにします(両ノードで下位デバイスとして同レベルでストレージブロックデバイスを使用する場合)。

Listing 3. 複数ボリュームのDRBDリソース構成例(/etc/drbd.d/r0.res)

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

ボリュームは既存のデバイスの動作中にも追加できます。新しいDRBDボリュームを既存のボリュームグループへ追加するをご参照ください。

global セクション

このセクションは設定の中で1回しか使用できません。通常この設定は /etc/drbd.d/global_common.conf ファイルに記述します。設定ファイルが1つの場合は、設定ファイルの一番上に記述します。このセクションで使用できるオプションはわずかですが、ほとんどのユーザーの場合、必要なのは次の1つだけです。

usage-count

DRBDプロジェクトはさまざまなバージョンのDRBDの使用状況について統計を取ります。これは、システムに新規のDRBDバージョンがインストールされるたびに、HTTPサーバに接続することにより実行されます。これを無効にするには、 usage-count no; を指定します。デフォルトは usage-count ask; で、 DRBDをアップグレードするたびにプロンプトが表示されます。

DRBDの使用状況の統計は公開されています。http://usage.drbd.orgを参照ください。

common セクション

このセクションで、各リソースに継承される設定を簡単に定義できます。通常この設定は /etc/drbd.d/global_common.conf に指定します。ここで定義するオプションは、リソースごとに定義することもできます。

common セクションは必須ではありませんが、複数のリソースを使用する場合は、記述することを強くお勧めします。これにより、オプションを繰り返し使用することによって設定が複雑になることを回避できます。

上の例では net { protocol C; } が common セクションで指定されているため、設定されているすべてのリソース(r0 含む)がこのオプションを継承します。ただし、明示的に別の protocol オプションが指定されている場合は除きます。使用可能なその他の同期プロトコルについては、レプリケーションのモードを参照してください。

resource セクション

各リソースの設定ファイルは、通常 /etc/drbd.d/resource.res という名前にします。定義するDRBDリソースは、設定ファイルでresource nameを指定して名前を付ける必要があります。通常は文字または数字、アンダースコアのみを使用します。

各リソースには各クラスタノードに最低2つの on <host> サブセクションも必要です。その他すべての設定は common セクション(記述した場合)から継承されるか、DRBDのデフォルト設定から取得されます。

さらに、オプションの値が両方のホストで等しい場合は、直接 resource セクションで指定することができます。このため、設定例は次のように短くすることができます。

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

4.1.4. ネットワークコネクションの定義

現時点では、DRBD9の通信リンクはフルメッシュである必要があります。つまり、全リソース全ノードが他の全ノードに直接のコネクションを持っている必要があります(当然、自ノードに対しては不要です)。

ホスト2台のシンプルな構成の場合、使い勝手と後方互換性のため、drbdadm は(1つの)ネットワークコネクションを自身で挿入します。

必要なネットワークコネクション数はホスト数の二次関数です。"従来の"2ノードでは1コネクションが必要でしたが、3つのホストでは3対、4つのホストでは6対、5つのホストでは10対のコネクションが…というように必要になってきます。32ノードであれば496対のコネクションが必要になります。

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

図 5. N 個のホストの時のコネクション数

以下は3つのホストでの設定ファイルの例です。

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

サーバに十分なネットワークカードがあれば、サーバ間をクロスケーブルで直結できます。 1つ4ポートのイーサネットカードであれば、4ノードのフルメッシュにするために1つの管理インターフェースと3つの他サーバへの接続を行うことができます。

この場合には直接接続に異なるIPアドレスを指定することができます。

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

管理とデバッグを容易にするため、エンドポイントごとに異なるポートを使用することをお勧めします。tcpdump 使用の際にパケットの追跡が容易になります。

以下の例は2サーバのみで使用するものです。4ノードでの例は4ノードでの構成例を参照ください。

4.1.5. トランスポートプロトコルの設定

DRBDは複数のネットワーク転送プロトコルに対応しています。 トランスポートプロトコルの設定はリソースの各コネクションごとに設定できます。

TCP/IP

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

デフォルトは tcp です。トランスポートオプションを設定していない場合は TCP トランスポートが使用されます。

tcp トランスポートオプションでは次のnetオプションが設定できます。 sndbuf-size 、 rcvbuf-size 、 connect-int 、 `sock-check-timeo` 、 ping-timeo 、 timeout 。

RDMA

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

rdma トランスポートでは次のnetオプションが設定できます。 sndbuf-size 、 rcvbuf-size 、 max_buffers 、

connect-int 、 sock-check-timeo 、 ping-timeo 、 timeout 。

rdma トランスポートはゼロコピーレシーブ転送です。その関係で、 max_buffers の設定オプションはすべての rcvbuf-size を保持するのに十分なサイズにする必要があります。

rcvbuf-size はバイト単位で設定しますが、 max_buffers はページ単位で設定します。パフォーマンス最適化のためには、 max_buffers はすべての rcvbuf-size と、下位デバイスとの間の全通信量を合わせたものよりも大きい必要があります。

InfiniBand HCAで rdma トランスポートを使っている場合、IPoIBも設定する必要があります。IPアドレスはデータ転送では使用しませんが、コネクションを確立する際に正しいアダプタとポートを見つけるために使用します。

設定オプションの sndbuf-size と rcvbuf-size はコネクションが確立される時に考慮されます。つまり、接続が確立している時は変更する事ができますが、その変更が反映されるのは、再接続後になります。

RDMAのパフォーマンスにおける考慮事項

擬似ファイル/sys/kernel/debug/drbd/<リソース>/connections/<対向ノード>/transportを見れば、使用可能な受信識別子(rx_desc)と送信識別子(tx_desc)の数を監視できます。識別子が枯渇した場合には sndbuf-size または rcvbuf-size を増やす必要があります。

4.1.6. リソースを初めて有効にする

すでに述べた手順に従って最初のリソース設定を完了したら、リソースを稼働させます。

両方のノードに対して、次の手順を行います。

さきほどの構成例(resource r0{ … })では、 <resource> は r0 となります。

メタデータを作成する

この手順は、最初にデバイスを作成するときにのみ必要です。これにより、DRBDのメタデータを初期化します。

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

メタデータに割り当てられるビットマップスロットの数はリソースのホストの数に依存します。 デフォルトではリソース設定のホストの数をカウントします。 メタデータの作成前にすべてのホストが指定されていれば、そのまま動作します。後から追加ノード用のビットマップを付け足すことも可能ですが、手動での作業が必要になります。

リソースを有効にする

これにより、リソースとその下位デバイス(マルチボリュームリソースの場合は、すべてのデバイス)とを結びつけます。また、対向ノードのリソースと接続します。

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

drbdadm status でステータスを確認する

drbdsetup のステータス出力は次のような情報を表示します。

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

この時点では Inconsistent/Inconsistent のディスク状態になっているはずです。

これで、DRBDがディスクリソースとネットワークリソースに正しく割り当てられ、稼働できるようになりました。次に、どちらのノードをデバイスの初期同期のソースとして使用するか指定する必要があります。

4.1.7. デバイスの初期同期

DRBDを完全に機能させるには、さらに次の2つの手順が必要です。

同期元を選択する

新しく初期化した空のディスクを使用する場合は、任意のディスクを同期元にできます。いずれかのノードにすでに重要なデータが格納されている場合は、 十分注意して、必ず そのノードを同期元として選択してください。デバイスの初期同期の方向が誤っていると、データを失うおそれがあります。慎重に行ってください。

初期フル同期を開始する

この手順は、最初のリソース設定の際に、同期ソースとして選択した1つのノードに対してのみ実行します。次のコマンドで実行します。

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

このコマンドを指定すると、初期フル同期が開始します。 drbdadm status で同期の進行状況を監視できます。デバイスのサイズによっては、同期に時間がかかる場合があります。

この時点で、初期同期が完了していなくてもDRBDデバイスは完全に稼働します(ただし、パフォーマンスは多少低いです)。空のディスクから開始した場合は、デバイスにファイルシステムを作成してもかまいません。これを下位ブロックデバイスとして使用し、マウントして、アクセス可能なブロックデバイスとしてさまざまな操作を実行することができます。

リソースに対して一般的な管理タスクを行う場合は、DRBDの使い方に進んでください。

4.1.8. トラックベースのレプリケーションの使用

リモートノードに同期するデータを前もってロードし、デバイスの初期同期をスキップする場合は、次の手順を行います。

初期設定済みで プライマリ に昇格し、相手ノードとの接続を切断した状態のDRBDリソースが必要です。つまり、デバイスの設定が完了し、両方のノードに同一の drbd.conf のコピーが存在し 最初のリソース昇格をローカルノードで実行するコマンドを発行した後、 — リモートノードがまだ接続されていない状態です。

- ローカルノードで次のコマンドを実行します。

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

または

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

- リソースのデータ およびそのメタデータ の正確に同一のコピーを作成します。 たとえば、ホットスワップ可能な RAID-1ドライブの一方を抜き取ります。この場合は、もちろん 新しいドライブをセットしてRAIDセットを再構築しておくべきでしょう。 抜き取ったドライブは、正確なコピーとして リモートサイトに移動できます。別の方法としては、ローカルのブロックデバイスがスナップショットコピーをサポートする場合 (LVMの上位でDRBDを使用する場合など)は、 dd を使用してスナップショットのビット単位のコピーを作ってもかまいません。
- ローカルノードで次のコマンドを実行します。

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

または drbdsetup コマンドを使用します。

この2回目のコマンドには --clear-bitmap がありません。

- 対向ホストの設置場所にコピーを物理的に移動します。
- コピーをリモートノードに追加します。ここでも物理ディスクを接続するか、リモートノードの既存のストレージに移動したデータのビット単位のコピーを追加します。レプリケートしたデータだけでなく、関連するDRBDメタデータも必ず復元するかコピーしてください。そうでない場合、ディスクの移動を正しく行うことができません。
- 新しいノードでメタデータのノードIDを修正し、2つのノード間で対抗ノードの情報を交換する必要があります。リソース r0 ボリューム 0 上でnode idを2から1に変更するには、次の行を参照ください。

これはボリュームが未使用中のときに実行する必要があります。

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

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

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

NOTE

バージョン 8.9.7 以前の drbdmeta 順不同の対抗ノードセクションを扱えません。エディタを用いてブロックを交換する必要があります。

- リモートノードで次のコマンドを実行します。

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

2つのホストを接続しても、デバイスのフル同期は開始されません。代わりに、 drbdadm --clear-bitmap new-current-uuid の呼び出し以降に変更されたブロックのみを対象とする自動同期が開始されます。

以降、データの変更が 全くない 場合でも、セカンダリでアクティビティログを含む領域がロールバックされるため、同期が短時間行われることがあります。これはチェックサムベースの同期を使用することで緩和されます。

この手順は、リソースが通常のDRBDリソースの場合でもスタックリソースの場合でも使用できます。スタックリソースの場合は、 -S または --stacked オプションを drbdadm に追加します。

4.1.9. 4ノードでの構成例

以下は4ノードクラスタの例です。

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

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

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

connection-mesh 設定を確認したい場合には drbdadm dump <resource> -v を実行します。

別の例として、4ノードに直接接続でフルメッシュにできるだけのインターフェースがある場合には、 [3]インターフェースにIPアドレスを指定することができます。

```
resource r0 {
 ...

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

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

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

IPアドレスやポート番号の付け方には注意してください。別のリソースであれば同じIPアドレスを使用することができますが、 71xy、72xy のようにずらしてください。

4.2. DRBDのステータスを確認する

4.2.1. drbdmon でステータスを取得する

DRBDの状態を見るのに便利な方法の1つは drbdmon ユーティリティを使うことです。これはリアルタイムでDRBDリソースの状態を更新します。

4.2.2. drbdtop でDRBDのステータス取得し対話する

その名前が示すように drbdtop は htop のようなツールと似ています。一方では、DRBDリソースの監視と対話（例えば Primary への切り替え、あるいはスプリット・ブレーンの解決など）を行うことができます。詳細は https://linbit.github.io/drbdtop/ を参照ください。

4.2.3. /proc/drbd でのステータス情報

''/proc/drbd'' は非推奨です。8.4系でこの機能を取り除く予定はありませんが、他の方法を使用することをおすすめします。drbdadm でのステータス情報や、より便利なモニタリングとしてdrbdsetup events2 を使用した一回のみ、またはリアルタイムでの監視など

/proc/drbd はDRBDモジュールの基本情報を表示する仮想ファイルです。 DRBD8.4まで広く使用されていましたが、DRBD9の情報量を表示するためには対応できません。

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

1行目にはシステムで使用するDRBDの バージョン を表示します。2行目にはビルド特有の情報を表示します。

4.2.4. drbdadm でのステータス情報

一番シンプルなものとして、1つのリソースのステータスを表示します。

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

ここではリソース home がローカルと nina と nino にあり、 UpToDate で セカンダリ であることを示しています。つまり、3ノードが同じデータをストレージデバイスに持ち、現在はどのノードでもデバイスを使用していないという意味です。

ノード nono は接続していません。 Connecting のステータスになっています。詳細はコネクションステータスを参照してください。

drbdsetup に --verbose および/または --statistics の引数を付けると、より詳細な情報を得ることができます。

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

この例では、ローカルノードについては多少異なりますが、このリソースで使用しているノードすべてを数行ごとにブロックで表示しています。以下で詳細を説明します。

各ブロックの最初の行は node-id です。(現在のリソースに対してのものです。ホストは異なるリソースには異なる node-id がつきます) また、 role (リソースのロール参照)も表示されています。

次に重要な行が、 volume の表示がある行です。通常は0から始まる数字ですが、設定によっては別のIDをつけることもできます。この行では replication で コネクションステータス(コネクションステータス参照)を、 disk でリモートのディスク状態 (ディスク状態参照)が表示されます。 また、ボリュームの統計情報が少し表示される行もあります。データの received 、 sent , out-of-sync などです。詳細は パフォーマンス指標と接続情報データを参照してください。

ローカルノードでは、最初の行はリソース名を表示します。この例では home です。最初の行には常にローカルノードが表示されますので、 Connection やアドレス情報は表示されません。

より詳細な情報については、 drbd.conf マニュアルページをご覧ください。

この例のブロックになっている他の4行は、すべての設定のあるDRBDデバイスごとになっており、最初にデバイスマイナー番号がついています。この場合にはデバイス /dev/drbd0 に対応して 0 です。

リソースごとの出力には様々なリソースに関する情報が含まれています。

4.2.5. drbdsetup events2 を使用した一回のみ、またはリアルタイムでの監視

この機能はユーザスペースのDRBDが8.9.3より後のバージョンでのみ使用できます

これはDRBDから情報を取得するための下位機能です。監視など自動化ツールでの使用に向いています。

一番シンプルな使用方法では、以下のように現在のステータスのみを表示します(端末上で実行した場合には色も付いています)。

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

''--now'' を付けないで実行した場合には動作し続け、以下のように更新を続けます。

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

そして監視用途に、''--statistics''という別の引数もあります。これはパフォーマンスその他のカウンタを作成するものです。

''drbdsetup'' の詳細な出力(読みやすいように一部の行は改行しています)

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

''--timestamp'' パラメータも便利な機能です。

4.2.6. コネクションステータス

リソースのコネクションステータスは drbdadm cstate コマンドで確認することができます。

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

確認したいのが１つのコネクションステータスだけの場合にはコネクション名を指定してください。

デフォルトでは設定ファイルに記載のある対向ノードのホスト名です。

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

リソースのコネクションステータスには次のようなものがあります。

StandAlone
ネットワーク構成は使用できません。リソースがまだ接続されていない、管理上の理由で切断している(drbdadm disconnect を使用)、認証の失敗またはスプリットブレインにより接続が解除された、のいずれかが考えられます。

Disconnecting
切断中の一時的な状態です。次の状態は StandAlone です。

Unconnected
接続を試行する前の一時的な状態です。次に考えられる状態は、 Connecting です。

Timeout
対向ノードとの通信のタイムアウト後の一時的な状態です。次の状態は Unconnected です。

BrokenPipe
対向ノードとの接続が失われた後の一時的な状態です。次の状態は Unconnected です。

NetworkFailure
対向ノードとの接続が失われた後の一時的な状態です。次の状態は Unconnected です。

ProtocolError
対向ノードとの接続が失われた後の一時的な状態です。次の状態は Unconnected です。

TearDown
一時的な状態です。対向ノードが接続を閉じています。次の状態は Unconnected です。

Connecting
対向ノードがネットワーク上で可視になるまでノードが待機します。

Connected
DRBDの接続が確立され、データミラー化がアクティブになっています。これが正常な状態です。

4.2.7. 複製ステータス

各ボリュームは接続を介した複製ステータスを持ちます。可能な複製ステータスは以下になります。

Off

ボリュームはこの接続を通して複製されていません。接続が Connected になっていません。す。次の状態は Unconnected です。

Established
このボリュームへのすべての書き込みがオンラインで複製されています。これは通常の状態です。

StartingSyncS
管理者により開始されたフル同期が始まっています。次に考えられる状態は SyncSource または PausedSyncS です。

StartingSyncT
管理者により開始されたフル同期が始まっています。次の状態は WFSyncUUID です。

WFBitMapS
部分同期が始まっています。次に考えられる状態は SyncSource または PausedSyncS です。

WFBitMapT
部分同期が始まっています。次に考えられる状態は WFSyncUUID です。

WFSyncUUID
同期が開始されるところです。次に考えられる状態は SyncTarget または PausedSyncT です。

SyncSource
現在、ローカルノードを同期元にして同期を実行中です。

SyncTarget
現在、ローカルノードを同期先にして同期を実行中です。

PausedSyncS
ローカルノードが進行中の同期の同期元ですが、現在は同期が一時停止しています。原因として、別の同期プロセスの完了との依存関係、または drbdadm pause-sync を使用して手動で同期が中断されたことが考えられます。

PausedSyncT
ローカルノードが進行中の同期の同期先ですが、現在は同期が一時停止しています。原因として、別の同期プロセスの完了との依存関係、または drbdadm pause-sync を使用して手動で同期が中断されたことが考えられます。

VerifyS
ローカルノードを照合元にして、オンラインデバイスの照合を実行中です。

VerifyT
現在、ローカルノードを照合先にして、オンラインデバイスの照合を実行中です。

Ahead
リンクが負荷に対応できないので、データの複製が中断しました。このステータスは on-congestion オプションの設定で有効にできます(輻輳ポリシーと中断したレプリケーションの構成を参照)。

Behind
リンクが負荷に対応できないので、データの複製が対抗ノードによって中断されました。このステータスは対抗ノードの on-congestion オプション設定で有効にできま>す(輻輳ポリシーと中断したレプリケーションの構成を参照)。

4.2.8. リソースのロール

リソースのロールは drbdadm role コマンドを実行することで確認できます。

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

以下のいずれかのリソースのロールが表示されます。

Primary
リソースは現在プライマリロールで読み書き加能です。2つのノードの一方だけがこのロールになることができます。ただし、デュアルプライマリモードの場合は例外です。

Secondary
リソースは現在セカンダリロールです。対向ノードから正常に更新を受け取ることができますが(切断モード以外の場合)、このリソースに対して読み書きは実行できません。1つまたは両ノードがこのロールになることができます。

Unknown
リソースのロールが現在不明です。ローカルリソースロールがこの状態になることはありません。切断モードの場合に、対向ノードのリソースロールにのみ表示されます。

4.2.9. ディスク状態

リソースのディスク状態は drbdadm dstate コマンドを実行することで確認できます。

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

ディスク状態は以下のいずれかです。

Diskless
DRBDドライバにローカルブロックデバイスが割り当てられていません。原因として、リソースが下位デバイスに接続されなかった、 drbdadm detach を使用して手動でリソースを切り離した、または下位レベルのI/Oエラーにより自動的に切り離されたことが考えられます。

Attaching
メタデータ読み取り中の一時的な状態です。

Detaching
切断され、進行中のIO処理が完了するのを待っている一時的な状態。

Failed
ローカルブロックデバイスがI/O障害を報告した後の一時的な状態です。次の状態は Diskless です。

Negotiating
すでに Connected のDRBDデバイスで attach が実行された場合の一時的状態です。

Inconsistent
データが一致しません。新規リソースを作成した直後に(初期フル同期の前に)両方のノードがこの状態になります。また、同期中には片方のノード(同期先)がこの状態になります。

Outdated
リソースデータは一致していますが、無効です。

DUnknown
ネットワーク接続を使用できない場合に、対向ノードディスクにこの状態が使用されます。

Consistent
接続していない状態でノードのデータが一致しています。接続が確立すると、データが UpToDate か Outdated か判断されます。

UpToDate
データが一致していて最新の状態です。これが正常な状態です。

4.2.10. 接続情報データ

local
ネットワークファミリ、ローカルアドレス、対向ノードから接続を許可されたポートを表示します。

peer
ネットワークファミリ、ローカルアドレス、接続に使用しているポートを表示します。

congested
データのTCP送信バッファを80%より多く使用している場合にこのフラグがつきます。

4.2.11. パフォーマンス指標

drbdadm status の出力には、以下のカウンタと指標があります。

send (network send)
ネットワークを通じて相手に送信された正味データ量(kibyte単位)。

receive (network receive)
ネットワークを通じて相手から受信した正味データ量(kibyte単位)。

read (disk write)
ローカルのハードディスクに書き込んだ正味データ量(kibyte単位)。

written (disk read)
ローカルのハードディスクから読み込んだ正味データ量(kibyte単位)。

al-writes (activity log)
メタデータのアクティビティログエリアのアップデート数。

bm-writes (bit map)
メタデータのビットマップエリアのアップデート数。

lower-pending (local count)
DRBDが発行したローカルI/Oサブシステムへのオープンリクエストの数。

pending
相手側に送信したが相手から応答がないリクエスト数。

unacked (unacknowledged)

ネットワーク接続を通じて相手側が受信したが応答がまだないリクエスト数。

upper-pending (application pending)
まだDRBDから応答がないDRBDへのブロックI/Oリクエスト数。

write-ordering (write order)
現在使用中の書き込み順序制御方法： b(barrier)、 f(flush)、 d(drain)、 n(none)

out-of-sync
現在同期していない正味ストレージ容量(kibyte単位)

resync-suspended
現在再同期が中断されているかどうか。値は no 、 user 、 peer 、 dependency のいずれかです。

blocked
ローカルI/Oの輻輳を示します。

- no:輻輳なし
- uppeer: ファイルシステムなどDRBD より上位 のI/Oがブロックされている。代表的なものには以下がある。
  - 管理者によるI/O中断。 drbdadm コマンドの suspend-io を参照。
  - アタッチ/デタッチ時の一時的なブロック。
  - バッファーの枯渇。DRBDパフォーマンスの最適化 参照。
  - ビットマップIO待ち
- lower: 下位デバイスの輻輳

upper、lower の値を見ることも可能です。

4.3. リソースの有効化と無効化

4.3.1. リソースの有効化

通常、自動的にすべての設定済みDRBDリソースが有効になります。これは、

- クラスタ構成に応じたクラスタ管理アプリケーションの操作によるものであり、
- またはシステム起動時の /etc/init.d/drbd によっても有効になります。

手動でリソースを起動する必要がある場合には、以下のコマンドの実行によって行うことができます。

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

他の場合と同様に、特定のリソース名の代わりにキーワード all を使用すれば、 /etc/drbd.conf で設定しているすべてのリソースを一度に有効にできます。

4.3.2. リソースを無効にする

特定のリソースを一時的に無効にするには、次のコマンドを実行します。

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

ここでも、リソース名の代わりにキーワード all を使用して、1回で /etc/drbd.conf に記述しているすべてのリソースを一時的に無効にできます。

4.4. リソースの再設定

DRBDの動作中にリソースを再設定することができます。次の手順を行います。

- /etc/drbd.conf のリソース設定を変更します。
- 両方のノードで /etc/drbd.conf ファイルを同期します。
- drbdadm adjust <resource> コマンドを 両ノードで実行します。

drbdadm adjust は drbdsetup を通じて実行中のリソースを調整します。保留中の drbdsetup 呼び出しを確認するには、 drbdadm を -d (dry-run,予行演習)オプションを付けて実行します。

/etc/drbd.conf の common セクションを変更して一度にすべてのリソースに反映させたいときには drbdadm adjust all を実行します。

4.5. リソースの昇格と降格

手動でリソースロールをセカンダリからプライマリに切り替える(昇格)、またはその逆に切り替える(降格)には、次のコマンドを実行します。

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

DRBDがシングルプライマリモード(DRBDのデフォルト)で、コネクションステータスが Connected の場合、任意のタイミングでどちらか1つのノード上でのみリソースはプライマリロールになれます。したがって、あるリソースが他のノードに対してプライマリロールになっているときに drbdadm primary <resource> を実行すると、エラーが発生します。

リソースがデュアルプライマリモードに対応するよう設定している場合には、両方のノードをプライマリロールに切り替えることができます。これは、例えば仮想マシンのオンラインマイグレーションの際に利用できます。

4.6. 基本的な手動フェイルオーバ

Pacemakerを使わず、パッシブ/アクティブ構成でフェイルオーバを手動で制御するには次のようにします。

現在のプライマリノードでDRBDデバイスを使用するすべてのアプリケーションとサービスを停止し、リソースをセカンダリに降格します。

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

プライマリにしたいノードでリソースを昇格してデバイスをマウントします。

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

自動プロモート 機能を使用している場合はロール(プライマリ/セカンダリ)を手動で変更する必要はありません。それぞれサービス停止とアンマウント、マウントの操作のみ必要です。

4.7. DRBDのアップグレード

DRBDのアップグレードは非常にシンプルな手順です。この章ではDRBD8.4から9.0へのアップグレード手順を説明します。DRBD9系内でのアップグレードの手順についてはより簡単ですので以下の項目を参照ください。

4.7.1. 概要

8.4から9.0へのアップグレード手順の概要は以下の通りです。

- 新しいリポジトリを設定する。(LINBITのパッケージを使っている場合)
- 現在の状態が問題ない事を確認する。
- クラスタ管理ソフトを停止する。
- 新しいバージョンをインストールする。
- 2ノード以上を移動させたい場合は、追加メタデータ用のスペースを加えるために下位レベルデバイスをリサイズする必要がある。この点についてはLVMの章参照。
- リソースの設定を取り除き、DRBD8.4をアンロードし、そしてv9のカーネルモジュールをロードする。
- v09 の形式にDRBDメタデータをコンバートする。ビットマップ数も同様の手順。
- リソースを起動をして完了。

4.7.2. リポジトリのアップデート

8.4と9.0のブランチ間の様々な変更のため、各々に別のリポジトリを作成しています。各サーバでリポジトリのアップデートを行います。

RHEL/CentOSのシステム

/etc/yum.repos.d/linbit.repo ファイルに以下の変更を反映します。

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

<hash>と<arch>の箇所は置き換えてください。<hash>はLINBITサポートから提供されるものです。

Debian/Ubuntuのシステム

/etc/apt/sources.list(または /etc/apt/sources.d/ にあるファイル)に以下の変更を反映します。

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

wheezy リリースを使用していない場合でも、また他のリリースの場合でも、この変更は必要です。

<hash>の箇所は置き換えてください。<hash>はLINBITサポートから提供されるものです。

次にDRBDの署名キーを信頼済みキーに追加してもよいでしょう。

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

最後に apt update を実行してリポジトリのアップデートを認識させます。

```
# apt update
```

4.7.3. DRBDの状態を確認する

リソースが同期していることを確認します。cat /proc/drbd が UpToDate/UpToDate を出力しています。(これは、DRBD8系 以下 でのみ有効です)

```
bob# cat /proc/drbd

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

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

4.7.4. クラスタを停止する

リソースが同期している事を確認したら、セカンダリノードをアップグレードします。この手順は手動で行うことができます。またはPacemakerを使用している場合にはノードをスタンバイモードにしておきます。両手順は以下の通りです。Pacemakerの動作中は手動では操作しないでください。

- 手動手順

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

- Pacemaker

セカンダリノードをスタンバイモードにします。この例では bob はセカンダリです。

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

クラスタの状態を 'crm_mon -rf' で確認することができます。または、リソースの状態を、もし "Unconfigured" と表示されていなければ 'cat /proc/drbd' で確認することができます。

4.7.5. パッケージのアップグレード

パッケージをyumまたはaptでアップデートします。

```
bob# yum upgrade
```

```
bob# apt upgrade
```

アップグレードが完了すると、最新のDRBD9のカーネルモジュールとdrbd-utilsがセカンダリノードの bob にインストールされています。

しかしカーネルモジュールはまだ有効化されていません。

4.7.6. 新しいカーネルモジュールのロード

現時点でDRBDモジュールは使用していませんので、以下コマンドでアンロードします。

```
bob# rmmod drbd
```

" ERROR : Module drbd is in use " のようなメッセージが出る場合は、まだすべてのリソースが正常に停止していません。DRBDのアップグレードを再実行し、そして/またはどのリソースがまだ稼働しているのか確認するため drbdadm down all を実行します。

よくあるアンロードを妨げる原因には以下のものがあります。

- DRBDが下位デバイスのファイルシステムにNFSエクスポートがある(exportfs -v の出力を確認)
- ファイルシステムをマウント中 - grep drbd /proc/mounts を確認
- ループバックデバイスがアクティブ (losetup -l)
- デバイスマッパーが直接または間接的にDRBDを使用している。(dmsetup ls --tree)
- DRBDデバイスが物理ボリュームとなっていて、そのLVMがアクティブ (pvs)

上記はよくあるケースであり、すべてのケースを網羅するものではないという事に注意ください。

これで、DRBDモジュールをロードすることができます。

```
bob# modprobe drbd
```

/proc/drbd の出力を確認し、正しく(新しい)バージョンがロードされたか確認してください。もしインストールされているパッケージが間違ったカーネルバージョンのものだった場合には、 modprobe は有効ですが、古いバージョンが残っていれば再び有効にすることもできます。

'cat /proc/drbd' の出力は9.0.xを表示し、次のようになっているでしょう。

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

プライマリノードのaliceでは、 'cat /proc/drbd' はアップグレードをするまでは以前のバージョンを表示します。

4.7.7. 設定ファイルの移行

DRBD 9.0は8.4の設定ファイルに後方互換性がありますが、一部の構文は変更しています。変更点の全一覧は設定上の構文の変更点を参照してください。'drbdadm dump all' コマンドを使えば古い設定を簡単に移すことができます。これは、新しいリソース設定ファイルに続いて新しいグローバル設定も両方とも出力します。この出力を使って適宜変更してください。

4.7.8. メタデータの変更

次にオンディスクのメタデータを新しいバージョンに変更します。この手順はとても簡単で、1つコマンドを実行して2つの質問に答えるだけです。

たくさんのノードを変更したい場合には、追加のビットマップに十分なスペースを確保するため、すでに下位デバイスの容量を拡張していることでしょう。この場合には以下のコマンドを追加の引数 --max-peers=<N> を付けて実行します。対向ノードの数(予定)を決める時には、DRBDクライアントも考慮に入れてください。

DRBDのメタデータのアップグレードはとても簡単で、1つコマンドを実行して2つの質問に答えるだけです。

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

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

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

Found some data

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

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

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

もちろん、リソース名を all にすることも可能です。また、以下のようにすれば質問されるのを回避することができます(コマンド内の順序が重要です)。

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

4.7.9. DRBDを再び起動する

あとはDRBDデバイスを再びupにして起動するだけです。単純に、 drbdadm up all で済みます。

次は、クラスタ管理ソフトを使用しているか、手動でリソースを管理しているかによって方法が異なります。

- 手動

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

- Pacemaker

```
# crm node online bob
```

これによってDRBDは他のノードに接続し、再同期プロセスが開始します。

すべてのリソースが2つのノードで UpToDate になったら、アップグレードしたノード(ここでは bob)にアプリケーションを移動させることができます。そして、まだ8.4が稼働しているノードで同じ手順を行ってください。

4.7.10. DRBD9からDRBD9

既にDRBD9を使っている場合には新しいバージョンのパッケージをインストールすることができます。クラスタノードをスタンバイにして、カーネルモジュールをアンロード/リロードし、リソースを起動、そしてクラスタノードを再度 online にします。 [4]

個々の手順については上述の項の通りですので、ここでは省略します。

4.8. デュアルプライマリモードを有効にする

デュアルプライマリモードではリソースが複数ノードで同時にプライマリになることができます。永続的でも一時的なものでも可能です。

デュアルプライマリモードではリソースが同期レプリケート(プロトコルC)で設定されていることが必要です。そのためレイテンシに過敏となり、WAN環境には向いていません。

さらに、両リソースが常にプライマリとなるので、いかなるノード間のネットワーク不通でもスプリットブレインが発生します。

DRBD 9.0.xのデュアルプライマリモードは、ライブマイグレーションで使用する 2 プライマリに制限されています。

4.8.1. 永続的なデュアルプライマリモード

デュアルプライマリモードを有効にするため、リソース設定の net セクションで、 allow-two-primaries オプションを yes に指定します。

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

そして、両ノード間で設定を同期することを忘れないでください。両ノードで drbdadm adjust <resource> を実行してください。

これで drbdadm primary <resource> で、両ノードを同時にプライマリのロールにすることができます。

適切なフェンシングポリシーを常に実装すべきです。フェンシングなしで 'allow-two-primaries' を設定するのは危険です。これはフェンシングなしで、シングルプライマリを使うことより危険になります。

4.8.2. 一時的なデュアルプライマリモード

通常はシングルプライマリで稼動しているリソースを、一時的にデュアルプライマリモードを有効にするには次のコマンドを実行してください。

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

一時的なデュアルプライマリモードを終えるには、上記と同じコマンドを実行します。ただし --allow-two-primaries=no としてください(また、適切であれば希望するレプリケーションプロトコルにも)。

4.9. オンラインデバイス照合の使用

4.9.1. オンライン照合を有効にする

オンラインデバイス照合はデフォルトでは有効になっていません。有効にするには /etc/drbd.conf のリソース設定に以下の行を追加します。

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

<algorithm> は、システムのカーネル構成内のカーネルcrypto APIでサポートされる任意のメッセージダイジェストアルゴリズムです。通常は sha1 、 md5 、 crc32c から選択します。

既存のリソースに対してこの変更を行う場合は、 drbd.conf を対向ノードと同期し、両方のノードで drbdadm adjust <resource> を実行します。

4.9.2. オンライン照合の実行

オンライン照合を有効にしたら、次のコマンドでオンライン照合を開始します。

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

コマンドを実行すると、DRBDが <resource> に対してオンライン照合を実行します。同期していないブロックを検出した場合は、ブロックに非同期のマークを付け、カーネルログにメッセージを書き込みます。このときにデバイスを使用しているアプリケーションは中断なく動作し続けます。また、リソースロールの切り替えも行うことができます。

照合中に同期していないブロックが検出された場合は、照合の完了後に、次のコマンド使用して再同期できます。

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

4.9.3. 自動オンライン照合

通常は、オンラインデバイス照合を自動的に実行するほうが便利です。自動化は簡単です。**一方** のノードに /etc/cron.d/drbd-verify という名前で、次のような内容のファイルを作成します。

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

これにより、毎週日曜日の午前0時42分に、 cron がデバイス照合を呼び出します。そのため、月曜の朝にリソース状態をチェックして結果を確認することができます。デバイスが大きくて32時間では足りなかった場合、 コネクションステータスが VerifyS または VerifyT になっています。これは verify がまだ動作中であることを意味しています。

オンライン照合をすべてのリソースで有効にした場合（例えば /etc/drbd.d/global_common.conf の common セクションに verify-alg <アルゴリズム> を追加するなど)には、以下のようにします。

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

4.10. 同期速度の設定

バックグラウンド同期中は同期先のデータとの一貫性が一時的に失われるため、同期はできるだけ短時間で完了させるべきです。ただし、すべての帯域幅がバックグラウンド同期に占有されてしまうと、フォアグラウンドレプリケーションに使用できなくなり、アプリケーションのパフォーマンス低下につながります。これは避ける必要があります。同期用の帯域幅はハードウェアに合わせて設定する必要があります。

同期速度をセカンダリノードの最大書き込みスループットを上回る速度に設定しても意味がありません。デバイス同期の速度をどれほど高速に設定しても、セカンダリノードがそのI/Oサブシステムの能力より高速に書き込みを行うことは不可能です。

また、同じ理由で、同期速度をレプリケーションネットワークの帯域幅の能力を上回る速度に設定しても意味がありません。

4.10.1. 同期速度の計算

概算としては使用可能なレプリケーション帯域幅の30%程度を想定するのがよいでしょう。400MB/sの書き込みスループットを維持できるI/Oサブシステム、および110MB/sのネットワークスループットを維持できるギガビットイーサネットネットワークの場合は、ネットワークがボトルネックになります。速度は次のように計算できます。

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

図 6. syncer 速度の例(有効帯域幅が110MB/sの場合)

この結果、 resynce-rate オプションの推奨値は 33M になります。

一方、最大スループットが80MB/sのI/Oサブシステム、およびギガビットイーサネット接続を使用する場合は、I/Oサブシステムが律速要因になります。速度は次のように計算できます。

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

図 7. syncer 速度の例(有効帯域幅が80MB/sの場合)

この場合、 resync-rate オプションの推奨値は 24M です。

同じようにして800MB/sのストレージ速度で10Gbeのネットワークコネクションであれば、〜240MB/sの同期速度が得られます。

4.10.2. 可変同期速度設定

複数のDRBDリソースが1つのレプリケーション/同期ネットワークを共有する場合、同期が固定レートであることは最適なアプローチではありません。そのためDRBD 8.4.0から可変同期速度がデフォルトで有効になっています。このモードではDRBDが自動制御のループアルゴリズムで同期速度を決定して調整を行います。このアルゴリズムはフォアグラウンド同期に常に十分な帯域幅を確保し、バックグラウンド同期がフォアグラウンドのI/Oに与える影響を少なくします。

最適な可変同期速度の設定は、使用できるネットワーク帯域幅、アプリケーションのI/Oパターンやリンクの輻輳によって変わります。DRBD Proxyの有無によっても適切な設定は異なります。DRBDの機能を最適化するためにコンサルタントを利用するのもよいでしょう。以下は(DRBD Proxy使用を想定した環境での)設定の 一例 です。

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

c-fill-target の初期値は BDP×2 がよいでしょう。 BDP とはレプリケーションリンク上の帯域幅遅延積(Bandwidth Delay Product)です。

例えば、1GBit/sの接続の場合、200µsのレイテンシになります。 [5]1GBit/sは約120MB/sです。120掛ける200*10^{-6}秒は24000Byteです。この値はMB単位まで丸めてもよいでしょう。

別の例をあげると、100MBitのWAN接続で200msのレイテンシなら12MB/s掛ける0.2sです。2.5MBくらいになります。c-fill-target の初期値は3MBがよいでしょう。

他の設定項目については drbd.conf のマニュアルページを参照してください。

4.10.3. 永続的な固定同期速度の設定

ごく限られた状況 [6]では、固定同期速度を使うことがあるでしょう。この場合には、まず c-plan-ahead 0; にして可変同期速度調節の機能をオフにします。

そして、リソースがバックグラウンド再同期に使用する最大帯域幅をリソースの resync-rate オプションによって決定します。この設定はリソースの /etc/drbd.conf の disk セクションに記載します。

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

同期速度の設定は1秒あたりの バイト 単位であり ビット 単位でない点に注意してください。デフォルトの単位は kibibyte で、 4096 であれば 4MiB となります。

これはあくまでDRBDが行おうとする速度にすぎません。低スループットのボトルネック(ネットワークやストレージの速度)がある場合、設定した速度(いわゆる"理想値")には達しません。

4.10.4. 同期に関するヒント

同期すべきデータ領域の一部が不要になった場合、例えば、対向ノードとの接続が切れている時にデータを削除した場合などにはTrim/Discardのサポートの有効性が感じられるかもしれません。

c-min-rate は間違われやすいです。これは同期速度の **最低値** ではなくて、この速度以上の速度低下を **意図的に** 行わないという制限です。ネットワークとストレージの速度、ネットワーク遅延(これは回線共有による影響が大きいでしょう)、アプリケーションIO（関与することは難しいかもしれません）に応じた同期速度の管理まで **手が及ぶか** どうかによります。

4.11. チェックサムベース同期の設定

チェックサムベース同期はデフォルトでは有効になっていません。有効にするには、 /etc/drbd.conf のリソース構成に次の行を追加します。

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

<algorithm> は、システムのカーネル構成内のカーネルcrypto APIでサポートされる任意のメッセージダイジェストアルゴリズムです。通常は sha1 、 md5 、 crc32c から選択します。

既存のリソースに対してこの変更を行う場合は、 drbd.conf を対向ノードと同期し、両方のノードで drbdadm adjust <resource> を実行します。

4.12. 輻輳ポリシーと中断したレプリケーションの構成

レプリケーション帯域幅が大きく変動する環境(WANレプリケーション設定で典型的)の場合、レプリケーションリンクは時に輻輳します。デフォルト設定では、プライマリノードのI/Oのブロックを引き起こし、望ましくない場合があります。

その代わりに、進行中の同期を suspend (中断)に設定し、プライマリのデータセットをセカンダリから pull ahead (引き離す)にします。このモードではDRBDはレプリケーションチャネルを開いたままにし、切断モードにはしません。しかし十分な帯域幅が利用できるようになるまで実際にはレプリケートを行いません。

次の例は、DRBD Proxy構成のためのものです。

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

通常は congestion-fill と congestion-extents を pull-ahead オプションと合わせて設定するのがよい方法でしょう。

congestion-fill の値は以下の値の90%にするとよいでしょう。

- DRBD Proxy越しの同期の場合のDRBD Proxyのバッファメモリの割り当て、 または
- DRBD Proxy構成でない環境でのTCPネットワークの送信バッファ

congestion-extents の値は、影響するリソースの al-extents に設定した値の90%がよいでしょう。

4.13. I/Oエラー処理方針の設定

DRBDが 下位レベルI/Oエラーを処理する際の方針は、リソースの /etc/drbd.conf の disk セクションの on-io-error で指定します。

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

すべてのリソースのグローバルI/Oエラー処理方針を定義したい場合は、これを common セクションで設定します。

<strategy> は以下のいずれかを指定します。

detach
これがデフォルトで、推奨オプションです。下位レベルI/Oエラーが発生すると、DRBDはそのノードの下位デバイスを切り離し、ディスクレスモードで動作を継続します。

pass-on
上位層にI/Oエラーを通知します。プライマリノードの場合は、マウントされたファイルシステムに通知されます。セカンダリノードの場合は無視されます(セカンダリノードには通知すべき上位層がないため)。

call-local-io-error
ローカルI/Oエラーハンドラとして定義されたコマンドを呼び出します。このオプションを使うには、対応する local-io-error ハンドラをリソースの handlers セクションに定義する必要があります。local-io-error で呼び出されるコマンド(またはスクリプト)にI/Oエラー処理を実装するかどうかは管理者の判断です。

DRBDの以前のバージョン(8.0以前)にはもう1つのオプション panic があり、これを使用すると、ローカルI/Oエラーが発生するたびにカーネルパニックによりノードがクラスタから強制的に削除されました。このオプションは現在は使用できませんが、 local-io-error / call-local-io-error インタフェースを使用すると同じような動作を実現します。ただし、この動作の意味を十分理解した上で使用してください。

次のコマンドで、実行中のリソースのI/Oエラー処理方針を再構成することができます。

- /etc/drbd.d/<resource>.res のリソース構成の編集
- 構成の対向ノードへのコピー
- 両ノードでの drbdadm adjust <resource> の実行

4.14. レプリケーショントラフィックの整合性チェックを設定

レプリケーショントラフィックの整合性チェック はデフォルトでは有効になっていません。有効にする場合は、 /etc/drbd.conf のリソース構成に次の行を追加します。

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

<algorithm> は、システムのカーネル構成内のカーネルcrypto APIでサポートされる任意のメッセージダイジェストアルゴリズムです。通常は sha1 、 md5 、 crc32c から選択します。

既存のリソースに対してこの変更を行う場合は、 drbd.conf を対向ノードと同期し、両方のノードで drbdadm adjust <resource> を実行します。

この機能は本番環境での使用は想定していません。データ破壊の問題や、通信経路(ネットワークハードウェア、ドライバ、スイッチ)に障害があるかどうかを診断する場合にのみ使用してください。

4.15. リソースのサイズ変更

4.15.1. オンライン拡張

動作中(オンライン)に下位ブロックデバイスを拡張できる場合は、これらのデバイスをベースとするDRBDデバイスについても動作中にサイズを拡張することができます。その際に、次の2つの条件を満たす必要があります。

1. 影響を受けるリソースの下位デバイスが、 LVMやEVMSなどの論理ボリューム管理サブシステムによって管理されている。
2. 現在、リソースのコネクションステータスが Connected になっている。

両方のノードの下位ブロックデバイスを拡張したら、一方のノードだけがプライマリ状態であることを確認してください。プライマリノードで次のように入力します。

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

新しいセクションの同期がトリガーされます。同期はプライマリノードからセカンダリノードへ実行されます。

追加する領域がクリーンな場合には、追加領域の同期を—assume-cleanオプションでスキップできます。

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

4.15.2. オフライン拡張する

外部メタデータを使っている場合、DRBD停止中に両ノードの下位ブロックデバイスを拡張すると、新しいサイズが自動的に認識されます。管理者による作業は必要ありません。両方のノードで次にDRBDを起動した際に、DRBDデバイスのサイズが新しいサイズになり、ネットワーク接続が正常に確立します。

DRBDリソースで内部メタデータを使用している場合は、リソースのサイズを変更する前に、メタデータを拡張されるデバイス領域の後ろの方に移動させる必要があります。これを行うには次の手順を実行します。

これは高度な手順です。慎重に検討した上で実行してください。

- DRBDリソースを停止します。

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

- 縮小する前に、メタデータをテキストファイルに保存します。

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

各ノードごとにそれぞれのダンプファイルを作成する必要があります。この手順は、両方のノードでそれぞれ実行します。一方のノードのメタデータのダンプを対向ノードに **コピーしないでください** 。 DRBDは **動作しなくなります** 。

- 両方のノードの下位ブロックデバイスを拡大します。
- /tmp/metadata ファイルのサイズ情報(la-size-sect)を書き換えます。la-size-sect は、必ずセクタ単位で指定する必要があります。
- メタデータ領域の再初期化をします。

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

- 両ノードで修正したメタデータをインポートします。

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

この例では bash パラメータ置換を使用しています。他のシェルの場合、機能する場合もしない場合もあります。現在使用しているシェルが分からない場合は、 SHELL 環境変数を確認してください。

- 再度DRBDリソースを起動します。

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

- 一方のノードでDRBDリソースを昇格します。

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

- 最後に、拡張したDRBDデバイスを活用するために、ファイルシステムを拡張します。

4.15.3. オンライン縮小

警告 : オンラインでの縮小は外部メタデータ使用の場合のみサポートしています。

DRBDデバイスを縮小する前に、DRBDの上位層(通常はファイルシステム)を縮小 しなければいけません 。ファイルシステムが実際に使用している容量を、DRBDが知ることはできないため、データが失われないように注意する必要があります。

ファイルシステムをオンラインで縮小できるかどうかは、使用しているファイルシステムによって異なります。ほとんどのファイルシステムはオンラインでの縮小をサポートしません。XFSは縮小そのものをサポートしません。

オンラインでDRBDを縮小するには、その上位に常駐するファイルシステムを縮小した_後に_、次のコマンドを実行します。

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

<new-size> には通常の乗数サフィックス(K、M、Gなど)を使用できます。DRBDを縮小したら、DRBDに含まれるブロックデバイスも縮小できます(デバイスが縮小をサポートする場合)。

DRBDのメタデータがボリュームの想定される箇所に **実際に** 書き込まれるように下位デバイスのリサイズ後に drbdadm resize <resource> を実行してもよいでしょう。

4.15.4. オフライン縮小

DRBDが停止しているときに下位ブロックデバイスを縮小すると、次にそのブロックデバイスを接続しようとしてもDRBDが拒否します。これは、ブロックデバイスが小さすぎる(外部メタデータを使用する場合)、またはメタデータを見つけられない(内部メタデータを使用する場合)ことが原因です。この問題を回避するには、次の手順を行います (オンライン縮小を使用出来ない場合)。

これは高度な手順です。慎重に検討した上で実行してください。

- DRBDがまだ動作している状態で、一方のノードのファイルシステムを縮小します。
- DRBDリソースを停止します。

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

- 縮小する前に、メタデータをテキストファイルに保存します。

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

各ノードごとにそれぞれのダンプファイルを作成する必要があります。この手順は、両方のノードでそれぞれ実行します。一方のノードのメタデータのダンプを対向ノードに **コピーしないでください** 。 DRBDは **動作しなくなります** 。

- 両方のノードの下位ブロックデバイスを縮小します。
- /tmp/metadata ファイルのサイズ情報(la-size-sect)を書き換えます。la-size-sect は、必ずセクタ単位で指定する必要があります。
- *内部メタデータを使用している場合は、メタデータ領域を再初期化します(この時点では、縮小によりおそらく内部メタデータが失われています)。

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

- 両ノードで修正したメタデータをインポートします。

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

この例では bash パラメータ置換を使用しています。他のシェルの場合、機能する場合もしない場合もあります。現在使用しているシェルが分からない場合は、 SHELL 環境変数を確認してください。

- 再度DRBDリソースを起動します。

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

4.16. 下位デバイスのフラッシュを無効にする

バッテリバックアップ書き込みキャッシュ(BBWC)を備えたデバイスでDRBDを実行している場合にのみ、デバイスのフラッシュを無効にできます。ほとんどのストレージコントローラは、バッテリが消耗すると書き込みキャッシュを自動的に無効にし、バッテリが完全になくなると即時書き込み(ライトスルー)モードに切り替える機能を備えています。このような機能を有効にすることを強くお勧めします。

BBWC機能を使用していない、またはバッテリが消耗した状態でBBWCを使用しているときに、DRBDのフラッシュを無効にすると、 データが失われるおそれがあります したがって、これはお勧めできません。

DRBDは下位デバイスのフラッシュを、レプリケートされたデータセットとDRBD独自のメタデータについて、個別に有効と無効を切り替える機能を備えています。この2つのオプションはデフォルトで有効になっています。このオプションのいずれか(または両方)を無効にしたい場合は、DRBD設定ファイルの /etc/drbd.conf の disk セクションで設定できます。

レプリケートされたデータセットのディスクフラッシュを無効にするには、構成に次の行を記述します。

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

DRBDのメタデータのディスクフラッシュを無効にするには、次の行を記述します。

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

リソースの構成を修正し、また、もちろん両ノードの /etc/drbd.conf を同期したら、両ノードで次のコマンドを実行して、これらの設定を有効にします。

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

1台のサーバのみがBBWCがある場合 [7]には、ホストセクションに以下のような設定をしてください。

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

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

4.17. スプリットブレイン時の動作の設定

4.17.1. スプリットブレインの通知

スプリットブレインが 検出される と、DRBDはつねに split-brain ハンドラを呼び出します(設定されていれば)。このハンドラを設定するには、リソース構成に次の項目を追加します。

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

<handler> はシステムに存在する任意の実行可能ファイルです。

DRBDディストリビューションでは /usr/lib/drbd/notify-split-brain.sh という名前のスプリットブレイン対策用のハンドラスクリプトを提供しています。これは指定したアドレスに電子メールで通知を送信するだけのシンプルなものです。root@localhost (このアドレス宛のメールは実際のシステム管理者に転送されると仮定)にメッセージを送信するようにハンドラを設定するには、 split-brain handler を次のように記述します。

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

実行中のリソースで上記の変更を行い(ノード間で設定ファイルを同期すれば)、後はハンドラを有効にするための他の操作は必要ありません。次にスプリットブレインが発生すると、DRBDが新しく設定したハンドラを呼び出します。

4.17.2. スプリットブレインからの自動復旧ポリシー

スプリットブレイン(またはその他)のシナリオの結果、データの相違状況を自動的に解決するDRBDの構成は、潜在的な **自動データ損失** を構成することを意味します。その意味をよく理解し、それを意味しない場合は設定しないでください。

むしろ、フェンシングポリシー、クォーラム設定、クラスタマネージャの統合、クラスタマネージャの冗長化通信リンクなどを調べて、まず最初にデータの相違状況をつくらないようにすべてきです。

スプリットブレインからの自動復旧ポリシーには、状況に応じた複数のオプションが用意されています。DRBDは、スプリットブレインを検出したときのプライマリロールのノードの数にもとづいてスプリットブレイン回復手続きを適用します。そのために、DRBDはリソース設定ファイルの net セクションの次のキーワードを読み取ります。

after-sb-0pri
スプリットブレインが検出されたときに両ノードともセカンダリロールの場合に適用されるポリシーを定義しま

す。次のキーワードを指定できます。

- disconnect: 自動復旧は実行されません。 split-brain ハンドラスクリプト(設定されている場合)を呼び出し、 コネクションを切断して切断モードで続行します。
- discard-younger-primary: 最後にプライマリロールだったホストに加えられた変更内容を破棄して ロールバックします。
- discard-least-changes:変更が少なかったほうのホストの変更内容を破棄してロールバックします。
- discard-zero-changes: 変更がなかったホストがある場合は、 他方に加えられたすべての変更内容を適用して 続行します。

after-sb-1pri

スプリットブレインが検出されたときにどちらか1つのノードがプライマリロールである場合に適用されるポリシーを定義します。次のキーワードを指定できます。

- disconnect: after-sb-0pri と同様に split-brain ハンドラスクリプト(構成されている場合)を呼び出し、 コネクションを切断して切断モードで続行します。
- consensus: after-sb-0pri で設定したものと同じ復旧ポリシーが適用されます。 これらのポリシーを適用した後で、 スプリットブレインの犠牲ノードを選択できる場合は自動的に解決します。それ以外の場合は、 disconnect を指定した場合と同様に動作します。
- call-pri-lost-after-sb: after-sb-0pri で指定した復旧ポリシーが適用されます。 これらのポリシーを適用した後で、 スプリットブレインの犠牲ノードを選択できる場合は、犠牲ノードで pri-lost-after-sb ハンドラを起動します このハンドラは handlers セクションで設定する必要があります。 また、クラスタからノードを強制的に削除します。
- discard-secondary: 現在のセカンダリロールのホストを、 スプリットブレインの犠牲ノードにします。

after-sb-2pri

スプリットブレインが検出されたときに両ノードともプライマリロールである場合に適用されるポリシーを定義します。このオプションは after-sb-1pri と同じキーワードを受け入れます。ただし、 discard-secondary と consensus は除きます。

上記の3つのオプションで、DRBDは他のキーワードも認識しますが、それらはめったに使用されないためここでは省略します。ここで説明されていないスプリットブレインの復旧キーワードに関しては drbd.conf のマニュアルページを参照ください。

たとえば、デュアルプライマリモードでGFSまたはOCFS2ファイルシステムのブロックデバイスとして機能するリソースの場合、次のように復旧ポリシーを定義できます。

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

4.18. スタック3ノード構成の作成

3ノード構成では、1つのDRBDデバイスを別のデバイスの上に スタック (積み重ね)します。

DRBD9.xでは1つの階層で複数ノードを使用できるのでスタッキングは非推奨です。詳細は ネットワークコネクションの定義をご参照ください。

4.18.1. デバイススタックの留意事項

次のような事項に注意する必要があります。

- スタックデバイス側が利用可能です。1つのDRBDデバイス /dev/drbd0 を設定して、その上位にスタックデバイス /dev/drbd10 があるとします。この場合は、 /dev/drbd10 がマウントして使用するデバイスになります 。
- 下位のDRBDデバイス および スタックDRBDデバイス(上位DRBDデバイス)の両方にそれぞれメタデータが存在します 。上位デバイスには、必ず 内部メタデータを使用してください。このため、 3ノード構成時の使用可能なディスク領域は、 2ノード構成に比べてわずかに小さくなります。
- スタックした上位デバイスを実行するには、下位のデバイスが プライマリロールになっている必要があります。
- バックアップノードにデータを同期するには、 アクティブなノードのスタックデバイスがプライマリモードで動作している必要があります。

4.18.2. スタックリソースの設定

次の例では alice 、 bob 、 charlie という名前のノードがあり、 alice と bob が2ノードクラスタを構成し、 charlie がバックアップノードになっています。

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

  on alice {
    address   10.0.0.1:7788;
  }

  on bob {
    address   10.0.0.2:7788;
  }
}

resource r0-U {
  protocol A;

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

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

他の drbd.conf 設定ファイルと同様に、この設定ファイルもクラスタのすべてのノード(この場合は3つ)に配布する必要があります。非スタックリソースの設定にはない、次のキーワードにご注意ください。

stacked-on-top-of

このオプションによって、DRBDに含まれるリソースがスタックリソースであることをDRBDに知らせます。これは、通常リソース設定内にある on セクションのうちの１つを置き換えます。stacked-on-top-of は下位レベルのリソースには使用しないでください。

スタックリソースにProtocol Aを使用することは必須ではありません。アプリケーションに応じて任意のDRBDのレプリケーションプロトコルを選択できます。

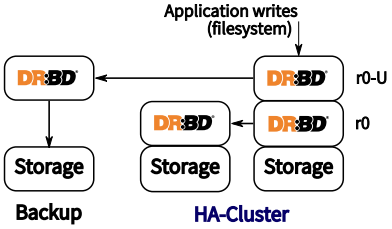


図 8. 単一スタックの構成

4.18.3. スタックリソースを有効にする

スタックリソースを有効にするには、まず、下位レベルリソースを有効にしてどちらか一方をプライマリに昇格します。

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

非スタックリソースと同様に、スタックリソースの場合もDRBDメタデータを作成する必要があります。次のコマンドで実行します。次に、スタックリソースを有効にします。

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

この後でバックアップノードのリソースを起動し、3ノードレプリケーションを有効にします。

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

この後でバックアップノードのリソースを起動し、3ノードレプリケーションを有効にします。

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

クラスタ管理システムを使えばスタックリソースの管理を自動化できます。Pacemakerクラスタ管理フレームワークで管理する方法については、PacemakerクラスタでスタックDRBDリソースを使用するを参照してください。

4.19. 永続的なディスクレスノード

ノードはしばしDRBDの中で永続的にディスクレスになるときがあります。以下はディスクをもつ３つのノードと永続的なディスクレスノードの構成例です。

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

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

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

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

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

永続的なディスクレスノードはビットマップスロットが割り当てられません。このようなノードのステータスは、エラーでも予期せぬ状態でもないので緑で表示されます。

DRBDクライアントはワイアデータ通信の簡単な実装ですが、iSCSIの Persistent Reservations のような高度な機能はありません。しかし、仮想マシン等で使われる read, write, trim/discard, resize のような基本的なIOのみが必要な場合には、適切に動作します。

4.20. データ再配置

以下の例では、すべてのデータ領域は3台に冗長化するポリシーを想定しています。したがって最小構成で3台のサーバが必要です。

しかし、データ量が増えてくると、サーバ追加の必要性に迫られます。その際には、また新たに3台のサーバを購入する必要はなく、１つのノードだけを追加をしてデータを 再配置 することができます。

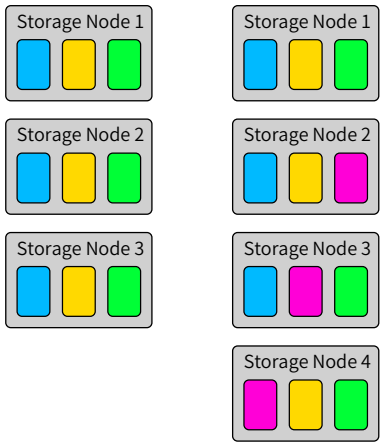


図 9. DRBDデータ再配置

上の図では、3ノードの各々に25TiBのボリュームがある合計75TiBの状態から、4ノードで合計100TiBの状態にしています。

データを再配置する時には、 新しい ノードとDRBDリソースを削除したいノードを選択する必要があります。 現在 アクティブ なノードからリソースを削除する場合には注意が必要です。 DRBDが プライマリ のノードからリソースを削除する場合にはサービスを移行するか、そのノードのリソースをDRBDクライアントとして稼働させるかにしてください。通常は セカンダリ のノードから選ぶほうが簡単です(それができない場合もあるでしょうが)。

4.20.1. ビットマップスロットの用意

移動するリソースには一時的に未使用のビットマップスロット用のスペースが必要です。

drbdadm create-md の時にもう一つ割り当てます。または、 drbdadm がもう１つスロットを取っておくように設定にプレースホルダを設けてもよいでしょう。

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

このスロットを実際に利用するには以下が必要です。

1. メタデータをダンプする,
2. メタデータ領域を拡大する,
3. ダンプファイルを編集する,
4. メタデータをロードする

今後のバージョンの drbdadm ではショートカットキーが用意されます。 drbdadm resize --peers N が使用できるようになる予定であり、カーネルがメタデータを書き換えられるようになります。

4.20.2. 新しいノードの用意と有効化

まずは新しいノードに(lvcreate 等で)下位のストレージリュームを作成する必要があります。 次に設定ファイルのプレイスホルダーに現在のホスト名、アドレス、ストレージのパスを書き込みます。そしてすべての必要なノードにリソース設定をコピーします。

新しいノードでメタデータを次のようにして(一度だけ)初期化します

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

4.20.3. 初期同期の開始

新しいノードでデータを受け取ります。

そのために以下のようにして既存のノードでネットワークコネクションを定義します。

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

そして新しいノードでDRBDデバイスを開始します。

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

4.20.4. 接続確認

この時に、新しいノードで以下のコマンドを実行してください。

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

他のノード すべて が接続しているか確認します。

4.20.5. 初期同期後

新しいノードが UpToDate になったらすぐに、設定中の他のノードのうち１つが for-later-rebalancing に変更できるようになり、次のマイグレーション用にキープしておけるようになります。

次の再配置用に小さなビットマップスロットしかない新規ノードに drbdadm create-md をすることにはリスクがあると思う方もいることでしょう。あらかじめIPアドレスとホスト名を用意しておくのが簡単な方法でしょう。

再度変更した設定をコピーして、以下のコマンドを

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

全ノードで実行します。

4.20.6. クリーニング

今までデータのあったノードでは、もうリソースを使用しない場合にはDRBDデバイスを次のようにして停止できます。

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

これで下位デバイスを使用しなくなり、他の用途で使用することができるようになります。論理ボリュームであれば lvremove でボリュームグループに戻すことができます。

4.20.7. まとめと他のステップ

１つのリソースを新しいノードに移動しました。同様の手順で１つまたはそれ以上のリソースを、クラスタ内の既存の2つまたは3つのノードの空きスペースに行う事も可能です。

再び3重の冗長化を行うのに必要な空き容量のあるノードが揃えば、新しいリソースを設定することが出来ます。

なお、上記の手順をDRBD Manageを使用して行う場合については[s-dm-rebalance]をご参照ください。

4.21. クォーラム設定

スプリットブレインや複製データの相違を防ぐために、フェンシングを構成する必要があります。フェンシングのすべてのオプションは、最後には冗長な通信路に依存します。それはノードが対向ノードのIPMIネットワークインタフェースに接続する形かもしれません。crm-fence-peerスクリプトの場合、DRBDのネットワークリンクが切断されたときに、pacemaker の通信が有効であることが必要になります。

一方クォーラムは完全に異なるアプローチをとります。基本的な考え方は、通信できるノードの数がノード総数の半分を超える場合、クラスタパーティションは複製されたデータセットを変更できるということです。そのようなパーティションのノードは、クォーラムを持つといいます。言い換えると、クォーラムを持たないノードは、複製されたデータを変更しないことを保証します。これはデータの相違を作成しないことを意味します。

DRBDのクォーラムの実装は、quorum リソースに majority , all または数値を設定することで有効にできます。majority の動作が、前の文章で説明した動作になります。

4.21.1. 保証された最低限の冗長化

デフォルトでは、ディスクを持つすべてのノードがクォーラム選挙で投票権を持ちます。言い換えると、ディスクレスノードだけが持ちません。従って、3ノードクラスタでは、 Inconsistent ディスクを持つ2つのノード、UpToDate を持つ1つノードは、クォーラム投票権を持つことができます。 quorum-minimum-redundancy を設定することによって、UpToDate ノードだけがクォーラム選挙で投票権を持つように変更することもできます。このオプションは quorum オプションと同じ引数をとります。

このオプションを使用すると、サービスを開始する前に必要な再同期操作が完了するまで待機する、ということも表現できます。 したがって、データの最低限の冗長性がサービスの可用性よりも保証されることを好む方法です。金融データとサービスなどはその一例です。

以下に5ノードクラスタの設定例を示します。パーティションは少なくとも3つのノードが必要であり、そのうち2つは UpToDate である必要があります。

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

4.21.2. クォーラムを失った時の動作

サービスを実行しているノードがクォーラムを失うと、すぐにデータセットの書き込み操作を中止する必要があります。これはIOがすぐにすべてのIO要求をエラーで完了し始めることを意味します。通常、これは、正常なシャットダウンが不可能であることを意味します。これはデータセットをさらに変更する必要があるためです。IOエラーはブロックレベルからファイルシステムに、ファイルシステムからユーザースペースアプリケーションに伝わります。

理想的には、IOエラーの場合、アプリケーションを単に終了させることです。これはその後、Pacemaker がファイルシステムをアンマウントし、DRBDリソースを降格させセカンダリロールに移すことを可能にします。その場合は、on-no-quorum リソースに io-error を設定してください。以下はその設定例です。

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

アプリケーションが最初のIOエラーで終了しない場合は、IOをフリーズさせノードをリブートさせることもできます。以下はその設定例です。

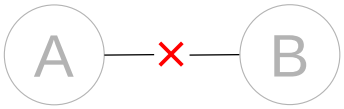
```
resource quorum-demo {
  quorum majority;
  on-no-quorum suspend-io;
  ...

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

4.21.3. タイブレーカーとしてのディスクレスノードを使用

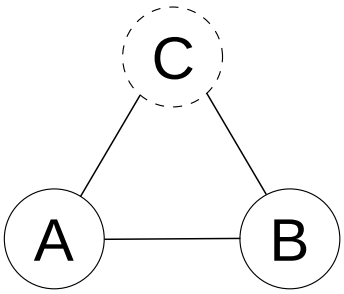
クラスタ内のすべてのノードに接続されているディスクレスノードを利用して、クォーラムネゴシエーションプロセスのタイブレークを解消できます。

ノードAがプライマリで、ノードBがセカンダリである、次の2ノードクラスタを考えます。

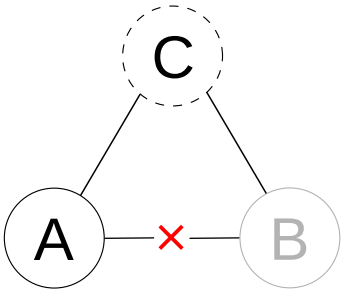
2つのノード間の接続が失われるとすぐに、2つのノードはクォーラムを失い、クラスタ上のアプリケーションはデータを書き込めなくなります。



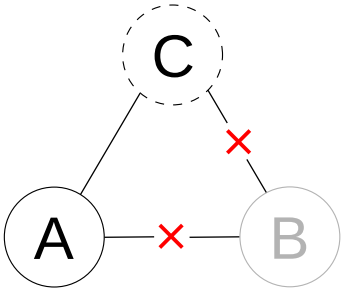
3番目のノードCをクラスタに追加し、それをディスクレスとして構成すると、タイブレーカーメカニズムを利用できます。



この場合、プライマリノードとセカンダリノードが接続を失っても、両ノードはディスクレスタイブレーカーとの接続をまだ維持しています。このため、プライマリは機能し続けることができますが、セカンダリはそのディスクをOutdatedに降格させるため、サービスをそこに移行することはできません。



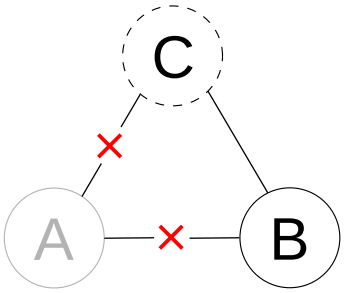
同時に2つの接続が失われる場合は特別なケースであり、次に説明します。



この場合、タイブレーカーノードはプライマリとパーティションを形成します。そのため、プライマリはクォーラムを維持し、セカンダリは古くなります。セカンダリのディスクの状態は "UpToDate" のままですが、ク

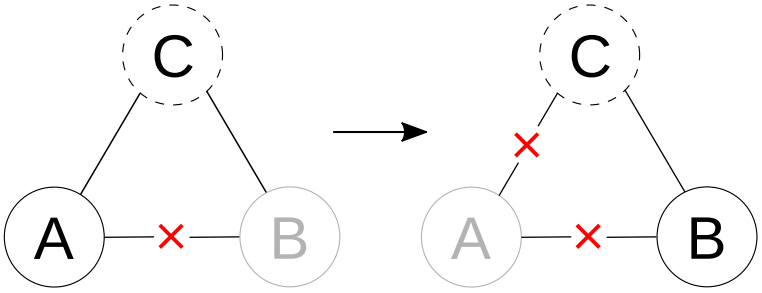
ォーラムがないためにプライマリに昇格できません。

プライマリがタイブレーカーとの接続を失う可能性を考えてみます。



この場合、プライマリは使用できなくなり、"クォーラム中断" 状態になります。これにより、DRBD上のアプリケーションが I/O エラーを受信します。その後、クラスタマネージャはノードBをプライマリに昇格させ、代わりにそこでサービスを実行し続けることができます。

ディスクレスタイブレーカーが "サイドを切り替える" 場合、データの相違を避ける必要があります。このシナリオを考えます。

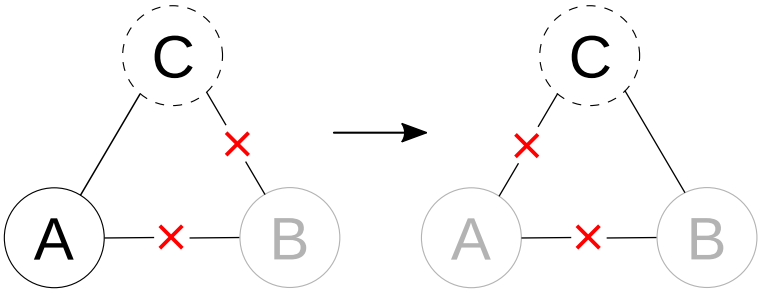


プライマリとセカンダリ間の接続が失われ、プライマリのディスクレスノードへの接続が突然失われた場合でも、アプリケーションはプライマリ上で実行を継続しています。

この場合は、どのノードもプライマリに昇格できず、クラスタは動作を継続できません。

データの分離を避けることは、サービスの可用性を確保することよりも常に優先されます。

他のシナリオを見てみます。



ここでは、アプリケーションはプライマリで実行されていますが、セカンダリは利用できません。その後、タイブレーカーはプライマリへの接続を失い、次にセカンダリに再接続します。ここで重要なのは、クォーラムを失ったノードは、ディスクレスノードに接続してもクォーラムを取り戻すことはできないことです。したがって、この場合、どのノードもクォーラムを持たず、クラスタは停止します。

4.21.4. ラストマン・スタンディング

クラスタから正常に離脱するノードは、障害が発生したノードとは異なるものとしてカウントされることに注意する必要があります。正常に離脱した場合、離脱ノードのデータは Outdated(無効状態) としてマークされ、残

りのノードにデータが無効状態であることを伝えることができます。

すべてのディスクが無効状態になっているノードのグループでは、そのグループの誰もプライマリに昇格できません [8]。

つまり、1 つのノードを除くすべてのノードがクラスターから離脱した場合、そのすべてのノードが正常に離脱している限り、クォーラムを維持できます。1 つのノードが正常に離脱しなかった場合、残ったすべてのノードでパティションを形成し、UpToDate データにアクセスすると仮定します。他のパーティションが（潜在的に）自分のパーティションよりも大きい場合、クォーラムを失います。

[3] 例:3つのノード間接続用と最低1つの外部接続/管理用インターフェース
[4] 少なくとも、過去にあった書き込み時の状態;)
[5] 概算は ping の結果を使用しています
[6] ベンチマークなど
[7] 例えばDRサイトでは異なるハードウェアを使うでしょう
[8] 例外は -force フラグを使用した手動昇格です。 --force を使用する場合、それが何を意味するかを認識していると仮定します。

5. DRBD Proxyの使用

5.1. DRBD Proxyの使用においての検討事項

DRBD Proxyプロセスは、DRBDが設定されているマシン上に直接配置するか、個別の専用サーバに配置することができます。DRBD Proxyインスタンスは、複数のノードの複数のDRBDデバイスのプロキシとして機能することができます。

DRBD ProxyはDRBDに対して完全に透過的です。通常は大量のデータパケットがDRBD Proxyを含む転送経路に溜まるため、アクティビティログがかなり大きくなります。これは、プライマリノードのクラッシュ後の長い再同期の実行を引き起こす可能性があるので、それはDRBDの csums-alg 設定を有効にすることをお勧めします。

DRBD Proxyのより詳細な情報についてはDRBD Proxyによる遠距離レプリケーションをご参照ください。

DRBD Proxy 3ではLiunxカーネル2.6.26以降からのカーネルの機能を使用しています。そのため、RHEL5などの古いシステムでは使用できません。なお、現在でもDRBD Proxy1のパッケージは提供しています。 [9].

5.2. インストール

DRBD Proxyを入手するには、(日本では)サイオステクノロジー株式会社またはその販売代理店に連絡してください。特別な理由がない限り、常に最新バージョンのDRBD Proxyを使用してください。

DebianとDebianベースのシステム上でDRBD Proxyをインストールするには、dpkgを次のように使用します(DRBD Proxyのバージョンとアーキテクチャは、ターゲットのアーキテクチャに合わせてください)。

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

RPMベースのシステム(SLESやRedhat)にDRBD Proxyをインストールする場合は、次のコマンドを使用します(DRBD Proxyのバージョンとアーキテクチャは、ターゲットのアーキテクチャに合わせてください)。

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

DRBD Proxyの設定にはdrbdadmが必要なので、これもインストールします。

DRBD Proxyバイナリだけでなく、 /etc/init.d に通常に入る起動スクリプトもインストールします。このスクリプトは単に起動/停止するだけでなく、 drbdadm を使ってDRBD Proxyの動作も設定します。

5.3. ライセンスファイル

DRBD Proxyの実行には、ライセンスファイルが必要です。DRBD Proxyを実行したいマシンにライセンスファイルを設定してくださいこのファイルは drbd-proxy.license と呼ばれ、対象マシンの /etc ディレクトリにコピーされ、また drbdpxy ユーザー/グループに所有されている必要があります。

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

5.4. LINSTORを使用した設定

DRBD Proxyは、LINSTOR ユーザーズガイドで説明しているようにLINSTORを使って構成できます

5.5. リソースファイルを使用した設定

DRBD Proxyもまたリソースファイルを編集して構成します。設定は、追加のオプションセクション proxy とホストセクション内の proxy on セクションで行います。

DRBDノードで直接実行されるプロキシのDRBD Proxyの設定例を次に示します。

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

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

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

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

inside IPアドレスはDRBDとDRBD Proxyとの通信に使用し、 outside IPアドレスはプロキシ間の通信に使用します。後者はファイヤーウォール設定で許可する必要があります。

5.6. DRBD Proxyの制御

drbdadm には proxy-up および proxy-down サブコマンドがあり、名前付きDRBDリソースのローカルDRBD Proxyプロセスとの接続を設定したり削除したりできます。これらのコマンドは、/etc/init.d/drbdproxy が実装する start および stop アクションによって使用されます。

DRBD Proxyには drbd-proxy-ctl という下位レベル構成ツールがあります。このツールをオプションを指定せずに呼び出した場合は、対話型モードで動作します。

対話型モードをにせずコマンドを直接渡すには、 '-c' パラメータをコマンドに続けて使用します。

使用可能なコマンドを表示するには次のようにします。

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

コマンドの周りのダブルクォートは読み飛ばされる点に注意ください。

以下にコマンドを列挙します。最初のいくつかのコマンドは通常は直接使用することはありません(drbdadm proxy-up や drbdadm proxy-down コマンド経由で使用されます)。それ以降のものは様々なステータスや情報を表示します。

add connection <name> lots of arguments
通信経路を作成します。これは drbdadm proxy-up コマンド経由で使用されるものなので、長い構文は省略しています。

del connection <name>
通信経路を削除します。

set memlimit <name> <memlimit-in-bytes>
コネクションにメモリ制限を設けます。これは新規に設定したときのみ有効で、稼働中に変更することはできません。 このコマンドでは通常使用する単位の k 、 M 、 G を使用できます。

show
現在設定されている通信経路を表示します。

show memusage
各コネクションでのメモリ使用量を表示します。

例：

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

メモリ使用を監視します。上記に挙げているように、クォートが必要である点にご注意ください。

show [h]subconnections
現在接続中の各コネクションを種々の情報と共に表示します。 h をつけると、人間が可読のバイト単位のフォーマットで出力します。

show [h]connections
現在接続中のコネクションをステータスと共に表示します。 h をつけると、人間が可読のバイト単位のフォーマットで出力します。

Status の行では以下のうちいずれかのステータスを表示します。

- Off: 対向のDRBD Proxyプロセスとの通信経路がない。
- Half-up: 対向のDRBD Proxyプロセスとの接続はおそらく確立しているものの、ProxyとDRBD間の経路がまだ確立していない。
- DRBD-conn: 最初の数パケットをコネクションを通じて送信してはいるものの、まだスプリットブレインなどの状態にある。
- Up: DRBDのコネクションが完全に確立された状態。

shutdown
drbd-proxy プログラムをシャットダウンする。Attention: 本操作を行うと、DRBD Proxyを使ったすべてのDRBDコネクションが終了します。

quit
drbd-proxy-ctlプログラムを終了します(プログラムとの接続を閉じます)。※DRBD Proxy自体は動作したままです。

print statistics
現在アクティブなコネクションの読みやすいフォーマットでの詳細な統計情報を表示します。この機能をご使用の監視方法に統合して利用するのもよいでしょう。

上述のコマンド群はすべて root ユーザーなどのUID0のユーザーだけが実行できますが、このコマンドは全ユーザが使用できます(/var/run/drbd-proxy/drbd-proxy-ctl.socket へのアクセス権があれば)。 /etc/init.d/drbdproxy の権限を設定している箇所をご確認ください。

5.7. DRBD Proxyプラグインについて

DRBD proxy3以降のプロキシではWANコネクション用のプラグインを使用できます。現在使用できるプラグインは zstd, lz4, zlib, lzma (すべてのソフトウェア圧縮), aha (ハードウェア圧縮サポート。詳細は http://www.aha.com/data-compression/ 参照ください。

zstd (Zstandard）は、高圧縮率を提供するリアルタイム圧縮アルゴリズムです。非常に高速なデコーダーのもとで、圧縮/速度の幅広いトレードオフ値を提供します。圧縮率はレベルパラメーターに依存し、1〜22 の範囲で指定できます。レベル 20 を超えると、Drbd プロキシに必要なメモリが増加します。

lz4 は非常に高速な圧縮アルゴリズムです。 通常データを1/2から1/4に圧縮でき、使用するネットワーク帯域も1/2から3/2程度になります。

zlib プラグインはGZIPアルゴリズムを圧縮に使用します。 lz4 よりも多少CPUを消費しますが、1/3から1/5になります。

lzma プラグインは liblzma2 ライブラリを使用します。数百MiBの辞書を使って、小さな変更であっても非常に効率的な繰り返しデータの差分符号化を行います。 lzma はより多くCPUとメモリを必要としますが、 zlib よりも高い圧縮率になります。DRBD上にVMを置いた実際の環境でテストしたところ、1/10から1/40になりました。lzma プラグインはライセンスで有効化する必要があります。

aha はAHA367PCIe (10Gbit/sec)やAHA372 (20GBit/sec)などのハードウェア圧縮カードを使用します。現在のハードウェアではこれがもっとも高速な圧縮です。ahaプラグインはライセンスで有効化する必要があります。

ご利用の環境に最適な設定についてはLINBIT(またはサイオステクノロジー)へご相談ください。性能はCPU(速度、スレッド数)、メモリ、帯域幅の入出力、CPUスパイクなどに依存します。一週間分の sysstat データがあれば、設定を決定するのに役立ちます。

proxy セクションの compression on は現在使用していません。近いうちに廃止する予定です。現在は zlib level 9 として扱います。

5.7.1. WAN側の帯域幅制限を使用する

DRBD Proxyの実験的な bwlimit は壊れていますので、使わないでください。DRBDを使うアプリケーションがIOでブロックするかもしれません。これは将来削除されます。

代わって、Linuxカーネルのトラフィック制御フレームワークを使ってください。

以下の例で、インターフェース名、ソースのポート、IPアドレスを変更して使ってください。

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

この帯域幅制限は以下のコマンドで削除できます。

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

5.8. トラブルシューティング

DRBD proxyのログはsyslogの LOG_DAEMON ファシリティに記録されます。通常ログは /var/log/daemon.log に記録されます。

DRBD Proxyでデバッグモードを有効にするには次のようにします。

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

たとえば、DRBD Proxyが接続に失敗すると、 Rejecting connection because I can’t connect on the other side というようなメッセージがログに記録されます。その場合は、DRBDが(スタンドアローンモードでなく)両方のノードで動作していて、両方ノードでプロキシが動作していることを確認してください。また、両方のノードで設定値を確認してください。

[9] バージョン1では異なるスケジューリングモデルを使用しており、そのためバージョン3と同じようなパフォーマンスを得ることはできません。そのため、もし本番環境がRHEL5の場合には、RHEL6/7の仮想マシンを各データセンターに置いてみるのはいかがでしょうか。

6. トラブルシューティングとエラーからの回復

この章では、ハードウェアやシステムに障害が発生した場合に必要な手順について説明します。

6.1. ハードドライブの障害の場合

ハードドライブの障害への対処方法は、DRBDがディスクI/Oエラーを処理する方法(ディスクエラー処理ストラテジー参照)、また設定されているメタデータの種類(DRBDメタデータ参照)によって異なります。

ほとんどの場合、ここで取り上げる手順はDRBDを直接物理ハードドライブ上で実行している場合にのみ適用されます。次に示す層の上でDRBDを実行している場合には通常は適用されません。

- MDソフトウェアRAIDセット(mdadm を使用してドライブ交換を管理)
- デバイスマッパRAID(dmraid 使用)
- ハードウェアRAID機器 (障害が発生したドライブの扱いについてはベンダーの指示に従ってください)
- 一部の非標準デバイスマッパ仮想ブロックデバイス(デバイスマッパのマニュアルを参照してください)

6.1.1. 手動でDRBDをハードドライブから切り離す

DRBDがI/Oエラーを伝えるように設定(非推奨)している場合、まずDRBDリソースを切り離すとよいでしょう。つまり、下位デバイスからの切り離しです。

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

drbdadm status [10]コマンドを実行することで、現在の状態を確認することができます。

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

ディスク障害がプライマリノードで発生した場合、スイッチオーバーと、この手順を組み合わせることもできます。

6.1.2. I/Oエラー時の自動切り離し

DRBDがI/Oエラー時に自動的に切り離しを行うように設定(推奨オプション)されている場合、 手動での操作なしで、DRBDはすでにリソースを下位ストレージから自動的に切り離しているはずです。その場合でも drbdadm status コマンドを使用して、リソースが実際にディスクレスモードで実行されているか確認します。

6.1.3. 障害が発生したディスクの交換(内部メタデータを使用している場合)

内部メタデータを使用している場合、新しいハードディスクでDRBDデバイスを再構成するだけで十分です。交

換したハードディスクのデバイス名が交換前と異なる場合は、DRBD設定ファイルを適切に変更してください。

新しいメタデータを作成してから、リソースを再接続します。

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

# drbdadm attach <resource>
```

新しいハードディスクの完全同期がただちに自動的に始まります。通常のバックグラウンド同期と同様、同期の進行状況を drbdadm status --verbose で監視することができます。

6.1.4. 障害の発生したディスクの交換(外部メタデータを使用している場合)

外部メタデータを使用している場合でも、手順は基本的に同じです。ただし、DRBDだけではハードドライブが交換されたことを認識できないため、追加の手順が必要です。

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

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

drbdadm invalidate は、必ず破棄するデータのある側のノードでを実行してください。 このコマンドはローカルのデータを対向ノードからのデータで上書きさせます。つまり、このコマンドを実行すると破棄する側のノードのデータが失われます。

上記の drbdadm invalidate コマンドが同期をトリガーします。この場合でも、同期の進行状況は drbdadm status --verbose で確認できます。

6.2. ノード障害に対処する

DRBDが(実際のハードウェア障害であれ手動による介入であれ)対向ノードがダウンしていることを検出すると、DRBDは自身のコネクションステータスを Connected から WFConnection に変更し、対向ノードが再び現れるのを待ちます。その後、DRBDリソースは disconnected mode(切断モード) で動作します。切断モードでは、リソースおよびリソースに関連付けられたブロックデバイスが完全に利用可能で、必要に応じて昇格したり降格したりします。ただし、ブロックの変更は対向ノードにレプリケートされません。切断中に変更されたブロックについての内部情報は対向ノードごとにDRBDが格納します。

6.2.1. 一時的なセカンダリノードの障害に対処する

セカンダリロールでリソースを持っているノードに一時的に障害が生じた場合(たとえばメモリ交換で直るようなメモリの問題)には、障害が発生したノードを修復してオンラインに戻すだけで十分です。修正したノードを起動すると、ノード間の接続が再確立され、停止中にプライマリノードに加えられた変更内容すべてがセカンダ

リノードに同期されます。

この時点で、DRBDの再同期アルゴリズムの性質により、セカンダリノードのリソースの一貫性が一時的に失われます。この短時間の再同期の間は、対向ホストが使用できない場合でも、セカンダリノードをプライマリロールに切り替えることができません。したがって、セカンダリノードの実際のダウンタイムとその後の再同期の間は、クラスタが冗長性を持たない期間になります。

DRBD9では各リソースに3ノード以上が接続することができます。そのため、例えば4ノードでは一度フェイルオーバーしてもまだ2回フェイルオーバーすることができます。

6.2.2. 一時的なプライマリノードの障害に対処する

DRBDからみると、プライマリノードの障害とセカンダリノードの障害はほぼ同じです。生き残ったノードが対向ノードの障害を検出し、切断モードに切り替わります。DRBDは生き残ったノードをプライマリロールに 昇格しません 。これはクラスタ管理システムが管理します。

障害が発生したノードが修復されてクラスタに戻る際に、セカンダリロールになります。すでに述べたように、それ以上の手動による介入は必要ありません。このときもDRBDはリソースのロールを元に戻しません。変更を行うように設定されている場合は、クラス管理システムがこの変更を行います。

プライマリノードに障害が発生すると、DRBDはアクティビティログというメカニズムによってブロックデバイスの整合性を確保します。詳細はアクティビティログを参照ください。

6.2.3. 永続的なノード障害に対処する

ノードに回復不能な問題が発生した場合やノードが永久的に破損した場合は、次の手順を行う必要があります。

- 障害が発生したハードウェアを同様のパフォーマンスと ディスク容量を持つハードウェアと交換します。

障害が発生したノードを、それよりパフォーマンスが低いものと置き換えることも可能ですが、お勧めはできません。障害が発生したノードを、それよりディスク容量が小さいものと置き換えることはできません。このような場合には、DRBDを置き換えたノードへの接続は拒否されます。 [11]

- OSとアプリケーションをインストールします。
- DRBDをインストールし、生き残ったノードから /etc/drbd.conf と、全ての /etc/drbd.d/ を コピーします。
- DRBDの設定に記載された手順を、 デバイスの初期同期の前まで実行します。

この時点で、デバイスの完全同期を手動で開始する必要はありません。生き残ったプライマリノードへの接続時に、同期が自動的に開始します。

6.3. スプリットブレインからの手動回復

ノード間の接続が可能になると、ノード間で初期ハンドシェイクのプロトコルが交換されます。この時点でDRBDはスプリットブレインが発生したかどうかを判断できます。両方のノードがプライマリロールであるか、もしくは切断中に両方がプライマリロールになったことを検出すると、DRBDは即座にレプリケーション接続を切断します。その場合、システムログにたとえば次のようなメッセージが記録されます。

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

スプリットブレインが検出されると、1つのノードは常に StandAlone の状態でリソースを保持します。もう一方のノードもまた StandAlone 状態になる(両方のノードが同時にスプリットブレインを検出した場合)、または WFConnection 状態になります(一方のノードがスプリットブレインを検出する前に対向ノードが切断をした場合)。

DRBDがスプリットブレインから自動的に回復するように設定されていない場合は、この時点で手動で介入して、変更内容を破棄するほうのノードを選択する必要があります(このノードは スプリットブレインの犠牲ノード と呼ばれる)。 この介入は次のコマンドで行います。

ここは現在 作業中 です。

構成の変更が行われる予定です。

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

他方のノード(スプリットブレインの 生存ノード)のコネクションステータスも StandAlone の場合には、次のコマンド実行します。

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

ノードがすでに Connecting の状態の場合は自動的に再接続するので、この手順は省略できます。

接続すると、スプリットブレインの犠牲ノードのコネクションステータスがすぐに SyncTarget に変化し、他のノードによって変更内容が上書きされます。

スプリットブレインの犠牲ノードは、デバイスのフル同期の対象にはなりません。代わりに、ローカル側での変更がロールバックされ、スプリットブレインの生存ノードに対して加えられた変更が犠牲ノードに伝播されます。

再同期が完了すると、スプリットブレインが解決したとみなされ、2つのノードが再び完全に一致した冗長レプリケーションストレージシステムとして機能します。

[10] 以前は drbdadm dstate コマンドを推奨していました
[11] 結局はレプリケーションできません

DRBDとアプリケーションの組み合わせ

7. DRBDとPacemakerクラスタ

PacemakerクラスタスタックとDRBDの組み合わせは、もっとも多いDRBDの使いみちです。そしてまた、Pacemakerはさまざまな使用シナリオでDRBDを非常に強力なものにするアプリケーションの１つです。

DRBDはPacemakerクラスタで2つの使用法があります。 * DRBDをSANのようにバックグラウンドのサービスとして使用する、または * DRBDをPacemakerで完全に制御する

それぞれの長所と短所は後で考察します。

フェンシング設定を行うことを推奨します。 クラスタ間の接続が途切れてノードが互いを見失うと、両ノードでサービスが開始し(フェイルオーバー)、通信が復帰したタイミングで **スプリットブレイン** が発生する事になります。

7.1. Pacemakerの基礎

PacemakerはLinuxプラットフォーム向けの高度で機能豊富なクラスタリソースマネージャで、さまざまな用途で使われています。マニュアルも豊富に用意されています。この章を理解するために、以下のドキュメントを読むことを強くお勧めします。

- Clusters From Scratch:高可用性クラスタ構築ステップバイステップガイド
- CRM CLI (command line interface) tool:CRMのシェルやシンプルかつ直感的なコマンドラインインタフェースのマニュアル
- Pacemaker Configuration Explained:Pacemakerの背景にあるコンセプトや設計を説明している参考ドキュメント

7.2. DRBDをPacemakerクラスタで管理する

自律的なDRBDストレージはローカルストレージのように扱えます。Pacemakerクラスタへの組み込みは、DRBDのマウントポイントを指定することで行えます。

初めに、DRBDの 自動プロモーション 機能を使用しますので、DRBDは必要な時に自分自身を プライマリ に設定します。この動作はすべてのリソースで同じなので common セクションで設定しておくとよいでしょう。

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

後はファイルシステム経由で使ってストレージにアクセスするだけです。

Listing 4. 自動プロモーション を使ったDRBDを下位デバイスにするMySQLサービスのPacemaker設定

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

基本的には、必要なのはDRBDリソースがマウントされるマウントポイント(この例では /var/lib/mysql)です。

Pacemaker管理下ではクラスタ内で1インスタンスのみがマウント出来ます。

7.3. DRBDを下位デバイスにするマスターとスレーブのあるサービスのクラスタ設定

このセクションではPacemakerクラスタでDRBDを下位デバイスにするサービスを使用する方法を説明します。

DRBDのOCFリソースエージェントを採用している場合には、DRBDの起動、停止、昇格、降格はOCFリソースエージェント のみ で行う事を推奨します。DRBDの自動起動は無効にしてください。

```
chkconfig drbd off
```

OCFリソースエージェントの ocf:linbit:drbd はマスター/スレーブ管理が可能であり、Pacemakerから複数ノードでのDRBDリソースの起動やモニター、必要な場合には降格を行うことができます。ただし、 drbd リソースエージェントはPacemakerのシャットダウン時、またはノードをスタンバイモードにする時には、管理する全DRBDリソースを切断してデタッチを行う事を理解しておく必要があります。

linbit のDRBD付属のOCFリソースエージェントは /usr/lib/ocf/resource.d/linbit/drbd にインストールされます。heartbeat のOCFリソースパッケージに付属するレガシーなリソースエージェントは /usr/lib/ocf/resource.d/heartbeat/drbd にインストールされます。レガシーなOCFリソースエージェントは非推奨なので使用しないでください。

drbd のOCFリソースエージェントを使用したPacemakerのCRMクラスタで、MySQLデータベースにDRBDの下位デバイス設定を有効にするためには、両方の必要なリソースエージェントを作成して、 先行してDRBDリソースが昇格したノードでだけサービスが起動するようにPacemakerの制約を設定しなければなりません。以下に示すように、crm シェルで設定することができます。

Listing 5. マスター/スレーブ リソースを使ったDRBDを下位デバイスにするMySQLサービスのPacemaker設定

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

これで設定が有効になります。そしてPacemakerがDRBDリソースを昇格するノードを選び、そのノードでDRBDを下位デバイスにするリソースを起動します。

7.4. Pacemakerクラスタでリソースレベルのフェンシングを使用する

ここでは、DRBDのレプリケーションリンクが遮断された場合に、Pacemakerが drbd マスター/スレーブリソースを昇格させないようにするために必要な手順の概要を説明します。これにより、Pacemakerが古いデータでサービスを開始し、プロセスでの不要な「タイムワープ」の原因になることが回避できます。

DRBD用のリソースレベルのフェンシングを有効にするには、リソース設定で次の行を追加する必要があります。

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

同時に、使用するクラスタインフラストラクチャによっては handlers セクションも変更しなければなりません。

- Heartbeatを使用したPacemakerクラスタは dopd でのリソースレベルフェンシングで説明している設定を使用できます。
- CorosyncとHeartbeatを使用したクラスタは CIB (Cluster Information Base)を使ったリソースレベルフェンシングで説明されている機能を使うことができます。

最低でも2つの独立したクラスタ通信チャネルを設定しなければ、この機能は正しく動作しません。HeartbeatベースのPacemakerクラスタでは ha.cf 設定ファイルに最低2つのクラスタ通信のリンクを定義する必要があります。Corosyncクラスタでは最低2つの冗長リングを corosync.conf に記載しなければなりません。

7.4.1. dopd でのリソースレベルフェンシング

Heartbeatを使用したPacemakerクラスタでは、DRBDは DRBD outdate-peer daemon 、または略して dopd と呼ばれるリソースレベルのフェンシング機能を使用できます。

dopd 用のHeartbeat設定

dopdを有効にするには、次の行を /etc/ha.d/ha.cf ファイルに追加します。

```
respawn hacluster /usr/lib/heartbeat/dopd
apiauth dopd gid=haclient uid=hacluster
```

使用するディストリビューションに応じて dopd のパスを調整する必要があります。一部のディストリビューションとアーキテクチャでは、正しいパスが /usr/lib64/heartbeat/dopd になります。

変更を行い ha.cf を対向ノードにコピーをしたら、設定ファイルを読み込むためにPacemakerをメンテナンスモードにして '/etc/init.d/heartbeat reload' を実行します。その後、 dopd プロセスが動作していることを確認できるでしょう。

このプロセスを確認するには、 ps ax | grep dopd を実行するか、 killall -0 dopd を使用します。

dopd 用のDRBD設定

dopd が起動したら、DRBDリソース設定にアイテムを追加します。

```
resource <resource> {
  handlers {
    fence-peer "/usr/lib/heartbeat/drbd-peer-outdater -t 5";
    ...
  }
  net {
    fencing resource-only;
    ...
  }
  ...
}
```

dopd と同様に、システムのアーキテクチャやディストリビューションによっては drbd-peer-outdater バイナリは /usr/lib64/heartbeat に配置されます。

最後に、drbd.conf を対向ノードにコピーし、 drbdadm adjust resource を実行して、リソースを再構成し、変更内容を反映します。

dopd 機能のテスト

設定した dopd が正しく動作しているか確認するためには、Heartbeatサービスが正常に動作しているときに、構成済みの接続されているリソースのレプリケーションリンクを遮断します。ネットワークリンクを物理的に取り外すことで簡単にできますが、少々強引ではあります。あるいは、一時的に iptables ルールを追加して、DRBD用TCPトラフィックを遮断します。

すると、リソースのコネクションステータスが Connected から Connecting に変わります。数秒後にディスク状態が Outdated/DUnknown に変化します。これで dopd が機能していることを確認できます。

これ以降は、古いリソースをプライマリロールに切り替えようとしても失敗します。

物理リンクを接続するか、一時的な iptables ルールを削除してネットワーク接続を再確立すると、コネクションステータスが Connected に変化し、すぐに SyncTarget になります(ネットワーク遮断中に、プライマリノードで変化が起こった場合)。同期が終了すると、無効状態であったリソースに再度 UpToDate のマークが付きます。

7.4.2. CIB (Cluster Information Base)を使ったリソースレベルフェンシング

Pacemaker用のリソースレベルフェンシングを有効にするには、 drbd.conf の2つのオプション設定をする必要があります。

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

DRBDレプリケーションリンクが切断された場合には crm-fence-peer.sh スクリプトがクラスタ管理システムに連絡し、このDRBDリソースに関連付けられたPacemakerのマスター/スレーブリソースが決定され、現在アクティブなノード以外のすべてのノードでマスター/スレーブリソースが昇格されることがないようにします。逆に、接続が再確立してDRBDが同期プロセスが完了すると、この制約は解除され、クラスタ管理システムは再び任意のノードのリソースを自由に昇格させることができます。

7.5. PacemakerクラスタでスタックDRBDリソースを使用する

DRBD9.xでは1つの階層で複数ノードを使用できるのでスタッキングは非推奨です。詳細はネットワークコネクションの定義をご参照ください。

スタックリソースを用いるとマルチノードクラスタの多重冗長性やオフサイトのディザスタリカバリ機能を実現するためにDRBDを利用できます。本セクションではそのような構成におけるDRBDおよびPacemakerの設定方法について説明します。

7.5.1. オフサイトディザスタリカバリ機能をPacemakerクラスタに追加する

この構成シナリオでは、1つのサイトの2ノードの高可用性クラスタと、多くは別のサイトに設置する独立した1つのノードについて説明します。第3のノードは、ディザスタリカバリノードとして機能するスタンドアロンサーバです。次の図で概念を説明します。

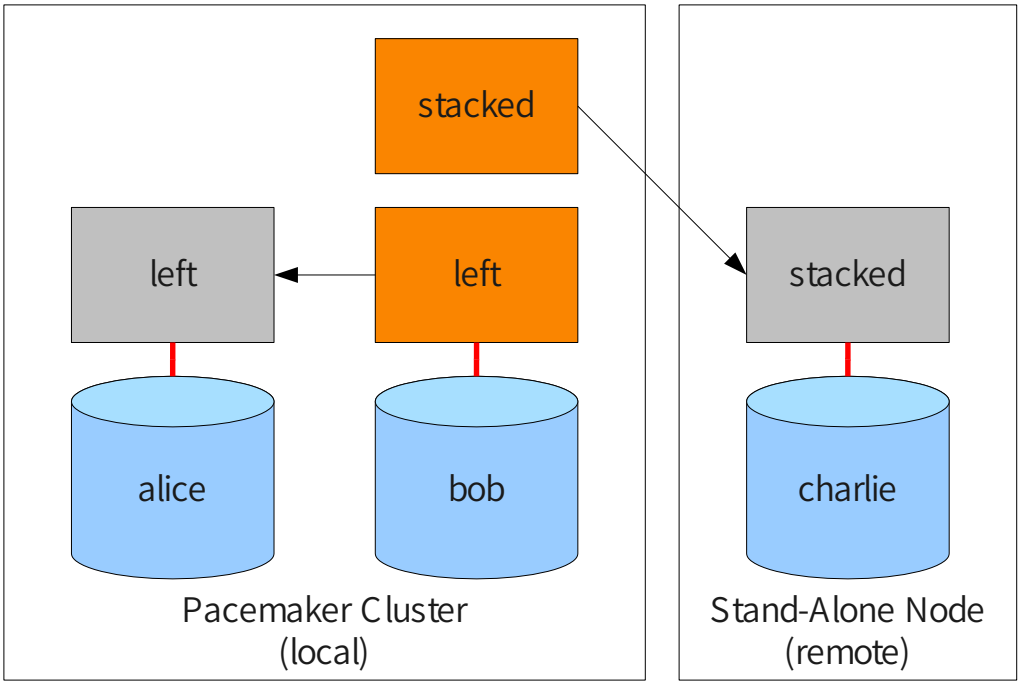


図 10. PacemakerクラスタのDRBDリソースのスタック

この例では alice と bob が2ノードのPacemakerクラスタを構成し、 charlie はPacemakerで管理されないオフサイトのノードです。

このような構成を作成するには、スタック3ノード構成の作成の説明に従って、まずDRBDリソースを設定と初期化を行います。そして、次のCRM構成でPacemakerを設定します。

```
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
```

この構成を /tmp/crm.txt という一時ファイルに保存し、次のコマンドで現在のクラスタにインポートします。

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

この設定により、次の操作が正しい順序で alice と bob クラスタで行われます。

1. PacemakerはDRBDリソース r0 を両クラスタノードで開始し、1つのノードをマスター(DRBD Primary)ロールに昇格させます。
2. PacemakerはIPアドレス192.168.42.1の、第3ノードへのレプリケーションに使用するスタックリソースを開始します。これは、 r0 DRBDリソースのマスターロールに昇格したノードで行われます。
3. r0 がプライマリになっていて、かつ、r0-U のレプリケーション用IPアドレスを持つノードで、Pacemakerはオフサイトノードに接続およびレプリケートを行う r0-U DRBDリソースを開始します。
4. 最後に、Pacemakerが r0-U リソースもプライマリロールに昇格させるため、ディザスタリカバリノードとのレプリケーションが始まります。

このように、このPacemaker構成では、クラスタノード間だけではなく第3のオフサイトノードでも完全なデータ冗長性が確保されます。

このタイプの設定には通常、DRBD Proxyと合わせて使用するのが一般的です。

7.5.2. スタックリソースを使って、Pacemakerクラスタの4ノード冗長化を実現する

この構成では、全部で3つのDRBDリソース(2つの非スタック、1つのスタック)を使って、4ノードのストレージ冗長化を実現します。4ノードクラスタの目的と意義は、3ノードまで障害が発生しても、可用なサービスを提供し続けることが可能であることです。

次の例で概念を説明します。

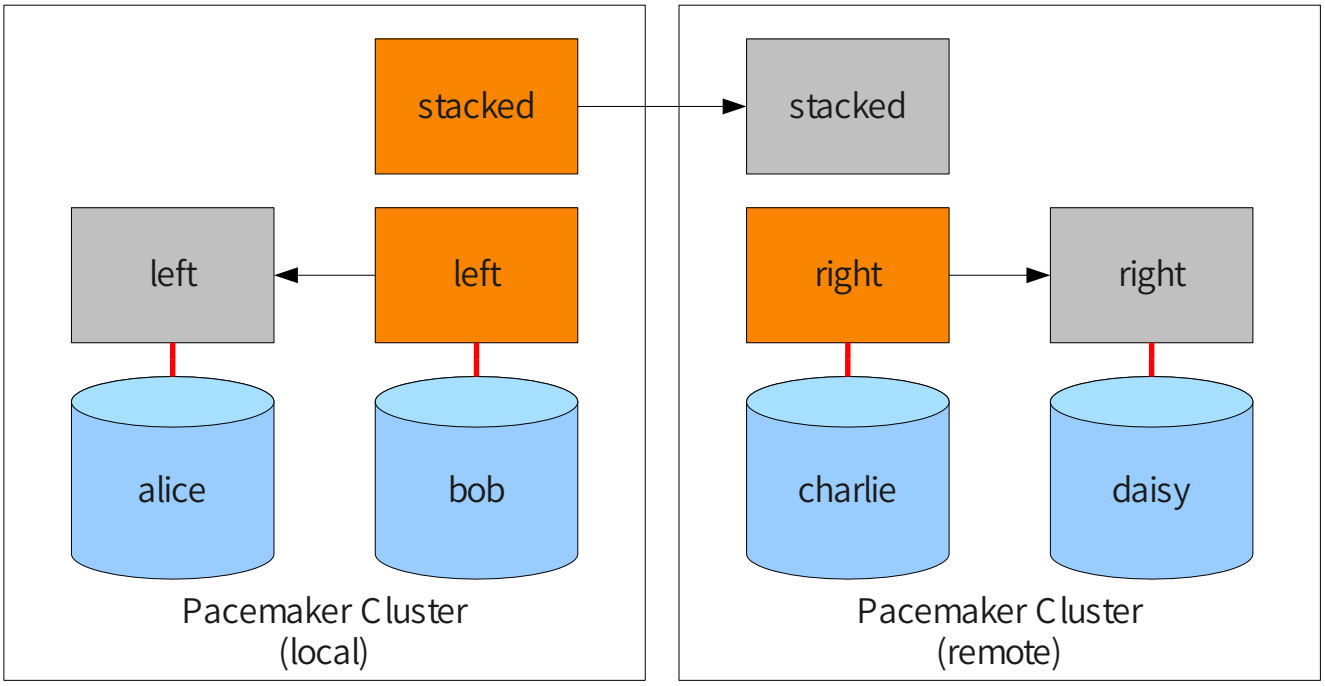


図 11. PacemakerクラスタのDRBDリソースのスタック

この例では、 alice と bob ならびに charlie と daisy が2セットの2ノードPacemakerクラスタを構成しています。alice と bob は left という名前のクラスタを構成し、互いにDRBDリソースを使ってデータをレプリケートします。一方 charlie と daisy も同様に、 right という名前の別のDRBDリソースでレプリケートします。3番目に、DRBDリソースをスタックし、2つのクラスタを接続します。

Pacemakerバージョン1.0.5のPacemakerクラスタの制限により、CIBバリデーションを有効にしたままで4ノードクラスタをつくることはできません。CIBバリデーションは汎用的に使うのには向かない特殊な高度な処理です。これは、今後のPacemakerのリリースで解決されることが予想されます。

このような構成を作成するには、スタック3ノード構成の作成の説明に従って、まずDRBDリソースを設定して初期化します(ただし、ローカルがクラスタになるだけでなく、リモート側にもクラスタになる点が異なります)。そして、次のCRM構成でPacemakerを設定し、 left クラスタを開始します。

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

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

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

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

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

colocation c_ip_on_left_master \
    inf: p_ip_stacked_left ms_drbd_left:Master

colocation c_drbd_stacked_on_ip_left \
    inf: ms_drbd_stacked p_ip_stacked_left

order o_ip_before_stacked_left \
    inf: p_ip_stacked_left ms_drbd_stacked:start

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

この構成を /tmp/crm.txt という一時ファイルに保存し、次のコマンドで現在のクラスタにインポートします。

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

CIBに上記の設定を投入すると、Pacemakerは以下のアクションを実行します。

1. alice と bob をレプリケートするリソース left を起動し、いずれかのノードをマスターに昇格します。
2. IPアドレス10.9.9.100 (alice または bob 、いずれかのリソース left のマスターロールを担っている方)を起動します。
3. IPアドレスを設定したのと同じノード上で、DRBDリソース stacked が起動します。
4. target-role="Master"が指定されているため、スタックリソースがプライマリになります。

さて、以下の設定を作り、クラスタ right に進みましょう。

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

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

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

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

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

colocation c_drbd_stacked_on_ip_right \
    inf: ms_drbd_stacked p_ip_stacked_right

colocation c_ip_on_right_master \
    inf: p_ip_stacked_right ms_drbd_right:Master

order o_ip_before_stacked_right \
    inf: p_ip_stacked_right ms_drbd_stacked:start

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

CIBに上記の設定を投入すると、Pacemakerは以下のアクションを実行します。

1. charlie と daisy 間をレプリケートするDRBDリソース right を起動し、これらのノードのいずれかをマスターにします。
2. IPアドレス10.9.10.101を開始します(charlie または daisy のいずれかのリソース right のマスターロールを担っている方)を起動します)。
3. IPアドレスを設定したのと同じノード上で、DRBDリソース stacked が起動します。
4. target-role="Slave" が指定されているため、スタックリソースはセカンダリのままになります。

7.6. 2セットのSANベースPacemakerクラスタ間をDRBDでレプリケート

これは、拠点が離れた構成に用いるやや高度な設定です。2つのクラスタが関与しますが、それぞれのクラスタは別々のSANストレージにアクセスします。サイト間のIPネットワークを使って2つのSANストレージのデータを同期させるためにDRBDを使います。

次の図で概念を説明します。

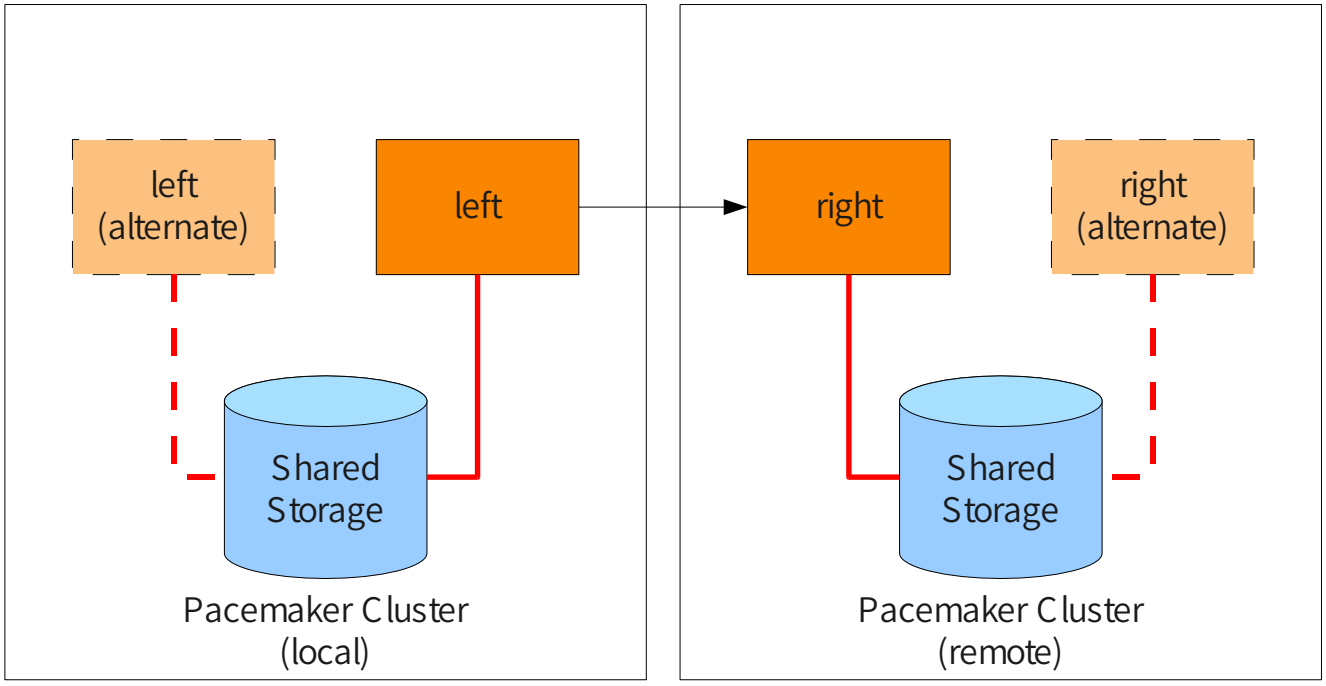


図 12. SANベースのクラスタ間のレプリケートにDRBDを用いる

このような構成の場合、DRBDの通信にかかわるホストをあらかじめ明示的に指定しておくことは不可能です。つまり、動的接続構成の場合、DRBDは特定の物理マシンではなく仮想IPアドレスで通信先を決めます。

このタイプの設定は通常、DRBD Proxyや、またはトラック輸送のレプリケーションと組み合わせます。

このタイプの設定は共有ストレージを扱うので、STONITHを構築してテストすることが、正常動作の確認のためには必要不可欠です。ただし、STONITHは本書の範囲を超えるので、以下の設定例は省略しています。

7.6.1. DRBDリソース構成

動的接続するDRBDリソースを有効にするには、次のように drbd.conf を設定します。

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

floating キーワードは、通常リソース設定の on <host> セクションの代わりに指定します。このモードでは、DRBDはホスト名ではなく、IPアドレスやTCPポートで相互接続を認識します。動的接続を適切に運用するには、物理IPアドレスではなく仮想IPアドレスを指定してください(これはとても重要です)。上の例のように、離れた地点ではそれぞれ別々のIPネットワークに属するのが一般的です。したがって、動的接続を正しく運用するにはDRBDの設定だけではなく、ルータやファイアウォールの適切な設定も重要です。

7.6.2. Pacemakerリソース構成

DRBD動的接続の設定には、少なくともPacemaker設定が必要です(2つの各Pacemakerクラスタに係わる)。

- 仮想クラスタIPアドレス
- マスター/スレーブDRBDリソース(DRBD OCFリソースエージェントを使用)
- リソースを適切なノードで正しい順序に起動するための各種制約

レプリケーション用アドレスに 10.9.9.100 を使う動的接続構成と、 mysql というリソースを構築するには、次のように crm コマンドでPacemakerを設定します。

```
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
```

CIBに上記の設定を投入すると、Pacemakerは以下のアクションを実行します。

1. IPアドレス10.9.9.100を起動する(alice または bob のいずれか)
2. IPアドレスの設定にもとづいてDRBDリソースを起動します。
3. DRBDリソースをプライマリにします。

次に、もう一方のクラスタで、これとマッチングする設定を作成します。 その Pacemakerのインスタンスを次のコマンドで設定します。

```
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
```

CIBに上記の設定を投入すると、Pacemakerは以下のアクションを実行します。

1. IPアドレス10.9.10.101を起動する(charlie または daisy のいずれか)。
2. IPアドレスの設定にもとづいてDRBDリソースを起動します。

3. target-role="Slave" が指定されているため、DRBDリソースは、セカンダリのままになります。

7.6.3. サイトのフェイルオーバ

拠点が離れた構成では、サービス自体をある拠点から他の拠点に切り替える必要が生じるかもしれません。これは、計画された移行か、または悲惨な出来事の結果でしょう。計画にもとづく移行の場合、一般的な手順は次のようになります。

- サービス移行元のクラスタに接続し、影響を受けるDRBDリソースの target-role 属性を マスター から スレーブ に変更します。DRBDがプライマリであることに依存したリソースは自動的に停止し、その後DRBDはセカンダリに降格します。
- サービス移行先のクラスタに接続し、DRBDリソースの target-role 属性を Slave から Master に変更します。DRBDリソースは昇格し、DRBDリソースのプライマリ側に依存した他のPacemakerリソースを起動します。 そしてリモート拠点への同期が更新されます。
- フェイルバックをするには、手順を逆にするだけです。

アクティブな拠点で壊滅的な災害が起きると、その拠点はオフラインになり、以降その拠点からバックアップ側にレプリケートできなくなる可能性があります。このような場合には

- リモートサイトが機能しているならそのクラスタに接続し、DRBDリソースの target-role 属性を スレーブ から マスター に変更します。DRBDリソースは昇格し、DRBDリソースがプライマリになることに依存する他のPacemakerリソースも起動します。
- 元の拠点が回復または再構成されると、DRBDリソースに再度接続できるようになります。その後、逆の手順でフェイルバックします。

8. DRBDとRed Hat Cluster Suite

この章ではRDBDをRed Hat Cluster高可用性クラスタのためのレプリケーションストレージとして使用する方法を説明します。

ここでは非公式な Red Hat Cluster という用語を歴代の複数の正式な製品名として使用しており、そのなかには Red Hat Cluster Suite と Red Hat Enterprise Linux High Availability Add-On が含まれています。

8.1. Red Hat Clusterの背景情報

8.1.1. フェンシング

Red Hat Clusterは本来は共有ストレージクラスタを主な対象として設計されたもので、ノードフェンシングによって共有リソースへの危険な同時アクセスを回避します。Red Hat Cluster Suiteのフェンシングインフラストラクチャは、フェンシングデーモン fenced とシェルスクリプトとして実装されるフェンシングエージェントに依存しています。

DRBDベースのクラスタは共有ストレージリソースを利用しないため、DRBDとしては厳密にはフェンシングは必要ありません。 ただし、Red Hat Cluster SuiteはDRBDベースの構成の場合でもフェンシングを必要とします。

8.1.2. リソースグループマネージャ

リソースグループマネージャ(rgmanager または clurgmgr)はPacemakerと似ています。これは、クラスタ管理スイートと管理対象アプリケーションとの間の主要なインタフェースとして機能します。

Red Hat Clusterリソース

Red Hat Clusterでは、個々の高可用性アプリケーション、ファイルシステム、IPアドレスなどを リソース と呼びます。

たとえば、NFSエクスポートがマウントされているファイルシステムに依存するように、リソースは互いに依存しています。リソースは別のリソース内に入れ子になって、 リソースツリー を構成します。入れ子の内部のリソースが、入れ子の外部のリソースからパラメータを継承する場合もあります。Pacemakerにはリソースツリーの概念はありません。

Red Hat Clusterサービス

相互に依存するリソースの集合を サービス と呼びます。Pacemakerではこのような集合を リソースグループ と呼んでいます。

rgmanagerリソースエージェント

rgmanager により呼び出されるリソースエージェントは、Pacemakerで使用されるものと同様に、Open Cluster Framework (OCF)で定義されたシェルベースのAPIを使用しますが、Pacemakerはフレームワークで定義されていない拡張機能も利用します。このように理論的には、Red Hat Cluster SuiteとPacemakerのリソースエージェントはおおむね互換性がありますが、実際にはこの2つのクラスタ管理スイートは似たようなタスクや同一のタスクに異なるリソースエージェントを使用します。

Red Hat Clusterリソースエージェントは /usr/share/cluster ディレクトリにインストールされます。オールインワン型のPacemaker OCFリソースエージェントとは異なり、一部の Red Hat Clusterリソースエージェントは、実際のシェルコードを含む .sh ファイルと、XML形式のリソースエージェントメタデータを含む、 .metadata フ

ァイルに分割されています。

DRBDは Red Hat Clusterリソースエージェントも提供しています。これは通常通りのディレクトリに drbd.sh および drbd.metadata としてインストールされています。

8.2. Red Hat Clusterの設定

ここでは、Red Hat Clusterを実行するために必要な設定手順の概要を説明します。クラスタ構成の準備は比較的容易で、すべてのDRBDベースのRed Hat Clusterに必要なのは、2つの参加ノード(Red Hatのドキュメントでは クラスタメンバ と呼ばれる)とフェンシングデバイスだけです。

Red Hat clusterの設定詳細は、http://www.redhat.com/docs/manuals/csgfs/[Red Hat ClusterとGFS(Global File System)についてのRed Hatのマニュアル]をご参照ください。

8.2.1. cluster.conf ファイル

RHELクラスタの設定は単一の設定ファイル　/etc/cluster/cluster.conf に記載されています。次のような方法でクラスタ構成を管理できます。

設定ファイルを直接編集する
これがもっとも簡単な方法です。テキストエディタ以外に必要なものはありません。

system-config-cluster GUIを使用する
Gladeを使用してPythonで記述したGUIアプリケーションです。Xディスプレイ(直接サーバコンソール上、またはSSH経由のトンネル)が必要です。

Conga webベース管理インフラストラクチャを使用する
Congaインフラストラクチャは、ローカルクラスタ管理システムと通信するノードエージェント(ricci)、クラスタリソースマネージャ、クラスタLVMデーモン、および管理用Webアプリケーション(luci)から構成されます。luciを使用して、シンプルなWebブラウザでクラスタインフラストラクチャを構成することができます。

8.3. Red Hat ClusterフェイルオーバクラスタでDRBDを使用する

ここでは、GFSについては取り上げず、Red Hat Clusterフェイルオーバクラスタ用のDRBDの設定方法のみを取り上げます。GFS (およびGFS2)の設定については、DRBDでGFSを使用するを参照してください。

このセクションでは、DRBDとPacemakerクラスタの同様のセクションのように、高可用性MySQLデータベースを構成することを前提としています。次のような構成パラメータを使用します。

- データベースのストレージ領域として使用するDRBDリソース名は mysql で、これによりデバイス /dev/drbd0 を管理する。
- DRBDデバイスにext3ファイルシステムが格納され、これが /var/lib/mysql(デフォルトのMySQLデータディレクトリ)にマウントされる。
- MySQLデータベースがこのファイルシステムを利用し、 専用クラスタIPアドレス192.168.42.1で待機する。

8.3.1. クラスタ構成の設定

高可用性MySQLデータベースを設定するには、/etc/cluster/cluster.conf ファイルを作成するか変更して、次の構成項目を記述します。

まず /etc/cluster/cluster.conf を適切なテキストエディタで開き、リソース設定で次の項目を記述します。

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

この例ではボリュームリソースが1つの場合を前提にしています。

<service/> でリソース参照を相互に入れ子にするのは、Red Hat Cluster Suiteでリソースの依存関係を記述する方法です。

設定が完了したら、必ず、 <cluster> 要素の config_version 属性をインクリメントしてください。次のコマンドを実行して、実行中のクラスタ構成に変更内容をコミットします。

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

必ず、2番目のコマンドの <version> を新しいクラスタ設定のバージョン番号と置き換えてください。

cluster.conf ファイルに drbd リソースエージェントを含めると、 system-config-cluster GUI設定ユーティリティとConga Webベースクラスタ管理インフラストラクチャの両方が、クラスタ設定に関する問題についてのメッセージを返します。これは、2つのアプリケーションが提供するPythonクラスタ管理ラッパーが、クラスタインフラストラクチャに他社製の拡張機能を使用することを前提としていないためです。

したがって、クラスタ設定に drbd リソースエージェントを使用する場合は、クラスタ構成のために system-config-cluster またはCongaを使用することはお勧めできません。これらのツールは正しく機能するはずですが、クラスタの状態を監視するためにのみ使用してください。

9. DRBDでのLVMの使用

この章では、DRBDの管理とLVM2について説明します。特に、次のような項目を取り上げます。

- LVM論理ボリュームをDRBDの下位デバイスとして使用する。
- DRBDデバイスをLVMの論理ボリュームとして使用する。
- 上記の2つを組み合わせ、 DRBDを使用して階層構造のLVMを実装する。

上記の用語についてよく知らない場合は、LVMの基礎を参照してください。また、ここで説明する内容にとどまらず、LVMについてさらに詳しく理解することをお勧めします。

9.1. LVMの基礎

LVM2は、Linuxデバイスマッパフレームワークでの論理ボリューム管理を実用化したものです。元々のLVMとは、名前以外は実際は何の共通点もありません。以前の実装(現在LVM1と呼ぶもの)は時代遅れとみなされているため、ここでは取り上げません。

LVMを使用する際には、次に示す基本的なコンセプトを理解することが重要です。

物理ボリューム (PV)

PV (Physical Volume: 物理ボリューム)はLVMのみが管理配下とするブロックデバイスです。ハードディスク全体または個々のパーティションがPVになります。ハードディスクにパーティションを1つだけ作成して、これをすべてLinux LVMで使用するのが一般的です。

パーティションタイプ"Linux LVM" (シグネチャは 0x8E)を使用して、LVMが独占的に使用するパーティションを識別できます。ただし、これは必須ではありません。PVの初期化時にデバイスに書き込まれるシグネチャによってLVMがPVを認識します。

ボリュームグループ (VG)

VG (Volume Group: ボリュームグループ)はLVMの基本的な管理単位です。VGは1つまたは複数のPVで構成されます。各VGは一意の名前を持ちます。実行時にPVを追加したり、PVを拡張して、VGを大きくすることができます。

論理ボリューム (LV)

実行時にVG内にLV (Logical Volume: 論理ボリューム)を作成することにより、カーネルの他の部分がこれらを通常のブロックデバイスとして使用できます。このように、LVはファイルシステムの格納をはじめ、ブロックデバイスと同様のさまざまな用途に使用できます。オンライン時にLVのサイズを変更したり、1つのPVから別のPVに移動したりすることができます(PVが同じVGに含まれる場合)。

スナップショット論理ボリューム (SLV)

スナップショットはLVの一時的なポイントインタイムコピーです。スナップショットはLVの一時的なポイントインタイムコピーです。元のLV(コピー元ボリューム)のサイズが数百GBの場合でも、スナップショットは一瞬で作成できます。通常、スナップショットは元のLVと比べてかなり小さい領域しか必要としません。

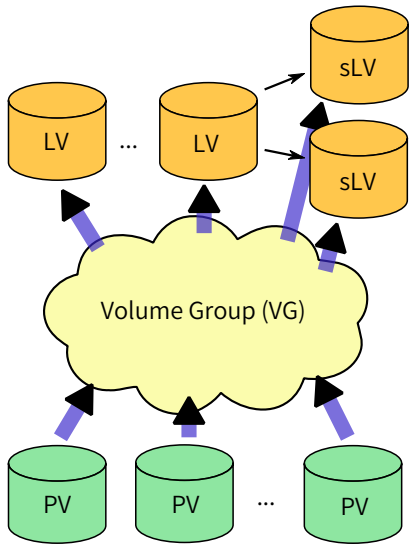


図 13. LVM概要

9.2. DRBDの下位デバイスとして論理ボリュームを使用する

Linuxでは、既存の論理ボリュームは単なるブロックデバイスであるため、これをDRBDの下位デバイスとして使用できます。この方法でLVを使用する場合は、通常通りにLVを作成してDRBD用に初期化するだけです。

次の例では、LVM対応システムの両方のノードに foo というボリュームグループがすでに存在します。このボリュームグループの論理ボリュームを使用して、 r0 というDRBDリソースを作成します。

まず、次のコマンドで論理ボリュームを作成します。

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

このコマンドはDRBDクラスタの両方のノードで実行する必要があります。これで、両ノードに /dev/foo/bar というブロックデバイスが作成されます。

次に、新しく作成したボリュームをリソース設定に加えます。

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

これでリソースを起動できます。手順はLVM対応以外のブロックデバイスと同様です。

9.3. DRBD同期中の自動LVMスナップショットの使用

DRBDが同期をしている間、 SyncTarget の状態は同期が完了するまでは Inconsistent (不整合)の状態です。この状況では SyncSource で(修復できない)障害があった場合に困った状況になります。正常なデータを持つノードが死に、誤った情報を持つノードが残ってしまいます。

LVM論理ボリュームをDRBDに渡す際には、同期の開始時に自動スナップショットを作成し、またこれを完了後に自動削除する方法によって、この問題を軽減することができます

再同期中に自動でスナップショットをするには、リソース設定に以下の行を追加します。

Listing 6. DRBD同期前の自動スナップショット作成

```
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";
  }
}
```

2つのスクリプトはDRBDが呼び出した ハンドラ に自動で渡す $DRBD_RESOURCE$ 環境変数を解析します。snapshot-resync-target-lvm.sh スクリプトは、同期の開始前に、リソースが含んでいるボリュームのLVMスナップショットを作成します。そのスクリプトが失敗した場合、同期は 開始 されません。

同期が完了すると、 unsnapshot-resync-target-lvm.sh スクリプトが必要のなくなったスナップショットを削除します。スナップショットの削除に失敗した場合、スナップショットは残り続けます。

できる限り不要なスナップショットは確認するようにしてください。スナップショットが満杯だと、スナップショット自身と元のボリュームの障害の原因になります。

SyncSource に修復できない障害が起きて、最新のスナップショットに復帰したいときには、いつでも lvconvert -M コマンドで行えます。

9.4. DRBDリソースを物理ボリュームとして構成する

DRBDリソースを物理ボリュームとして使用するためには、DRBDデバイスにPVのシグネチャを作成する必要があります。リソースが現在プライマリロールになっているノードで、 次のいずれかのコマンドを実行します。

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

または

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

この例ではボリュームリソースが1つの場合を前提にしています。

次に、LVMがPVシグネチャをスキャンするデバイスのリストにこのデバイスを加えます。このためには、通常は /etc/lvm/lvm.conf という名前のLVM設定ファイルを編集する必要があります。 devices セクションで filter というキーワードを含む行を見つけて編集します。すべて のPVをDRBDデバイスに格納する場合には、次ように filter オプションを編集します。

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

このフィルタ表現がDRBDデバイスで見つかったPVシグネチャを受け入れ、それ以外のすべてを拒否(無視)します。

デフォルトではLVMは /dev にあるすべてのブロックデバイスのPVシグネチャをスキャンします。これは filter = ["a|.*|"] に相当します。

LVMのPVでスタックリソースを使用する場合は、より明示的にフィルタ構成を指定する必要があります。対応する下位レベルリソースや下位デバイスでPVシグネチャを無視している場合、LVMはスタックリソースでPVシグネチャを検出する必要があります。次の例では、下位レベルDRBDリソースは0から9のデバイスを使用し、スタックリソースは10以上のデバイスを使用しています。

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

このフィルタ表現は、 /dev/drbd10 から /dev/drbd19 までのDRBDのデバイスで見つかったPVシグニチャを受け入れ、それ以外のすべてを拒否(無視)します。

lvm.conf ファイルを変更したら　vgscan コマンドを実行します。LVMは構成キャッシュを破棄し、デバイスを再スキャンしてPVシグネチャを見つけます。

システム構成に合わせて、別の filter 構成を使用することもできます。重要なのは、次の2つです。

- PVとして使用するデバイスに、DRBDのデバイスを許容する
- 対応する下位レベルデバイスを拒否(除外)して、 LVMが重複したPVシグネチャを見つけることを回避する。

さらに、次の設定で、LVMのキャッシュを無効にする必要があります。

```
write_cache_state = 0
```

LVMのキャッシュを無効にした後、 /etc/lvm/cache/.cache を削除して、古いキャッシュを削除してください。

対向ノードでも、上記の手順を繰り返します。

システムのルートファイルシステムがLVM上にある場合、bootの際にボリュームグループはイニシャルRAMディスク(initrd)から起動します。この時、LVMツールはinitrdイメージに含まれる lvm.conf ファイルを検査します。そのため、 lvm.conf に変更を加えたら、ディストリビューションに応じてユーティリティ(mkinitrd、update-initramfs など)で確実にinitrdをアップデートしなければなりません。

新しいPVを設定したら、ボリュームグループに追加するか、 新しいボリュームグループを作成します。このとき、DRBDリソースがプライマリロールになっている必要があります。

```
# vgcreate <name> /dev/drbdX
```

同じボリュームグループ内にDRBD物理ボリュームとDRBD以外の物理ボリュームを混在させることができますが、これはお勧めできません。また、混在させても実際には意味がないでしょう。

VGを作成したら、 lvcreate
コマンドを使用して、(DRBD以外を使用するボリュームグループと同様に)ここから論理ボリュームを作成します。

9.5. 新しいDRBDボリュームを既存のボリュームグループへ追加する

新しいDRBDでバックアップした物理ボリュームを、ボリュームグループへ追加したいといった事があるでしょう。その場合には、新しいボリュームは既存のリソース設定に追加しなければなりません。そうすることでVG内の全PVにわたってレプリケーションストリームと書き込み順序の忠実性が確保されます。

LVMボリュームグループをPacemakerによる高可用性LVMで説明しているようにPacemakerで管理している場合には、DRBD設定の変更前にクラスタメンテナンスモードになっている事が 必要 です。

追加ボリュームを加えるには、リソース設定を次のように拡張します。

```
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;
  }
}
```

DRBD設定が全ノード間で同じである事を確認し、次のコマンドを発行します。

```
# drbdadm adjust r0
```

これによって、リソース r0 の新規ボリューム 1 を有効にするため、暗黙的に drbdsetup new-minor r0 1 が呼び出されます。新規ボリュームがレプリケーションストリームに追加されると、イニシャライズやボリュームグループへの追加ができるようになります。

```
# pvcreate /dev/drbd/by-res/<resource>/1
# vgextend <name> /dev/drbd/by-res/<resource>/1
```

で新規PVの /dev/drbd/by-res/<resource>/1 が <name> VGへ追加され、VG全体にわたって書き込みの忠実性を保護します。

9.6. DRBDを使用する入れ子のLVM構成

論理ボリュームをDRBDの下位デバイスとして使用し、かつ、同時にDRBDデバイス自体を物理ボリュームとして使用することもできます。 例として、次のような構成を考えてみましょう。

- /dev/sda1 と /dev/sdb1 という2つのパーティションがあり、 これらを物理ボリュームとして使用
- 2つのPVが local というボリュームグループに含まれる。
- このGVに10GiBの論理ボリュームを作成し、 r0 という名前を付ける。
- このLVがDRBDリソースのローカル下位デバイスになり、 名前は同じ r0 で、デバイス /dev/drbd0 に対応する。
- このデバイスが replicated というもう1つのボリュームグループの唯一のPVになる。
- このVGには foo (4GiB)と bar (6GiB)というさらに2つの論理ボリュームが含まれる。

この構成を有効にするために、次の手順を行います。

- /etc/lvm/lvm.conf で適切な filter オプションを設定する。

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

  このフィルタ表現が、SCSIおよびDRBDデバイスで見つかったPVシグネチャを受け入れ、その他すべてを拒否(無視)します。

  lvm.conf ファイルを変更したら vgscan コマンドを実行します。LVMは構成キャッシュを破棄し、デバイスを再スキャンしてPVシグネチャを見つけます。

- LVMキャッシュ無効の設定

  ```
  write_cache_state = 0
  ```

  LVMのキャッシュを無効にした後、 /etc/lvm/cache/.cache を削除して、古いキャッシュを削除してください。

- ここで、2つのSCSIパーティションをPVとして初期化します。

  ```
  # pvcreate /dev/sda1
  Physical volume "/dev/sda1" successfully created
  # pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
  ```

- 次に、初期化した2つのPVを含む "local" という名前の下位レベルVGを作成します。

  ```
  # vgcreate local /dev/sda1 /dev/sda2
  Volume group "local" successfully created
  ```

- これで、DRBDの 下位デバイスとして使用する論理ボリュームを作成可能です。

```
# lvcreate --name r0 --size 10G local
Logical volume "r0" created
```

- 対向ノードでも、ここまでのすべての手順を繰り返します。
- /etc/drbd.conf を編集して、r0 という名前の新しいリソースを作成。

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

新しいリソース構成を作成したら、忘れずに drbd.conf の内容を対向ノードにコピーします。

- リソースを初めて有効にするに従って(両方のノードの)リソースを初期化します 。
- 一方のノードリソースを昇格する。

```
# drbdadm primary r0
```

- リソースを昇格したノードで、DRBDデバイスを新しい物理ボリュームとして初期化します。

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

- 初期化したPVを使用して、同じノードに replicated というVGを作成します。

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

- 最後に、新しく作成したこのVG内に新しい論理ボリュームを作成します。[LVM]

```
# lvcreate --name foo --size 4G replicated
Logical volume "foo" created
# lvcreate --name bar --size 6G replicated
Logical volume "bar" created
```

これで、論理ボリューム foo と bar をローカルノードで /dev/replicated/foo と /dev/replicated/bar として使用できます。

9.6.1. VGを他ノードに移動する

他のノードでも使用できるように、プライマリノードで次のコマンドを実行します。

```
# vgchange -a n replicated
0 logical volume(s) in volume group "replicated" now active
# drbdadm secondary r0
```

次に他のノード(まだセカンダリの)でコマンドを発行します。

```
# drbdadm primary r0
# vgchange -a y replicated
2 logical volume(s) in volume group "replicated" now active
```

これでブロックデバイス /dev/replicated/foo と /dev/replicated/bar が他の(現在はプライマリの)ノードで有効になります。

9.7. Pacemakerによる高可用性LVM

対向ノード間でのボリュームグループの転送と、対応する有効な論理ボリュームの作成のプロセスは自動化することができます。Pacemakerの LVM リソースエージェントはまさにこのために作られています。

既存のPacemaker管理下にあるDRBDの下位デバイスのボリュームグループをレプリケートするために、 crm シェルで次のコマンドを実行してみましょう。

Listing 7. DRBDの下位デバイスのLVMボリュームグループに関するPacemaker設定

```
primitive p_drbd_r0 ocf:linbit:drbd \
  params drbd_resource="r0" \
  op monitor interval="29s" role="Master" \
  op monitor interval="31s" role="Slave"
ms ms_drbd_r0 p_drbd_r0 \
  meta master-max="1" master-node-max="1" \
      clone-max="2" clone-node-max="1" \
      notify="true"
primitive p_lvm_r0 ocf:heartbeat:LVM \
  params volgrpname="r0"
colocation c_lvm_on_drbd inf: p_lvm_r0 ms_drbd_r0:Master
order o_drbd_before_lvm inf: ms_drbd_r0:promote p_lvm_r0:start
commit
```

この設定を反映させると、現在のDRBDリソースのプライマリ(マスター)ロールがどちらであっても、Pacemakerは自動的に r0 ボリュームグループを有効にします。

10. DRBDでGFSを使用する

本章では共有グローバルファイルシステム(GFS)のブロックデバイスをDRBDリソースとするために必要な手順を説明します。GFS、GFS2の両方に対応しています。

DRBD上でGFSを使用するためには、 デュアルプライマリモードでDRBDを設定する必要があります。

全クラスタファイルシステムには fencingが_必要_ です。DRBDリソース経由だけでなくSTONITHもです。問題のあるノードは 必ず killされる必要があります。

次のような設定がよいでしょう。

```
net {
  fencing resource-and-stonith;
}
handlers {
  # Make sure the other node is confirmed
  # dead after this!
  outdate-peer "/sbin/kill-other-node.sh";
}
```

これらは 非揮発性 のキャッシュである必要があります。 詳細は https://fedorahosted.org/cluster/wiki/DRBD_Cookbook を参考にしてください。

10.1. GFS基礎

Red Hat Global File System (GFS)は、同時アクセス共有ストレージファイルシステムのRed Hatによる実装です。同様のファイルシステムのように、GFSでも複数のノードが読み取り/書き込みモードで、安全に同時に同じストレージデバイスにアクセスすることが可能です。これには、クラスタメンバからの同時アクセスを管理するDLM (Distributed Lock Manager)が使用されています。

本来、GFSは従来型の共有ストレージデバイスを管理するために設計されたものですが、デュアルプライマリモードでDRBDをGFS用のレプリケートされたストレージデバイスとして問題なく使用することができます。アプリケーションについては、読み書きの待ち時間が短縮されるというメリットがあります。 これは、GFSが一般的に実行されるSANデバイスとは異なり、DRBDが通常はローカルストレージに対して読み書きを行うためです。また、DRBDは各GFSファイルシステムに物理コピーを追加して、冗長性を確保します。

GFSはクラスタ対応版の LVMで、クラスタ化論理ボリュームマネージャ (CLVM)を使用します。このような対応関係が、GFSのデータストレージとしてDRBDを使用することと、従来のLVMの物理ボリュームとしてDRBDを使用することとの間に存在します。

GFSファイルシステムは通常はRedHat独自のクラスタ管理フレームワークの Red Hat Clusterと密接に結合されています。この章ではDRBDをGFSとともに使用する方法を Red Hat Clusterの観点から説明します。

GFS、Pacemaker、Red Hat ClusterはRed Hat Enterprise Linux (RHEL)と、 CentOSなどの派生ディストリビューションで入手できます。同じソースからビルドされたパッケージが Debian GNU/Linuxでも入手できます。この章の説明は、Red Hat Enterprise LinuxシステムでGFSを実行することを前提にしています。

10.2. GFS用のDRBDリソースの作成

GFSは共有クラスタファイルシステムで、すべてのクラスタノードからストレージに対して同時に読み取り/書き込みアクセスが行われることを前提としています。したがって、GFSファイルシステムを格納するために使用するDRBDリソースはデュアルプライマリモードで設定する必要があります。また、スプリットブレインからの自動回復のための機能を利用することをおすすめします。そのためには、以下の設定をリソース設定ファイル

に加えてください。

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

回復ポリシーを設定することは、事実上、自動データロス設定を行うことです。十分にご理解のうえご使用ください。

これらのオプションを新規に設定したリソースに追加したら、通常通りにリソースを初期化できます。リソースの allow-two-primaries オプションが yes に設定されているので、両ノードのリソースをプライマリにすることができます。

10.3. DRBDリソースを認識するようにLVMを設定する

GFSでは、ブロックデバイスを利用するためにクラスタ対応版のLVMであるCLVMを使用しています。DRBDでCLVMを使用するために、LVMの設定を確認してください。

- クラスタでのロックを行いますので、 /etc/lvm/lvm.conf で以下のオプション設定をしてください。

  ```
  locking_type = 3
  ```

- DRBDベースの物理ボリュームを認識させるためにDRBDデバイスをスキャンします。 これは、従来の(非クラスタの)LVMのように適用されます。 詳細はDRBDリソースを物理ボリュームとして構成するを参照してください。

10.4. GFS対応のためのクラスタ設定

新規DRBDリソースを作成し、初期クラスタ設定を完了したら、両ノードで以下のサービスを有効にして起動する必要があります。

- cman (同時に ccsd と fenced を起動します)
- clvmd.

10.5. GFSファイルシステムの作成

デュアルプライマリのDRBDリソースでGFSファイルシステムを作成するために、まずLVMの論理ボリュームとして初期化する必要があります。

従来のクラスタ非対応のLVM設定とは異なり、CLVMはクラスタ対応のため以下の手順は必ず1ノードでのみ実行してください。

```
# 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
```

この例ではボリュームリソースが1つの場合を前提にしています。

CLVMは即座に変更を対向ノードに伝えます。　lvs (または lvdisplay)を対向ノードで実行すれば、新規作成の論理ボリュームが表示されます。

ファイルシステムの作成を行います。

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

GFS2ファイルシステムの場合は以下になります。

```
# mkfs -t gfs2 -p lock_dlm -j 2 -t <cluster>:<name>
  /dev/<vg-name>/<lv-name>
```

コマンド中の -j オプションは、GFSで保持するジャーナルの数を指定します。これは同時にGFSクラスタ内で同時に プライマリ になるノード数と同じである必要があります。DRBD9までは同時に2つより多い プライマリ ノードをサポートしていないので、ここで設定するのは常に2です。

-t オプションはGFS2ファイルシステムでのみ使用できます。ロックテーブル名を定義します。ここでは <cluster>:<name> の形式を使用し、 <cluster> は /etc/cluster/cluster.conf で定義したクラスタ名と一致する必要があります。そのため、そのクラスタメンバーのみがファイルシステムを使用することができます。一方で <name> はクラスタ内で一意の任意のファイルシステム名を使用できます。

10.6. GFSファイルシステムを使用する

ファイルシステムを作成したら、 /etc/fstab に追加することができます。

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

GFS2の場合にはファイルシステムタイプを変更します。

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

この変更は必ずクラスタの両ノードで行ってください。

設定が終わったら gfs サービスを(両ノードで)開始することで、新規ファイルシステムをマウントすることができます。

```
# service gfs start
```

以降、システムの起動時にRHCSサービスと gfs サービスの前にDRBDが自動起動するよう設定してあれば、従来

の共有ストレージのようにGFSファイルシステムを使用することができます。

11. DRBDとOCFS2の使用

この章では、共有Oracle Cluster File Systemバージョン2 (OCFS2)を格納するブロックデバイスとしてDRBDリソースを設定する方法を説明します。

全クラスタファイルシステムには fencingが_必要_ です。DRBDリソース経由だけでなくSTONITHもです。問題のあるノードは 必ず killされる必要があります。

次のような設定がよいでしょう。

```
net {
  fencing resource-and-stonith;
}
handlers {
  # Make sure the other node is confirmed
  # dead after this!
  outdate-peer "/sbin/kill-other-node.sh";
}
```

これらは 非揮発性 のキャッシュである必要があります。 GFS2に関するものであり、OCFS2についてではありませんが、 https://fedorahosted.org/cluster/wiki/DRBD_Cookbook が参考になるでしょう。

11.1. OCFS2の基礎

Oracle Cluster File Systemバージョン2 (OCFS2)は、Oracle Corporationが開発した同時アクセス共有ストレージファイルシステムです。前バージョンのOCFSはOracleデータベースペイロード専用に設計されていましたが、OCFS2はほとんどのPOSIXセマンティクスを実装する汎用ファイルシステムです。OCFS2の最も一般的なユースケースは、もちろんOracle Real Application Cluster (RAC)ですが、OCFS2は負荷分散NFSクラスタなどでも使用できます。

本来、OCFS2は従来の共有ストレージデバイス用に設計されたものですが、dual-Primary DRBDでも問題なく使用できます。OCFS2が通常実行されるSANデバイスとは異なり、DRBDはローカルストレージに対して読み書きを行うため、ファイルシステムからデータを読み取るアプリケーションの読み取り待ち時間が短縮できるというメリットがあります。さらに、DRBDの場合は、単一のファイルシステムイメージを単に共有するのではなく、各ファイルシステムイメージにさらにコピーを追加するため、OCFS2に冗長性が加わります。

GFSなど他の共有クラスタファイルシステムと同様に、OCFS2で複数のノードが読み取り/書き込みモードで同じストレージデバイスに同時にアクセスできます。データが破損するおそれはありません。これには、クラスタノードからの同時アクセスを管理するDLM (Distributed Lock Manager)が使用されます。DLM自体は、システムに存在する実際のOCFS2ファイルシステムとは別個の仮想ファイルシステム(ocfs2_dlmfs)を使用します。

OCFS2は組み込みクラスタ通信層を使用して、クラスタメンバシップおよびファイルシステムのマウントとマウント解除操作を管理したり、これらのタスクをPacemakerクラスタインフラストラクチャに委ねることができます。

OCFS2は、SUSE Linux Enterprise Server (OCFS2が主にサポートされる共有クラスタファイルシステム)、CentOS、Debian GNU/LinuxおよびUbuntu Server Editionで利用できます。また、OracleはRed Hat Enterprise Linux (RHEL)用のパッケージも提供しています。この章の説明は、SUSE Linux Enterprise ServerシステムでOCFS2を実行することを前提にしています。

11.2. OCFS2用のDRBDリソースの作成

OCFS2は共有クラスタファイルシステムで、すべてのクラスタノードからストレージに対して同時に読み取り/書き込みアクセスが行われることを前提としています。したがって、OCFS2ファイルシステムを格納するために使用するDRBDリソースは、デュアルプライマリモードで設定する必要があります。また、スプリットブレインからの自動回復のための機能を利用することをおすすめします。そのためには、以下の設定をリソース設定ファイル に加えてください。

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

回復ポリシーを設定することは、事実上、自動データロス設定を行うことです。十分にご理解のうえご使用ください。

最初の構成の際に allow-two-primaries オプションを yes にするのはお勧めできません。これは、最初のリソース同期が完了してから有効にしてください。

これらのオプションを新規に設定したリソースに追加したら、通常通りにリソースを初期化できます。 リソースの allow-two-primaries オプションを yes にすると、両方のノードのリソースをプライマリロールに昇格することができます。

DRBD9.1では3ノード以上で プライマリ にすることが可能になる予定です。DRBD9.0ではプライマリは2つまでですが、複数のノードを セカンダリ にして冗長性を高めることができます。

11.3. OCFS2ファイルシステムの作成

OCFS対応の mkfs コマンドを使って、OCFS2ファイルシステムを作成します。

```
# 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
```

2ノードスロットOCFS2ファイルシステムが /dev/drbd0 に作成され、ファイルシステムのラベルが ocfs2_drbd0 に設定されます。mkfs の呼び出し時に、その他のオプションを指定することもできます。詳細は mkfs.ocfs2 システムマニュアルページを参照ください。

11.4. PacemakerによるOCFS2の管理

11.4.1. Pacemakerにデュアルプライマリ DRBDリソースを追加する

既存のデュアルプライマリDRBDリソースは、次の crm 設定でPacemakerリソース管理に追加することができます。

```
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
```

メタ変数 master-max=2 に注意してください。これはPacemakerのマスター/スレーブのデュアルマスターモードを有効にします。その場合、 allow-two-primaries も yes にDRBD設定で設定されている必要があります。設定していない場合にはリソースの検証中に設定エラーが発生します。

11.4.2. OCFS2をPacemakerで管理するには

OCFS2とロックマネージャ(DLM)の分散カーネルを管理するために、Pacemakerは、3つの異なるリソースエージェントを使用します。

- ocf:pacemaker:controld — PacemakerのDLMに対してのインタフェース
- ocf:ocfs2:o2cb — PacemakerのOCFS2クラスタ管理へのインタフェース
- ocf:heartbeat:Filesystem — Pacemakerのクローンとして構成したときにクラスタファイルシステムをサポートする 汎用ファイルシステム管理リソース

次の crm 設定のように リソースグループのクローン を作成することによって、OCFS2の管理に必要なPacemakerリソースをすべてのノードで起動できます。

```
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
```

この構成がコミットされると、Pacemakerは、クラスタ内のすべてのノードで controld と o2cb のリソースタイプのインスタンスを起動します。

11.4.3. PacemakerにOCFS2ファイルシステムを追加する

PacemakerはOCF2ファイルシステムにアクセスするのに、従来の ocf:heartbeat:Filesystem リソースエージェントを使います。これはクローンモードであっても同様です。Pacemakerの管理下にOCFS2ファイルシステムを配置するには、次の crm 設定を使用します。

```
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
```

この例ではボリュームリソースが1つの場合を前提にしています。

11.4.4. OCFS2ファイルシステムを管理するPacemakerの制約の追加

すべてのOCFS2関連のリソースとクローンを結びつけるには、Pacemaker構成に以下の制約を加えてください。

```
order o_ocfs2 ms_drbd_ocfs2:promote cl_ocfs2mgmt:start cl_fs_ocfs2:start
colocation c_ocfs2 cl_fs_ocfs2 cl_ocfs2mgmt ms_drbd_ocfs2:Master
```

11.5. Pacemakerを使わないOCFS2管理

OCFS2 DLMをサポートしない旧バージョンのPacemakerしか使えない場合、この節が参考になります。この節は以前の方式を使っている方の参照のためにだけ残してあります。新規インストールの場合にはPacemaker方式を使ってください。

11.5.1. OCFS2をサポートするようにクラスタを設定する

設定ファイルの作成

OCFS2は主要な設定ファイルとして /etc/ocfs2/cluster.conf を使用します。

OCFS2クラスタを作成する際には、必ず、両方のホストを設定ファイルに追加してください。クラスタの相互接続通信には、通常はデフォルトポート(7777)が適切です。他のポート番号を選択する場合は、DRBD (および他の設定済みTCP/IP)が使用する既存のポートと衝突しないポートを選択する必要があります。

cluster.conf ファイルを直接編集したくない場合は、 ocfs2console というグラフィカルな構成ユーティリティを使用することもできます。通常はこちらのほうが便利です。いずれの場合も /etc/ocfs2/cluster.conf ファイルの内容はおおよそ次のようになります。

```
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
```

クラスタ構成を設定したら、scp を使用して構成をクラスタの両方のノードに配布します。

O2CBドライバの設定

SUSE Linux Enterprisesシステム

SLESでは、 o2cb の起動スクリプトの configure オプションを利用することができます。

```
# /etc/init.d/o2cb configure
Configuring the O2CB driver.

This will configure the on-boot properties of the O2CB driver.
The following questions will determine whether the driver is loaded on
boot.  The current values will be shown in brackets ('[]').  Hitting
<ENTER> without typing an answer will keep that current value.  Ctrl-C
will abort.

Load O2CB driver on boot (y/n) [y]:
Cluster to start on boot (Enter "none" to clear) [ocfs2]:
Specify heartbeat dead threshold (>=7) [31]:
Specify network idle timeout in ms (>=5000) [30000]:
Specify network keepalive delay in ms (>=1000) [2000]:
Specify network reconnect delay in ms (>=2000) [2000]:
Use user-space driven heartbeat? (y/n) [n]:
Writing O2CB configuration: OK
Loading module "configfs": OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading module "ocfs2_nodemanager": OK
Loading module "ocfs2_dlm": OK
Loading module "ocfs2_dlmfs": OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Starting O2CB cluster ocfs2: OK
```

Debian GNU/Linuxシステム

Debianの場合は、 /etc/init.d/o2cb の configure オプションは使用できません。代わりに、 ocfs2-tools パッケージを再設定してドライバを有効にします。

```
# dpkg-reconfigure -p medium -f readline ocfs2-tools
Configuring ocfs2-tools
Would you like to start an OCFS2 cluster (O2CB) at boot time? yes
Name of the cluster to start at boot time: ocfs2
The O2CB heartbeat threshold sets up the maximum time in seconds that a node
awaits for an I/O operation. After it, the node "fences" itself, and you will
probably see a crash.

It is calculated as the result of: (threshold - 1) x 2.

Its default value is 31 (60 seconds).

Raise it if you have slow disks and/or crashes with kernel messages like:

o2hb_write_timeout: 164 ERROR: heartbeat write timeout to device XXXX after NNNN
milliseconds
O2CB Heartbeat threshold: `31`
      Loading filesystem "configfs": OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading stack plugin "o2cb": OK
Loading filesystem "ocfs2_dlmfs": OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Setting cluster stack "o2cb": OK
Starting O2CB cluster ocfs2: OK
```

11.5.2. OCFS2ファイルシステムの使用

クラスタ構成を完了して、ファイルシステムを作成すると、他のファイルシステムと同様にマウントすることができます。

```
# mount -t ocfs2 /dev/drbd0 /shared
```

dmesg コマンドで表示されるカーネルログに次のような行が見つかるはずです。

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

その時点から、両方のノードでOCFS2ファイルシステムに読み取り/書き込みモードでアクセスできるようになります。

12. DRBDでのXenの使用

この章では、Xenハイパーバイザを使用する仮想化環境のVBD (Virtual Block Device: 仮想ブロックデバイス)としてDRBDを使用する方法を説明します。

12.1. Xenの基礎

Xenはケンブリッジ大学(英国)で開発された仮想化フレームワークで、その後はXenSource, Inc. (現在はCitrix傘下)が維持管理しています。Debian GNU/Linux (バージョン4.0以降)、SUSE Linux Enterprise Server (リリース10以降)、Red Hat Enterprise Linux (リリース5)など、ほとんどのLinuxディストリビューションの比較的新しいリリースにはXenが含まれています。

Xenでは 準仮想化が使用されます。これは、仮想化ホストとゲスト仮想マシンの高度な協調を必要とする仮想化方式です。従来のハードウェアエミュレーションにもとづく仮想化ソリューションよりも高いパフォーマンスを実現します。適切な仮想化拡張機能をサポートするCPUの場合、 Xenは完全なハードウェアエミュレーションもサポートします。これはXenの用語ではHVM (「Hardware-assisted Virtual Machine: ハードウェア支援仮想マシン」)と呼ばれます。

本書の執筆時点で、XenがHVM用にサポートするCPU拡張機能はIntelのVirtualization Technology (VT、以前のコードネームは「Vanderpool」)およびAMDのSecure Virtual Machine (SVM、以前の「Pacifica」)です。

Xenは ライブマイグレーション をサポートします。これは、実行中のゲストオペレーティングシステムを1つの物理ホストからもう1つへ中断なく転送する機能です。

DRBDリソースをレプリケートされたXen用仮想ブロックデバイス(VBD)として設定すると、2つのサーバの両方でdomUの仮想ディスクとして使えます。さらに自動フェイルオーバさせることも可能になります。このように、DRBDは、仮想化以外の他の用途と同様に、Linuxサーバに冗長性を提供するだけでなく、Xenによって仮想化できる他のオペレーティングシステムにも冗長性を提供します。これには32ビットまたは64ビットIntel互換アーキテクチャで実行可能な実質上すべてのオペレーティングシステムが含まれます。

12.2. Xenとともに使用するためにDRBDモジュールパラメータを設定する

Xen Domain-0カーネルの場合は、 disable_sendpage を 1 に設定してDRBDモジュールをロードすることをお勧めします。ファイル /etc/modprobe.d/drbd.conf を作成するか開いて、次の行を入力します。

```
options drbd disable_sendpage=1
```

12.3. Xen VBDとして適切なDRBDリソースを作成する

Xenの仮想ブロックデバイスとして使用するDRBDリソースの設定は比較的簡単です。基本的には、他の目的に使用するDRBDリソースの構成とほぼ同様です。ただし、ゲストインスタンスの ライブマイグレーションを有効にしたい場合は、そのリソースの デュアルプライマリモードを有効にする必要があります。

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

デュアルプライマリモードを有効にするのは、Xenがライブマイグレーションを開始する前にすべてのVBDの書き込みアクセスをチェックするためです。リソースは、移行元ホストと移行先ホストの両方で使用されるように設定されます。

12.4. DRBD VBDの使用

DRBDリソースを仮想ブロックデバイスとして使用するには、Xen domU設定に次のような行を追加する必要があります。

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

この設定例では、resource という名前のDRBDリソースを読み取り/書きみモード(w)で /dev/xvda としてdomUで使用できるようにします。

もちろん、複数のDRBDリソースを単一のdomUで使用することもできます。その場合は、上記の例のように、コンマで区切って disk オプションにさらに項目を追加します。

ただし、次に示す3つの状況ではこの方法を使用できません。

- 完全に仮想化された(HVM) domUを構成する場合
- グラフィカルインストールユーティリティを使用してdomUをインストールし、 さらに そのグラフィカルインストーラが drbd: 構文をサポートしない場合。
- kernel 、 initrd 、 extra オプションを指定せずにdomUを構成し、代わりに bootloader と bootloader_args によりXen擬似ブートローダを使用するように指定し、さらにこの擬似ブートローダが drbd: 構文をサポートしない場合。
    - pygrub (Xen 3.3より前)および domUloader.py (SUSE Linux Enterprise Server 10のXenに同梱) は、仮想ブロックデバイス構成の drbd: 構文をサポートしない擬似ブートローダの例です。
    - Xen 3.3以降の pygrub およびSLES 11に同梱される domUloader.py のバージョンはこの構文を サポートします 。

このような場合は、従来の phy: デバイス構文、およびリソース名ではなく、リソースに関連付けられたDRBDデバイス名を使用する必要があります。ただし、この場合はDRBDの状態遷移をXenの外部で管理する必要があるため、drbd リソースタイプを使用する場合より柔軟性が低下します。

12.5. DRBDで保護されたdomUの開始、停止、移行

domUの開始

DRBDで保護されたdomUを設定したら、通常のdomUと同様に開始できます。

```
# xm create <domU>
Using config file "/etc/xen/<domU>".
Started domain <domU>
```

このとき、VBDとして設定したDRBDリソースは、プライマリロールに昇格され、通常どおりにXenにアクセスできるようになります。

domUの停止

これも同様に簡単です。

```
# xm shutdown -w <domU>
Domain <domU> terminated.
```

この場合も、domUが正常にシャットダウンされると、DRBDリソースがセカンダリロールに戻ります。

domUの移行

これも通常のXenツールで実行します。

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

この場合、短時間に連続して複数の管理ステップが自動的に実行されます。 * destination-host のリソースがプライマリロールに昇格されます。 * domU のライブマイグレーションがローカルホストで開始します。 * 移行先ホストへの移行が完了すると、 ローカル側でリソースがセカンダリロールに降格されます。

最初にリソースをデュアルプライマリモードに設定するのは、両方のリソースを短時間だけ両方のホストでプライマリロールとして実行する必要があるためです。

12.6. DRBDとXen統合の内部

Xenはネイティブで次のタイプの仮想ブロックデバイスをサポートします。

phy

このデバイスタイプは、ホスト環境で使用可能な「物理的な」ブロックデバイスをゲストdomUに基本的には透過的な方法で渡します。

file

このデバイスタイプは、ファイルベースのブロックデバイスイメージをゲストdomUで使用するためのものです。元のイメージファイルからループブロックデバイスを作成し、このブロックデバイスを phy デバイスタイプとほぼ同じ方法でdomUに渡します。

domU構成の disk オプションで設定された仮想ブロックデバイスが phy: と file: 以外の接頭辞を使用する場合、または接頭辞をまったく使用しない場合(この場合はXenのデフォルトの phy デバイスタイプが使用される)は、Xenスクリプトディレクトリ(通常は /etc/xen/scripts)にある block- prefix というヘルパースクリプトが使用されます。

DRBDは drbd デバイスタイプ用のスクリプト(/etc/xen/scripts/block-drbd)を提供しています。この章の前半で述べたように、このスクリプトは必要に応じてDRBDリソースの状態遷移を制御します。

12.7. XenとPacemakerの統合

DRBDで保護されるXen VBDのメリットを十分に活用するためにheartbeatを使用し、関連するdomUをheartbeatリソースとして管理することをお勧めします。

Xen domUをPacemakerリソースとして構成し、フェイルオーバを自動化することができます。これには Xen OCFリソースエージェントを使用します。この章で説明したXenデバイスタイプとして drbd を使用している場合は、Xenクラスタリソースで使用するために個別にdrbdリソースを設定する必要は ありません 。block-drbd ヘルパースクリプトによって必要なリソース移行がすべて実行されます。

DRBDパフォーマンスの最適化

13. ブロックデバイスのパフォーマンス測定

13.1. スループットの測定

DRBDを使用することによるシステムのI/Oスループットへの影響を測定する際には、スループットの 絶対値 はほとんど意味がありません。必要なのは、DRBDがI/Oパフォーマンスに及ぼす 相対的 な影響です。したがって、I/Oスループットを測定する際には、必ずDRBDがある場合とない場合の両方を測定する必要があります。

ここで説明するテストは過激なものです。データが上書きされるため、DRBDデバイスの同期が失われます。そのためこのテストは、テストが完了したら破棄できるスクラッチボリュームで行ってください。

I/Oスループットを見積もるには、比較的大きなデータチャンクをブロックデバイスに書き込み、システムが書き込み操作を完了するまでの時間を測定します。これは一般的なユーティリティである dd を使用して簡単に測定できます。比較的新しいバージョンにはスループット見積もり機能が組み込まれています。

簡単な dd ベースのスループットベンチマークは、たとえば次のようになります。ここでは、test という名前のスクラッチリソースが現在接続されており、両方のノードでセカンダリロールになっています。

```
# TEST_RESOURCE=test
# TEST_DEVICE=$(drbdadm sh-dev $TEST_RESOURCE | head -1)
# TEST_LL_DEVICE=$(drbdadm sh-ll-dev $TEST_RESOURCE | head -1)
# drbdadm primary $TEST_RESOURCE
# for i in $(seq 5); do
    dd if=/dev/zero of=$TEST_DEVICE bs=1M count=512 oflag=direct
  done
# drbdadm down $TEST_RESOURCE
# for i in $(seq 5); do
    dd if=/dev/zero of=$TEST_LL_DEVICE bs=1M count=512 oflag=direct
  done
```

このテストでは、512Mのデータチャンクを DRBDデバイスに書き込み、次に比較のために下位デバイスに書き込みます。両方のテストをそれぞれ5回繰り返し、統計平均を求めます。この結果は dd が生成したスループット測定値です。

新規に有効にしたDRBDデバイスの場合は、最初の dd 実行ではパフォーマンスがかなり低くなりますが、これは正常な状態です。アクティビティログがいわゆる「コールド」な状態のためで、問題はありません。

パフォーマンスの参考値についてはDRBDのスループット最適化をご参照ください。

13.2. レイテンシの測定

レイテンシ測定はスループット測定とはまったく異なります。I/Oレイテンシテストでは非常に小さいデータチャンク(理想的にはシステムが処理可能な最も小さいデータチャンク)を書き込み、書き込みが完了するまでの時間を測定します。通常はこれを何度か繰り返して、通常の統計変動を相殺します。

スループットの測定と同様に、I/O待ち時間の測定にも一般的な dd ユーティリティを使用できますが、異なる設定とまったく異なった測定を行います。

以下は簡単な dd ベースの待ち時間のマイクロベンチマークです。ここでは、 test という名前のスクラッチリソースが現在接続されており、両方のノードでセカンダリロールになっています。

```
# TEST_RESOURCE=test
# TEST_DEVICE=$(drbdadm sh-dev $TEST_RESOURCE | head -1)
# TEST_LL_DEVICE=$(drbdadm sh-ll-dev $TEST_RESOURCE | head -1)
# drbdadm primary $TEST_RESOURCE
# dd if=/dev/zero of=$TEST_DEVICE bs=4k count=1000 oflag=direct
# drbdadm down $TEST_RESOURCE
# dd if=/dev/zero of=$TEST_LL_DEVICE bs=4k count=1000 oflag=direct
```

このテストでは、４kiBのデータチャンクをDRBDデバイスに1,000回書き込み、次に比較のために下位デバイスにも1000回書き込みます。4096バイトはLinuxシステム(s390以外のすべてのアーキテクチャ)が扱うことができる最小のブロックサイズです。

dd が生成するスループット測定はこのテストではまったく意味がなく、ここで重要なのは、1,000回の書き込みが完了するまでにかかった時間です。この時間を1,000で割ると、1つのセクタへの書き込みの平均待ち時間を求めることができます。

これは、シングルスレッドで厳密に１つずつ書き込みをしますので、最悪のケースではあります。つまり、I/O-depthが1の場合などです。レイテンシ vs. IOPsをご参照ください。

参考情報としてDRBDレイテンシの最適化もご参照ください。

14. DRBDのスループット最適化

この章では、DRBDのスループットの最適化について説明します。スループットを最適化するためのハードウェアに関する検討事項と、チューニングを行う際の詳細な推奨事項について取り上げます。

14.1. ハードウェアの検討事項

DRBDのスループットは、配下のI/Oサブシステム(ディスク、コントローラおよび対応するキャッシュ)の帯域幅とレプリケーションネットワークの帯域幅の両方の影響を受けます。

I/Oサブシステムのスループット

I/Oサブシステムのスループットは主に並行して書き込み可能なストレージユニットの数と種類(ハードディスク、SSD、その他のフラッシュストレージ{FusionIOなど}…)によって決まります。比較的新しいSCSIまたはSASディスクの場合、一般に1つのディスクに約40MB/sでストリーミング書き込みを行うことができます。SSDなら300MiB/s、最新のフラッシュストレージ(NVMe)なら1GiB/sほどになります。ストライピング構成で配備する場合は、I/Oサブシステムにより書き込みがディスク間に並列化されます。 この場合、スループットは、1つのディスクのスループットに構成内に存在するストライプの数を掛けたものになります。RAID-0またはRAID-1`0構成で、 3つのストライプがある場合は同じ40MB/sのディスクのスループットが120MB/sになり、5つのストライプがある場合は200MB/sになります。SSDとNVMeまたはNVMeのみで構成すれば容易に1GiB/s超になるでしょう。

バッテリーバックアップされたバッファメモリを持つRAIDコントローラでは少量の書き込みの速度は、それらがバッファリングされることによって大幅に加速されます。非常に短い測定テストでは1GiBを示すかもしれません。持続的な書き込みではバッファがフルになるので速度は低下します。

ハードウェアのディスク_ミラーリング_(RAID-1)は、一般にスループットにはほとんど影響ありません。ディスクの_パリティ付きストライピング_(RAID-5)はスループットに影響し、通常はストライピングに比べてスループットが低下します。ソフトウェアRAIDのRAID-5やRAID-6であればさらに低下します。

ネットワークのスループット

ネットワークスループットは一般にネットワーク上に存在するトラフィック量と経路上にあるルータやスイッチなどのスループットによって決定されます。ただし、DRBDのレプリケーションリンクは通常は専用の背面間ネットワーク接続であるため、これらはあまり関係がありません。したがって、ネットワークのスループットを向上させるには、高スループットのプロトコル(10ギガビットイーサネットや56GiBインフィニバンド)に切り換えるか、複数のネットワークリンクからなるリンクアグリゲーションを使用します。後者はLinux bondingネットワークドライバを使用して実現できます。

14.2. スループットオーバヘッドの予測

DRBDに関連するスループットオーバヘッドを見積もる際には、 必ず次の制限を考慮してください。

- DRBDのスループットは下位I/Oサブシステムのスループットにより制限される。
- DRBDのスループットは使用可能なネットワーク帯域幅により制限される。

DRBDが使用できる理論上の 最大 スループットは上記の 小さい方 によって決まります。この最大スループットはDRBD自体のスループットオーバヘッドが加わることにより低下しますが、 これは3%未満だと考えられます。

- たとえば、600MB/sのスループットが可能なI/Oサブシステムを持つ2つのクラスタノードが、 ギガビットイーサネットリンクで接続されているとします 。ギガビットイーサネットのスループットは 110MB/s程度だと考えられるため、 この構成ではネットワーク接続がボトルネックになります。 DRBDの最大スループットは110MB/s程度でしょう。

- 一方、I/Oサブシステムの持続した書き込みスループットが80MB/s程度の場合は、これがボトルネックとなり、DRDBの最大スループットはわずか77MB/sになります。

14.3. チューニングの推奨事項

DRBDには、システムのスループットに影響する可能性があるいくつかの構成オプションがあります。ここでは、スループットを向上させるための調整について、いくつかの推奨事項を取り上げます。ただし、スループットは主としてハードウェアに依存するため、ここで取り上げるオプションを調整しても、効果はシステムによって大きく異なります。パフォーマンスチューニングについて、必ず効く「特効薬」は存在しません。

14.3.1. max-buffers と max-epoch-size の設定

これらのオプションはセカンダリノードの書き込みパフォーマンスに影響します。max-buffers はディスクにデータを書き込むためにDRBDが割り当てるバッファの最大数で、 max-epoch-size は、2つの書き込みバリア間で許容される書き込み要求の最大数です。パフォーマンスを上げるためには max-buffers は max-epoch-size 以上でなければなりません。 デフォルト値は両方とも2048です。比較的高パフォーマンスのハードウェアRAIDコントローラの場合、この値を8000程度に設定すれば十分です。

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

14.3.2. TCP送信バッファサイズの調整

TCP送信バッファは送信TCPトラフィックのメモリバッファです。デフォルトサイズは128KiBです。高スループットネットワーク (専用ギガビットイーサネット、負荷分散ボンディング接続など)で使用する場合は、このサイズを2MiBかそれ以上に設定すると効果があるでしょう。送信バッファサイズを16MiB以上にすることは一般にはお勧めできません。 また、スループットが向上する可能性もありません。

```
resource <resource> {
  net {
    sndbuf-size 2M;
    ...
  }
  ...
}
```

TCP送信バッファの自動調整もサポートされます。この機能を有効にすると、DRBDが適切なTCP送信バッファサイズを動的に選択します。TCP送信バッファの自動調整を有効にするには、次のようにバッファサイズをゼロに設定します。

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

sysctl 設定の net.ipv4.tcp_rmen と net.ipv4.tcp_wmem は挙動に影響します。これらの設定を確認してください。またこれらを 131072 1048576 16777216 (最小128kiB、デフォルト1MiB、最大16MiB)にするとよいかもしれません。

net.ipv4.tcp_mem はまったく異なるものですので、変更しないでください。間違った値を指定すると、容易にマシンがout-of-memoryになってしまいます。

14.3.3. アクティビティログサイズの調整

DRBDを使用するアプリケーションが、デバイス全体に分散した小さな書き込みを頻繁に発行する、書き込み集約型の場合は、十分に大きなアクティビティログを使用することをお勧めします。そうでない場合、頻繁なメタデータ更新により書き込みパフォーマンスが低下する可能性があります。

```
resource <resource> {
  disk {
    al-extents 6007;
    ...
  }
  ...
}
```

14.3.4. バリアとディスクフラッシュを無効にする

次に取り上げる推奨事項は、不揮発性(バッテリでバックアップされた)のコントローラキャッシュのあるシステム のみ に適用されます。

バッテリでバックアップされた書き込みキャッシュを持つシステムには、停電時にデータを保護する機能が組み込まれています。その場合は、同じ目的を持つDRBD自体の保護機能の一部を無効にすることもできます。これはスループットの面で有効な場合があります。

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

14.4. 冗長性を高め読み込み性能を向上させる

drbd.conf マニュアルページの read-balancing で説明の通り、データのコピーを追加することで読み込み性能を上げることができます。

概算: FusionIO カード利用の１ノードでの読み込み要求が fio では100kIOPsでしたが、 read-balancing を利用したた場合にはパフォーマンスが180kIOPsとなり、80%の性能向上が見られました!

読み込みのワークロードが多い環境の場合(ランダムリードが多い巨大なデータベースなど)では、read-balancing を有効にする意義があります。さらに読み込みIOスループットが欲しい場合にはコピーを増やすとよいでしょう。

15. DRBDレイテンシの最適化

この章では、DRBDのレイテンシの最適化について説明します。レイテンシを最小にするためのハードウェアに関する検討事項と、チューニングを行う際の詳細な推奨事項について取り上げます。

15.1. ハードウェアの検討事項

DRBDのレイテンシは、配下のI/Oサブシステム(ディスク、コントローラおよび対応するキャッシュ)のレイテンシとレプリケーションネットワークのレイテンシの両方の影響を受けます。

I/Oサブシステムのレイテンシ

HDDなど 回転するメディア の場合、I/Oサブシステムのレイテンシには、主としてディスクの回転速度が影響します。したがって、高速回転のディスクを使用すればI/Oサブシステムのレイテンシが短縮します。

SSDなど 回転しないメディア の場合は、フラッシュストレージのコントローラが重要な要素です。次に重要なのは未使用容量です。DRBDのTrim/Discardのサポートを使用すると、コントローラーにどのブロックがリサイクル可能かについて必要な情報を渡すことができます。そのため書き込み要求があった時に、事前に使用可能になっているブロックを すぐ に使用する事ができ、書き込めるスペースが空くまで待つ必要がありません。 [12]

同様に、 BBWC (Battery-Backed Write Cache)を使用すると、書き込み完了までの時間が短くなり、書き込みのレイテンシを短縮できます。手頃な価格のストレージサブシステムの多くは何らかのバッテリバックアップキャッシュを備えており、管理者がキャッシュのどの部分を読み書き操作に使用するか設定できます。ディスク読み取りキャッシュを完全に無効にし、すべてのキャッシュメモリをディスク書き込みキャッシュとして使用する方法をお勧めします。

ネットワークレイテンシ

ネットワークレイテンシは、基本的にはホスト間の round-trip time (RTT)RTT (Packet Round-Trip Time)です。 これ以外にもいくつかの要因がありますが、そのほとんどは、DRBDレプリケーションリンクとして推奨する、DRBD専用回線接続の場合は問題になりません。イーサネットリンクには常に一定のレイテンシが発生しますが、ギガビットイーサネットでは通常、100～200マイクロ秒程度のパケット往復時間です。

ネットワークレイテンシをこれより短くするには、レイテンシが短いネットワークプロトコルを利用する以外ありません。たとえば、Dolphin SuperSocketsのDolphin Express、10GBeの直結などを介してDRDBを実行します。これらの場合にはおよそ50マイクロ秒程になります。InfiniBandを利用するとさらにレイテンシが小さくなります。

15.2. レイテンシオーバーヘッドの予測値

スループットに関して、DRDBに関連するレイテンシオーバヘッドを見積もる際には、必ず次の制限を考慮してください。

- DRBDのレイテンシは下位I/Oサブシステムのレイテンシにより制限される。
- DRBDのレイテンシは使用可能なネットワークレイテンシにより制限される。

DRBDの理論上の 最小 待ち時間は、上記2つの 合計 です。 [13]。さらにわずかなオーバヘッドが追加されますが、これは1%未満だと予測されます。

- たとえば、ローカルディスクサブシステムの書き込みレイテンシが3msで、 ネットワークリンクのレイテンシが0.2msだとします予測されるDRBDの レイテンシは3.2msで、 ローカルディスクに書き込むだけのときと比べて約7%レイテンシが増加することになります。

CPUのキャッシュミス、コンテキストスイッチなど、他にも待ち時間に影響する要因があります。

15.3. レイテンシ vs. IOPs

IOPs は "I/O operations per second" の略です。

一般的にマーケティングでは数字が小さくなったとは書かないものです。プレスリリースでは "レイテンシが10マイクロ秒小さくなって、50マイクロ秒から40秒になりました!" とは書かずに、 "パフォーマンスが25%向上し、20000から25000IPOsになりました!" と書きます。"大きいものは良い" のような事を言うためIOPsは作られました。

言い方を換えれば、IOPsとレイテンシは相互的なものです。レイテンシの測定で書いた方法は、純粋にシーケンシャルのシングルスレッドのIOロードのIOPsごとのレイテンシです。一方で、よくある他のドキュメントでは、見栄えのいい数字を出すために多重並行処理でのIOPsを使用している [14]という事を覚えておいてください。こういったトリックを使うのなら、どんなIOPsにでもできます。

そのため、シリアルのシングルスレッドでのレイテンシを敬遠しないでください。 もっとIOPsが欲しい場合には fio を threads=8 で iodepth=16 にするなどの設定をしてください。しかし、そうした数字はデータベースに多数のクライアントからのコネクションが同時にあった場合など以外には、意味がないという事を覚えておいてください。

15.4. チューニングの推奨事項

15.4.1. CPUマスクの設定

DRBDでは、カーネルスレッドの明示的なCPUマスクを設定できます。これは、CPUサイクルがDRBDと競合するアプリケーションの場合に特に役立ちます。

CPUマスクは数字で、バイナリ表現の最下位ビットが第1のCPUを表し、その次のビットが第2のCPUを表します。ビットマスクの設定ビットは、対応するCPUがDRBDによって使用されていることを示し、クリアされたビットは使用されていないことを示します。たとえば、CPUマスク1(00000001)は、DRBDが第1のCPUだけを使用することを示します。マスク12(00001100)はDRBDが第3と第4のCPUを使用することを示します。

次に、リソースのCPUマスク設定の例を示します。

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

DRBDとこれを使用するアプリケーションとのCPU競合を最小限に抑えるためには、もちろん、DRBDが使用していないCPUだけをアプリケーションが使用するように設定する必要があります。

一部のアプリケーションは、DRBD自体と同様に設定ファイルでこの設定を行うことができます。アプリケーションによっては、initスクリプトで taskset コマンドを呼び出す必要があります。

複数のDRBDスレッドに同一のL2/L3キャッシュを使わせることは意味があります。

しかし、CPU番号の物理パーティショニングとの関連付けは必要ありません。 X11環境では lstopo (または hwloc-ls)プログラムを、コンソールでは hwloc-info -v -p を使うとトポロジーの概要を確認することができます。

15.4.2. ネットワークMTUの変更

レプリケーションネットワークの最大転送ユニット(MTU)サイズをデフォルトの1500バイトよりも大きくすることにはメリットがあります。いわゆる "ジャンボフレームを使用する" ことです。

MTUは、次のコマンドを使用して変更できます。

```
# ifconfig <interface> mtu <size>
```

または

```
# ip link set <interface> mtu <size>
```

<interface> にはDRBDのレプリケーションに使用するネットワークインタフェース名を指定します。<size> の一般的な値は9000 (バイト)です。

15.4.3. deadline I/Oスケジューラを有効にする

高性能なライトバックに対応したハードウェアRAIDコントローラを使う場合、CFQの代わりに単純な deadline をI/Oスケジューラに指定する方がDRBDのレイテンシを小さくできることがあります。通常はCFQがデフォルトで有効になっています。

I/Oスケジューラ構成に変更を加える場合は、/sys にマウントされる sysfs 仮想ファイルシステムを使用できます。スケジューラ構成は /sys/block/<device> に置かれています。<device>はDRBDが使用する下位デバイスです。

deadline スケジューラを有効にするには、次のコマンドを使用します。

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

次の値も設定することにより、さらに待ち時間を短縮できます。

- フロントマージを無効にします。

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

- 読み取りI/O deadlineを150ミリ秒にします(デフォルトは500ms)。

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

- 書き込みI/Oデッドラインを1500ミリ秒にします(デフォルトは3000ms)。

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

上記の値の変更により待ち時間が大幅に改善した場合は、システム起動時に自動的に設定されるようにしておくと便利です。 Debianおよび Ubuntuシステムの場合は、 sysfsutils パッケージと /etc/sysfs.conf 設定ファイルでこの設定を行うことができます。

グローバルI/Oスケジューラを選択するには、カーネルコマンドラインを使用して elevator オプションを渡します。そのためには、ブートローダ構成(GRUBブートローダを使用する場合通常は /etc/default/grub に格納)を編集し、カーネルブートオプションのリストに elevator=deadline を追加します。

[12] ローエンドのハードウェアの場合は、10~20%のスペースを未使用のままにしておくと、多少スペースの節約になります。
[13] プロトコルCの場合。他ノードも同様に書き込まなくてはならないため
[14] 例えば"16スレッドでIO-depthが32"これは512のI/O要求が並行して行われたという事です。

さらに詳しく知る

16. DRBDの内部

この章ではDRBDの内部アルゴリズムと内部構造について、 いくつか の背景情報を取り上げます。 これは、DRBDの背景について関心のあるユーザーが対象です。DRBD開発者が参考にできるほど深い内容には踏み込んでいません。開発者の方は資料に記載の文章やDRBDのソースコードを参照してください。

16.1. DRBDメタデータ

DRBDはレプリケートするデータに関するさまざまな情報を専用の領域に格納しています。メタデータには次のようなものがあります。

- DRBDデバイスのサイズ
- 世代識別子(GI、詳細は世代識別子を参照)
- アクティビティログ(AL、詳細はアクティビティログ)を参照
- クイック同期ビットマップ(詳細はクイック同期ビットマップを参照)

このメタデータは 内部 または 外部 に格納されます。格納する場所はリソースごとに設定できます。

16.1.1. 内部メタデータ

内部メタデータを使用するようにリソースを設定すると、DRBDはメタデータを実際の本稼働データと同じ下位レベルの物理デバイスに格納します。デバイスの 末尾 の領域がメタデータを格納するための領域として確保されます。

メリット

メタデータは実際のデータと密接にリンクされているため、ハードディスクに障害が発生しても、管理者は特に何かする必要はありません。メタデータは実際のデータとともに失われ、ともに復元されます。

デメリット

RAIDセットではなく、下位レベルデバイスが唯一の物理ハードディスクの場合は、内部メタデータが原因で書き込みスループットが低下することがあります。アプリケーションによる書き込み要求の実行により、DRBDのメタデータの更新が引き起こされる場合があります。メタデータがハードディスクの同じ磁気ディスクに格納されている場合は、書き込み処理によって、ハードディスクの書き込み/読み取りヘッドが2回余分に動作することになります。

すでにデータが書き込まれているディスク領域を下位デバイスに指定してDRBDでレプリケートする場合で、内部メタデータを使うには、DRBDのメタデータに必要な領域を 必ず 確保してください。

そうでない場合、DRBDリソースの作成時に、新しく作成されるメタデータによって下位レベルデバイスの末尾のデータが上書きされ、既存のファイルが破損します。

以下のいずれかの方法でこれを避けることができます。

- 下位レベルデバイスを拡張します。これには、LVMなどの論理 ボリューム管理機能を使用します。 ただし、対応するボリュームグループに有効な空き領域が必要です。 ハードウェアストレージソリューションを使用することもできます。
- 下位レベルデバイスの既存のファイルシステムを縮小します。ファイルシステムによっては 実行できない場合があります。
- 上記2つが不可能な場合は、 代わりに外部メタデータを使用します。

下位レベルデバイスをどの程度拡張するか、またはファイルシステムをどの程度縮小するかについては、メタデータサイズの見積りを参照してください。

16.1.2. 外部メタデータ

外部メタデータは、本稼働データを格納するものとは異なる個別の専用ブロックデバイスに格納します。

メリット
一部の書き込み処理では、外部メタデータを使用することにより、待ち時間をいくらか短縮できます。

デメリット
メタデータが実際の本稼働データに密接にリンクされません。つまり、ハードウェア障害により本稼働データだけが破損したか、DRBDメタデータは破損しなかった場合、手動による介入が必要です。生き残ったノードから切り替えるディスクに、手動でデータの完全同期を行う必要があります。

次の項目の すべて に該当する場合には、外部メタデータ使用する以外の選択肢はありません。

- DRBDでレプリケートしたい領域に すでにデータが格納されており、
- この既存デバイスを拡張することができず、
- デバイスの既存のファイルシステムが縮小をサポートしていない。

デバイスのメタデータを格納する専用ブロックデバイスが必要とするサイズににについては、メタデータサイズの見積りを参照してください。

外部メータデータは少なくとも1MBのデバイスサイズを必要とします。

16.1.3. メタデータサイズの見積り

次の式を使用して、DRBDのメタデータに必要な領域を正確に計算できます。

$$M_s = \lceil \frac{C_s}{2^{18}} \rceil * 8 * N + 72$$

図 14. DRBDメタデータサイズの計算(正確)

C_s はセクタ単位のデータデバイスサイズです。 **N** は対向ノードの数です。

デバイスサイズは(バイト単位で) blockdev --getsize64 <device> を実行して取得できます。MBに変換するときは1048576で割ってください(= 2^{20} または 1024^2)

実際には、次の計算で得られる大まかな見積りで十分です。この式では、単位はセクタではなくメガバイトである点に注意してください。

$$M_{MB} < \frac{C_{MB}}{32768} * N + 1$$

図 15. DRBDメタデータサイズの見積り(概算)

16.2. 世代識別子

DRBDでは 世代識別子 (GI)を使用してレプリケートされたデータの「世代」を識別します。

これはDRBDの内部メカニズムで、次のような目的で使用されます。

- 2つのノードが実際に同じクラスタのメンバか確認する (2つのノードが誤って接続されたものではないこと確認する)。

- バックグラウンド同期の方向を確認する (必要な場合)。
- 完全な再同期が必要か、 部分的な再同期で十分か判断する。
- スプリットブレインを検出する。

16.2.1. データ世代

DRBDは次のような場合に、新しい データ世代 の開始としてマークを付けます。

- デバイスの初期フル同期。
- 切断したリソースがプライマリロールに切り替わる。
- プライマリロールのリソースが切断する。

つまり、リソースのコネクションステータスが Connected になり、両方のノードのディスク状態が UpToDate になると、両方のノードの現在のデータ世代が同一になります。逆も同様です。 現在はノードのロール(プライマリ/セカンダリ)を表すために最下位ビットを使用しています。そのため、同じデータ世代であってもあるノードでは最下位ビットが異なることがあります。

データ世代は8バイトで定義される、全体でユニークな識別子(UUID)です。

16.2.2. 世代識別子タプル

DRBDでは、現在と履歴のデータ世代についての情報がローカルリソースメタデータに格納されます。

カレントUUID
これは、ローカルノードからみた最新のデータ世代の世代識別子です。リソースが Connected になり完全に同期されると、両ノードのカレントUUIDが同一になります。

ビットマップUUID
リモードホストごとに変更を追跡しているオンディスクのビットマップの世代のUUIDです。オンディスク同期ビットマップ自体と同様に、リモートホストと切断されてい場合のみ意味を持ちます。

履歴UUID
現在のものより以前のデータ世代の識別子で、リモートホストごとに1スロットです。

これらをまとめて 世代識別子タプル 、または略して「 GIタプル 」と呼びます。

16.2.3. 世代識別子の変化

新規データ世代の開始

それがネットワーク障害であれ、意図的なものであれ、 プライマリ のノードが対向ノードへのコネクションを失うと、DRBDは次のようにしてローカルの世代識別子を変更します。

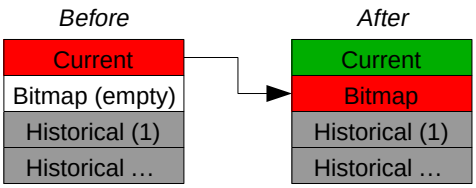


図 16. 新規データ世代の開始時に変化するGIタプル

1. プライマリが新しいデータ世代用の新規UUIDを作ります。これがプライマリノードの 新しい カレントUUIDになります。

2. 以前の カレントUUIDはビットマップが変更を追跡している世代を参照します。したがって、これがプライマリノードの新しいビットマップUUIDになります。
3. セカンダリノードではGIタプルは変化しません。

再同期の完了

再同期が完了すると、同期対象は同期元のGIタプルをすべて適用します。

同期元は元のUUIDを維持し、新しいUUIDは作成しません。

16.2.4. 世代識別子とDRBDの状態

ノード間の接続が確立すると、2つのノードは現在入手可能な世代識別子を交換し、それに従って処理を続行します。結果は次のようにいくつか考えられます。

両ノードのカレントUUIDが空の場合

ローカルノードと対向ノードの両方でカレントUUIDが空の状態です。新規に構成され、初回フル同期が完了していない場合は、通常この状態です。同期が開始していないため、手動で開始する必要があります。

1つのノードのカレントUUIDが空の場合

対向ノードのカレントUUIDが空で、自身は空でない場合です。これは、ローカルノードを同期元とした初期フル同期が進行中であることを表します。ローカルノードのDRBDはディスク上の同期ビットマップのすべてのビットを1にして、ディスク全体が非同期だと マークします。その後ローカルノードを同期元とした同期が始まります。逆の場合(ローカルのカレントUUIDが空で、対向ノードが空でない場合)は、DRBDは同様のステップをとります。ただし、ローカルノードが同期先になります。

カレントUUIDが等しい場合

ローカルのカレントUUIDと対向ノードのカレントUUIDが空でなく、同じ値を持っている状態です。両ノードがともにセカンダリで、通信切断中にどのノードもプライマリにならなかったことを表します。この状態では同期は必要ありません。

ビットマップUUIDが対向ノードのカレントUUIDと一致する場合

ローカルノードのビットマップUUIDが対向ノードのカレントUUIDと一致し、対向ノードのビットマップUUIDが空の状態です。これは、ローカルノードがプライマリで動作している間にセカンダリノードが停止して再起動したときに生じる正常な状態です。これは、リモートノードは決してプライマリにならず、ずっと同じデータ世代にもとづいて動作していたことを意味します。この場合、ローカルノードを同期元とする通常のバックグラウンド再同期が開始します。逆に、ローカルノード 自身の ビットマップUUIDが空で、 対向ノードの ビットマップがローカルノードのカレントUUIDと一致する状態の場合は、これはローカルノードの再起動に伴う正常な状態です。そして、ローカルノードを同期先とする通常のバックグラウンド再同期が開始します。

カレントUUIDが対向ノードの履歴UUIDと一致する場合

ローカルノードのカレントUUIDが対向ノードの履歴UUIDのうちの1つと一致する状態です。これは過去のある時点では同じデータを持っていたが、現在は対向ノードが最新のデータを持ち、しかし対向ノードのビットマップUUIDが古くなって使用できない状態です。通常の部分同期では不十分なため、ローカルノードを同期元とするフル同期が開始します。DRBDはデバイス全体を非同期状態とし、ローカルノードを同期先とするバックグラウンドでのフル再同期を始めます。逆の場合(ローカルノードの履歴UUIDのうち1つが対向ノードのカレントUUIDと一致する)、DRBDは同様のステップを行いますが、ローカルノードが同期元となります。

ビットマップUUIDが一致し、カレントUUIDが一致しない場合

ローカルノードのカレントUUIDが対向ノードのカレントUUIDと異なるが、ビットマップUUIDは一致する状態はスプリットブレインです。ただし、データ世代は同じ親を持っています。この場合、設定されていればDRBDがスプリットブレイン自動回復ストラテジが実行されます。設定されていない場合、DRBDはノード間の通信を切断し、手動でスプリットブレインが解決されるまで待機します。

カレントUUIDもビットマップUUIDも一致しない場合

ローカルノードのカレントUUIDが対向ノードのカレントUUIDと異なり、ビットマップUUIDも 一致しない 状態です。これもスプリットブレインで、しかも過去に同一のデータ状態であったという保証もありません。したがって、自動回復ストラテジが構成されていても役に立ちません。DRBDはノード間通信を切断し、手動でスプリットブレインが解決されるまで待機します。

いずれのUUIDも一致しない場合

最後は、DRBDが2つのノードのGIタプルの中に一致するものを1つも検出できない場合です。この場合DRBDは、"Unrelate data"という警告をログに書き込んでコネクションを切断します。これは、相互にまったく関連のない2つのクラスタノードが誤って接続された場合に備えるDRBDの機能です。

16.3. アクティビティログ

16.3.1. 目的

書き込み操作中に、DRBDは書き込み操作をローカルの下位ブロックデバイスに転送するだけでなく、ネットワークを介して送信します。実用的な目的で、この2つの操作は同時に実行されます。タイミングがランダムな場合は、書込み操作が完了しても、ネットワークを介した転送がまだ始まっていないといった状況が発生する可能性があります。その逆の場合もあります。

この状況で、アクティブなノードに障害が発生してフェイルオーバが始まると、このデータブロックのノード間の同期は失われます。障害が発生したノードにはクラッシュ前にデータブロックが書き込まれていますが、レプリケーションはまだ完了していません。そのため、ノードが回復しても、このブロックは回復後の同期のデータセット中から取り除かれる必要があります。さもなくば、クラッシュしたノードは生き残ったノードに対して「先書き」状態となり、レプリケーションストレージの「オール・オア・ナッシング」の原則に違反してしまいます。これはDRBDだけでなく、実際、すべてのレプリケーションストレージの構成で問題になります。バージョン0.6以前のDRBDを含む他の多くのストレージソリューションでは、アクティブなノードに障害が発生した場合、回復後にそのノードを改めてフル同期する必要があります。

バージョン0.7以降のDRBDは、これとは異なるアプローチを採用しています。アクティビティログ(AL)は、メタデータ領域にに格納され、「最近」書き込まれたブロックを追跡します。この領域は ホットエクステント と呼ばれます。

アクティブモードだったノードに一時的な障害が発生し、同期が行われる場合は、デバイス全体ではなくALでハイライトされたホットエクステントだけが同期されます。(それに加えて現在アクティブな対向ノードのビットマップのマークされたブロックも)これによって、アクティブなノードがクラッシュしたときの同期時間を大幅に短縮できます。

16.3.2. アクティブエクステント

アクティビティログの設定可能なパラメータに、アクティブエクステントの数があります。アクティブエクステントは4MiB単位でプライマリのクラッシュ後に再送されるデータ量に追加されます。このパラメータは、次の対立する2つの状況の折衷案としてご理解ください。

アクティブエクステントが多い場合

大量のアクティビティログを記録すれば書き込みスループットが向上します。新しいエクステントがアクティブになるたびに、古いエクステントが非アクティブにリセットされます。この移行には、メタデータ領域への書き込み操作が必要です。アクティブエクステントの数が多い場合は、古いアクティブエクステントはめったにスワップアウトされないため、メタデータの書き込み操作が減少し、その結果パフォーマンスが向上します。

アクティブエクステントが少ない場合

アクティビティログが小さい場合は、アクティブなノードが障害から回復した後の同期時間が短くなります。

16.3.3. アクティビティログの適切なサイズの選択

エクステントの数は所定の同期速度における適切な同期時間にもとづいて定義します。アクティブエクステントの数は次のようにして算出できます。

$$E = \frac{R * t_{sync}}{4MiB}$$

図 17. 同期速度とターゲットの同期時間にもとづくアクティブエクステントの計算

R はMiB/秒単位の同期速度、t_{sync} 秒単位のターゲットの同期時間です。E は求めるアクティブエクステントの数です。

スループット速度が200MiByte/秒のI/Oサブシステムがあり、同期速度(R)が60MiByte/sに設定されているとします。ターゲットの同期時間(t_{sync}) は4分または240秒を維持する必要があります。

図 18. 同期速度とターゲット同期時間にもとづくアクティブエクステントの計算(例)

また、最後に付け加えると、DRBD9ではデータを他のセカンダリノードへ同期させるので、セカンダリノードでもALを維持する必要があります。

16.4. クイック同期ビットマップ

クイック同期ビットマップはDRBDが各対向ノードがリソースごとに使用する内部データ構造で、同期ブロック(両方のノードで同一)または非同期ブロックを追跡します。ビットマップはノード間通信が切断しているときのみ使われます。

クイック同期ビットマップでは、1ビットが4KiBチャンクのオンディスクデータを表します。ビットがクリアされていれば、対応するブロックが対向ノードと同期しています。つまり、切断以降、ブロックに書き込まれていないということです。逆に、ビットがセットされていればブロックが変更されているため、接続が再確立したらすぐに再同期を行う必要があります。

スタンドアロンノードでディスクにデータが書き込まれると、クイック同期ビットマップへの書き込みも始まります。ディスクへの同期的なI/Oは負荷が大きいため、実際にはメモリ上のビットマップのビットがセットされます。アクティビティログが期限切れになってブロックがコールドになると、メモリ上のビットマップがディスクに書き込まれます。同様に、生き残ったスタンドアロンのノードでリソースが手動でシャットダウンされると、DRBDは すべての ビットマップをディスクにフラッシュします。

リモートノードが回復するか接続が再確立すると、DRBDは両方のノードのビットマップ情報を照合して、再同期が必要な すべてのデータ領域 を決定します。同時にDRBDは世代識別子を調べ、同期の 方向 を決定します。

同期元ノードが同期対象ブロックを対向ノードに送信し、同期先が変更を確認すると、ビットマップの同期ビットがクリアされます。その後再開すると、中断した箇所から同期を続行します。 — 中断中にブロックが変更された場合、もちろんそのブロックが再同期データセットに追加されます。

drbdadm pause-sync と drbdadm resume-sync コマンドを使用して、再同期を手動で一時停止したり再開することもできます。ただしこれは慎重に行ってください。再同期を中断すると、セカンダリノードのディスクが必要以上に長く Inconsistent 状態になります。

16.5. peer fencingインタフェース

DRBDにはレプリケーションリンクが途切れたときに対向ノードをフェンシング [15]するよう定義されたインタ

フェースがあります。Heartbeatに同梱の drbd-peer-outdater
ヘルパーはこのインタフェースのリファレンス実装です。ただし、独自のpeer fencingヘルパープログラムも簡単に実装できます。

fencingヘルパーは次のすべてを満たす場合にのみ呼び出されます。

1. リソース(または common)の handlers セクションで fence-peer ハンドラが定義されており
2. fencing オプションで、 resource-only または resource-and-stonith が設定されており、
3. レプリケーションリンクの中断時間が、DRBDがネットワーク障害を検出するために十分である [16]

fence-peer ハンドラとして指定されたプログラムかスクリプトが呼び出されると、 DRBD_RESOURCE と DRBD_PEER 環境変数が利用できるようになります。これらの環境変数には、それぞれ、影響を受けるDRBDリソース名と対向ホストのホスト名が含まれています。

peer fencingヘルパープログラム(またはスクリプト)は、次のいずれかの終了コードを返します。

表 1. fence-peer ハンドラの終了コード

終了コード	意味
3	対向ノードのディスク状態がすでに Inconsistent になっている。
4	対向ノードのディスク状態が正常に Outdated に設定された(または最初から Outdated だった。)
5	対向ノードへの接続に失敗。対向ノードに到達できなかった。
6	影響を受けるリソースがプライマリロールになっていたため、対向ノードを無効にできなかった。
7	対向ノードがクラスタから正常にフェンシングされた。影響を受けるリソースの fencing を resource-and-stonith に設定しておかなければ発生しない。

[15] フェンシングとSTONITHについてはhttp://clusterlabs.org/doc/crm_fencing.htmlの対応するPacemakerのページをご参照ください
[16] ネットワークリンク切断によるTCPタイムアウト、 ping-timeout 、またはカーネルによるコネクション中断など

17. 詳細情報の入手

17.1. 商用DRBDのサポート

DRBDの商用サポート、コンサルティング、トレーニングサービスは、プロジェクトのスポンサ企業LINBIT社が提供しています。日本ではhttp://www.linbit.com/[LINBIT]社との提携にもとづいてサイオステクノロジー株式会社が各種サポートを提供しています。

17.2. 公開メーリングリスト

DRBDの一般的な使用法について質問がある場合は、公開メーリングリスト drbd-user@lists.linbit.com にお問い合わせください。このメーリングリストは登録が必要です。 https://lists.linbit.com/listinfo/drbd-user/ で登録してください。リスト全体のアーカイブは https://lists.linbit.com/pipermail/drbd-user/ にあります。

17.3. 公開IRCチャンネル

公開IRCサーバ irc.freenode.net の次のチャンネルに、DRBDの開発者も参加しています。

- #drbd, and
- #clusterlabs.

IRCを使って、DRBDの改善を提案したり、開発者レベルの議論をすることもあります。

問題について質問する際にはDRBDの設定、ログファイル、 drbdsetup status --verbose --statistics の結果、/proc/drbd の内容を提供してください。これらの情報がない場合には回答は困難です。

17.4. 公式twitterアカウント

LINBITはツイッターをやっています。こちらです:http://twitter.com/linbit[linbit]

DRBDについてツイートする時には、どうぞ #drbd のハッシュタグをお使いください。

17.5. 資料

DRBDの開発者により作成されたDRBD一般やDRBDの特定の機能についての文書が公開されています。その一部を次に示します。

- Philipp Reisner. 'DRBD 9 - What's New'. 2012. Available at http://www.drbd.org/fileadmin/drbd/publications/whats-new-drbd-9.pdf // FIXME only bad version at https://www.netways.de/fileadmin/images/Events_Trainings/Events/OSDC/2013/Slides_2013/Philipp_Reisner_Neues_in_DRBD9.pdf // FIXME - put fixed version there
- Lars Ellenberg. 'DRBD v8.0.x and beyond'. 2007. Available at http://drbd.linbit.com/fileadmin/drbd/publications/drbd8.linux-conf.eu.2007.pdf
- Philipp Reisner. 'DRBD v8 - Replicated Storage with Shared Disk Semantics'. 2007. Available at http://drbd.linbit.com/fileadmin/drbd/publications/drbd8_orig.pdf FIXME sagt 2005?
- Philipp Reisner. 'Rapid resynchronization for replicated storage'. 2006. Available at http://drbd.linbit.com/fileadmin/drbd/publications/drbd-activity-logging_v6.pdf.

You can find many more on http://drbd.linbit.com/home/publications/.

17.6. その他の資料

- WikipediaにはHttp://en.wikipedia.org/wiki/DRBD[DRBDの記事]があります。
- Linux-HA wikiと
- ClusterLabs websiteに、高可用性クラスタでDRBDを活用するための有益な情報があります 。

付録

付録 A: 最近の変更

この付録はDRBD9.0以前のバージョンからアップグレードするユーザー向けです。DRBDの設定と挙動についての重要な変更点を説明します。

A.1. コネクション

DRBD9では3台以上のノードでもデータのレプリケーションが可能です。

そのためDRBDボリュームのスタック利用はもはや(可能ではありますが)推奨しません。 また、DRBDをネットワークブロックデバイス(DRBD client)として利用する事が可能になりました。

関連する変更項目

- メタデータサイズの変更(対向ノードごとに1ビットマップ)
- /proc/drbd では最小限の情報のみを表示。 <<s-drbdadm-status, drbdadm status> 参照
- 再同期の同期元/先が複数の対向ノードに。
- アクティビティログが セカンダリロールでも使用される

A.2. 自動プロモーション 機能

DRBD9では必要なときに自動的に プライマリ/セカンダリ のロール切り替え を行います。

この機能は、 become-primary-on の設定ならびにHeartbeat v1の drbddisk スクリプトを無用にするものです。

詳細はリソースの自動プロモーションを参照してください。

A.3. パフォーマンスの向上

DRBD9はパフォーマンスの面でも目覚ましく向上しました。ハードウェアによっては最大で100倍高速になります。（ランダム書き込みにおける1秒あたりのI/O処理数の測定）

A.4. 1リソース内の複数ボリューム

ボリュームはDRBD 8.3以前からの新しい概念です。DRBD8.4より前はリソースには1つのブロックデバイスだけしか設定できませんでしたので、DRBDデバイスとリソースには1対1の関係でした。DRBD8.4からは複数ボリューム(それぞれが1つのブロックデバイスと対応する)が、1つのレプリケーションコネクションを共有します。そしてこのレプリケーションコネクションが、1つのリソースに対応します。

この結果、挙動においていくつか必要な変更点があります。

A.4.1. udevシンボリックリンクの変更

DRBDのudev統合スクリプトは個々のブロックデバイスノードを示すシンボリックリンクを管理します。これらは /dev/drbd/by-res/ と /dev/drbd/by-disk/ ディレクトリにあります。

DRBD8.4以降、1つのリソースが複数ボリュームに対応できるので、 /dev/drbd/by-res/<resource> は個々のボリュームに貼られたシンボリックリンクを含む directory になりました。

Listing 8. DRBD 8.4でのudevが管理するDRBDシンボリックリンク

```
lrwxrwxrwx 1 root root 11 2015-08-09 19:32 /dev/drbd/by-res/home/0 -> ../../drbd0
lrwxrwxrwx 1 root root 11 2015-08-09 19:32 /dev/drbd/by-res/data/0 -> ../../drbd1
lrwxrwxrwx 1 root root 11 2015-08-09 19:33 /dev/drbd/by-res/nfs-root/0 -> ../../drbd2
lrwxrwxrwx 1 root root 11 2015-08-09 19:33 /dev/drbd/by-res/nfs-root/1 -> ../../drbd3
```

シンボリックリンクが参照するファイルシステムの構成は、DRBD 8.4に移行する場合には、通常は単純にシンボリックリンクのパスに /0 を追加することで更新する必要があります。ファイルシステムが /etc/fstab で UUID= や /dev/drbdX で参照されていれば、変更は必要ありません。

A.5. 設定上の構文の変更点

このセクションでは構成における構文の変更点について扱います。DRBD設定ファイルの /etc/drbd.d と /etc/drbd.conf に影響があります。

drbdadm のパーサーは8.4より前の構文も受け付けて、内部で自動的に現在の構文に変換します。DRBD9の新機能を使わないのであれば、これまでの構文に変更を加える必要はありません。しかし、DRBD9自体は古い構文に対応していないので、新しい構文を使用することをお勧めします。

A.5.1. ブール型設定オプション

drbd.conf は多くのブール型設定オプションに対応しています。DRBD 8.4より前の構文では、これらのブール型オプションは以下のようになっていました。

Listing 9. DRBD 8.4より前のブール型オプションが付いた構成例

```
resource test {
  disk {
    no-md-flushes;
  }
}
```

これには設定上の問題がありました。 common 構成セクションにブール型変数を設定したいとき、その後で個々のリソースを上書きしていました。

Listing 10. DRBD 8.4より前の common セクションにブール型オプションが付いた設定例[source,drbd]

```
common {
  disk {
    no-md-flushes;
  }
}
resource test {
  disk {
    # "common"で無効化した機能を有効化する方法はない
  }
}
```

DRBD 8.4では、すべてのブール型オプションは yes か no の値をとります。 common からでも個々の resource セクションからでも、これらオプションが操作しやすくなっています。

Listing 11. common セクションでのDRBD 8.4のブール型オプションのある設定例

```
common {
  md-flushes no;
}
resource test {
  disk {
    md-flushes yes;
  }
}
```

A.5.2. syncer セクションはなくなりました

DRBD 8.4より前では、設定構文で syncer セクションが使用できましたが、8.4では廃止されました。既存の syncer オプションはリソース内の net や disk セクションに移動しました。

Listing 12. DRBD8.4より前の syncer セクションのある設定例

```
resource test {
  syncer {
    al-extents 3389;
    verify-alg md5;
  }
  ...
}
```

上記の例で表しているものは、DRBD 8.4の構文では次のようになります。

Listing 13. syncer セクションが置き換えられたDRBD 8.4の設定例

```
resource test {
  disk {
    al-extents 3389;
  }
  net {
    verify-alg md5;
  }
  ...
}
```

A.5.3. protocol オプションが変則的ではなくなりました

以前のDRBDリリースでは、 protocol オプションは奇妙にも(そして直感的でなく)、 net セクションではなくて単独で記載する必要がありました。DRBD8.4でこの変則がなくなりました。

Listing 14. DRBD8.4より前の単独で protocol オプションがある設定例

```
resource test {
  protocol C;
  ...
  net {
    ...
  }
  ...
}
```

対応するDRBD8.4での設定は以下のようになります。

Listing 15. net 内に protocol オプションがあるDRBD8.4の設定例

```
resource test {
  net {
    protocol C;
    ...
  }
  ...
}
```

A.5.4. 新しいリソース毎の options セクション

DRBD 8.4では、resource セクション内でも common セクション内でも定義できる新しい options セクションを導入しました。以前は不自然にも syncer セクションで設定していた cpu-mask オプションは、このセクションに移動しました。8.4より前のリリースで disk セクションにあった on-no-data-accessible オプションも同様にこのセクションに移動しました。

Listing 16. DRBD 8.4より前の cpu-mask と on-no-data-accessible オプションのある設定例.

```
resource test {
  syncer {
    cpu-mask ff;
  }
  disk {
    on-no-data-accessible suspend-io;
  }
  ...
}
```

対応するDRBD8.4での設定は以下のようになります。

Listing 17. options セクションのあるDRBD 8.4の設定例

```
resource test {
  options {
    cpu-mask ff;
    on-no-data-accessible suspend-io;
  }
  ...
}
```

A.6. ネットワーク通信のオンライン変更

A.6.1. レプリケーションプロトコルの変更

DRBD 8.4より前では、リソースがオンラインまたはアクティブの状態では、レプリケーションプロトコルの変更は不可能でした。リソースの構成ファイル内で protocol オプションを変更し、それから両ノードで drbdadm disconnect を実行し、最後に drbdadm connect を行わなければなりませんでした。

DRBD 8.4では即座にレプリケーションプロトコルの変更が行えます。たとえば、一時的に接続を通常の同期レプリケーションから非同期レプリケーションのモードに変更することができます。

Listing 18. 接続が確率された状態でレプリケーションプロトコルを変更する

```
drbdadm net-options --protocol=A <resource>
```

A.6.2. シングルプライマリからデュアルプライマリのレプリケーションに変更する

DRBD 8.4より前では、リソースがオンラインやアクティブの状態ではシングルプライマリからデュアルプライマリに変更したり戻したりする事はできませんでした。リソースの構成ファイル内の allow-two-primaries オプションを変更し、 drbdadm disconnect を実行し、それから drbdadm connect を両ノードで実行する必要がありました。

DRBD 8.4ではオンラインで変更が行えます。

DRBDのデュアルプライマリモードを使うアプリケーションはクラスタファイルシステムまたは他のロッキング機構を使用していることが 必要 です。このために、デュアルプライマリモードが一時的または永続的であることは区別されません。

リソースがオンライン中にデュアルプライマリに変更する場合は、一時的なデュアルプライマリモードを参照してください。

A.7. drbdadm コマンドの変更点

A.7.1. pass-throughオプションの変更点

DRBD 8.4以前では、drbdadm で特殊なオプションを drbdsetup に引き渡す場合には、次の例のように難解な -- --<option> の構文を使わなければなりませんでした。

Listing 19. DRBD 8.4より前の drbdadm の引き渡しオプション

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

これに代わって、通常オプションと同じように drbdadm に引き渡すオプションが使えるようになりました。

Listing 20. DRBD 8.4の drbdadm 引き渡しオプション

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

古い構文もまだサポートされますが、使わないことを強くお勧めします。なお、この新しい簡単な構文を使用する場合には、(--discard-my-data)オプションを、(connect)サブコマンドの 後 で、かつリソース識別子の 前 に指定する必要があります。

A.7.2. --overwrite-data-of-peer は --force オプションに置き換わりました

--overwrite-data-of-peer オプションはDRBD8.4でなくなり、より簡潔な --force に置き換わりました。このため、リソースの同期を開始するために、次のコマンドは使えません。

Listing 21. DRBD 8.4より前の drbdadm 同期開始コマンド.

```
drbdadm -- --overwrite-data-of-peer primary <resource>
```

代わりに次のようなコマンドをご使用ください。

Listing 22. DRBD 8.4の drbdadm 同期開始コマンド

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

A.8. デフォルト値の変更点

DRBD 8.4では、Linuxカーネルや利用できるサーバハードウェアの向上に合わせて、drbd.conf のいくつかのデフォルト値が更新されました。

A.8.1. 同時にアクティブなアクティビティログのエクステント数(al-extents)

al-extents の以前のデフォルト値の127は1237に変更になりました。これによりメタデータのディスク書き込み作業量の現象によるパフォーマンスの向上が可能になりました。この変更によりプライマリノードのクラッシュ時の再同期時間が長くなります。これはギガビットイーサネットと高帯域幅のレプリケーションリンクの一般化が関係しています。

A.8.2. ランレングス符号化(use-rle)

ビットマップ転送のためのランレングス符号化(RLE)がDRBD 8.4ではデフォルトで有効になっています。use-rle オプションのデフォルト値は yes です。RLEは、2つの切断されたノードの接続時に常に起きる クイック同期ビットマップ中のデータ転送量を大きく減らします。

A.8.3. I/Oエラーの処理ストラテジー(on-io-error)

DRBD 8.4ではデフォルトでI/Oスタックの上位レイヤにI/Oエラーを伝えない(detach))設定です。以前のバージョンではI/Oエラーを伝える(pass-on)設定でしたが、置き換わりました。つまり問題のあるドライブのDRBDボリュームが自動的に Diskless のディスク状態に遷移し、対向ノードからのデータを提供します。

A.8.4. 可変レート同期

可変レート同期はDRBD 8.4ではデフォルトで有効です。デフォルト設定は次の構成のようになっています。

Listing 23. DRBD 8.4での可変レート同期のデフォルトオプション

```
resource test {
  disk {
    c-plan-ahead 20;
    c-fill-target 50k;
    c-min-rate 250k;
  }
  ...
```

A.8.5. 設定可能なDRBDデバイス数(minor-count)

DRBD 8.4で設定可能なDRBDデバイス数は1,048,576 (2^{20})です。(前のバージョンでは255)これは理論的な限界であり、本番環境で使い切ることはないでしょう。