1.1 信頼性を上げるためのテクノロジー

ネットワーク(特にLAN)上で複数のノード・システムを組むクラスタリング・システム(クラスタ・システム)では，信頼性を上げるためにさまざまなテクノロジーが駆使されています．

たとえば，アベイラビリティの監視(各ノード・システム間での稼働監視)や，クライアントからの接続に対する冗長化や負荷/機能分散，クラスタリング・システム内でのデータやデバイスのシェアリングやミラーリングあるいは同期，ネットワーク上のサーバ・サービスをはじめとしてデバイスやインターフェース，各ノードのCPUからプロセス，平均負荷，メモリ，あるいはネットワーク管理要素(ディスクやファイルの利用率など)やシステム情報をリソースとして監視するネットワーク監視などです．

アベイラビリティの監視

アベイラビリティの監視とは，各ノード(特に各サーバ)間で，物理的なルート(回線)や論理的なルート(TCP/IPプロトコル)を複数利用して動作状況を把握し合うことにより信頼性を高める仕組みです．複数の異種のネットワークを接続し合う物理的なルートでは，物理的回線の「単一障害点」(一つのポイントにネットワークが連なり，そこで障害が発生した場合はネットワーク全体の障害となる)を防止できるようになります．

Ethernet回線だけでなくファイバ・チャネルやシリアル回線を使用したり，TP(ツイスト・ペア；より対線)だけでなく光ファイバも使用したり，一つのルータやスイッチだけでなく複数のルータやスイッチを使用することで，各ノード間の物理的なルートを多重化して「必ずルートを確保」しておくことができます．

論理的なルートにおいて単一障害点を防止するには，TCP/IPの各階層におけるプロトコルを複数利用することです．各ノード間を複数のNICインターフェースでEthernetを張ったり束ねたりし，IPルーティングで複数のルートを確保することで，論理的なインフラの複数確保ができます．本書では，この第1章で取り扱っています．

冗長化，負荷/機能分散

クラスタ接続の冗長化や負荷/機能分散は，クライアントからの多数の接続要求をアプリケーション・レベルやインフラ・レベルで平均化することです．受け手を一つ(1ノード)に固定すると，そのノードが入力における単一障害点となるので，仮想的な受け手は一つであっても実際にはクラスタリング・システムとして複数の接続を受けつけることにより，単一障害点の防止と接続の分散ができます．

このような分散システムでは，クラスタリング・システムを構成する各ノード間の相

互監視・管理を行うHAの仕組みが必ず必要になります．これにより信頼性を高められますが，本書では第2章から第4章で取り扱っています．

シェアリング，ミラーリング，同期

クラスタリング・システム内でのデータやデバイスをシェアリング，ミラーリングあるいは同期することは，アプリケーション・システムとしての仕組みです．論理的なデータ・ファイルやディレクトリなど，物理的・論理的デバイスを含む対象オブジェクトについてシェアリングやミラーリングを行ってデータの信頼性を高めます．ネットワークRAIDやディレクトリ・システム，あるいはOS独自の手法により実現することになります．本書では，第5章で取り扱っています．

リソースやサービスの監視

ネットワーク上のリソースやサービスの監視は，OSによる各システム情報の提供と交換，主としてSNMPによるネットワーク・リソース情報やアプリケーション・サービス情報の提供と交換などを行って，ネットワーク全体のリソースとサービスの稼働状況を監視します．

SNMPはTCP/IPプロトコルのネットワーク管理プロトコルとして有名ですが，情報MIB(管理情報データベース)を使用して数多くの情報源を提供・交換することが可能です．また，サーバ・システムだけではなく，クライアント・システムやルータ，スイッチ，周辺装置などの多くのシステムに装備されて，広く利用されています．さらに，アプリケーション・システム独自の手法を使用してリソースやサービスの情報交換を行っているものもあります．本書では，第6章で取り扱っています．

その他，ネットワーク・システムを統合して運用管理する仕組みも数多くあります．本書では，第7章で一例を紹介しています．

こうしたさまざまな仕組みを，環境に合わせて複数適用することにより，クラスタリング・システムとしての信頼性の向上を図ることができます．

本書では，以降でこれらについて具体例を示しながら詳細に説明しています．

1.2 HA(ハイ・アベイラビリティ，Linux-HA Heartbeat)

Linux-HAプロジェクト(*1.2)では，HAをはじめとするRAS(Reliability，Availability，Serviceability)に対する取り組みを行っています．中でも，HeartbeatはLinuxばかりではなく，BSD UNIXやSolarisなどでも動作するクラスタリングによるHAの代表的なパッケージです．そこでここでは，このHeartbeatについて詳細に解説します．

Heartbeatは，複数ノードを使用したクラスタ・システムの環境下で，稼働中のノードの停止検出からフェールオーバ(fail over)，フェールバック(fail back)(*1.3)などを含むクラスタ管理を行い，クラスタ内の複数のシステムが一つの仮想システムとして稼動するのをサポートします．また，STONITH(ストーニス)(*1.4)と呼ばれる，障害ノードを強制的に停止する機能もあり，非正常と判断したクラスタ内のシステムを再起動させて，正常な状態に復帰させることができます．

Heartbeatには，バージョン1(以降，V1とも記述)とバージョン2(以降，V2とも記述)がありますが，本書ではHeartbeatバージョン2を使用します．バージョン2では，マルチノード(16ノードまでは検証済み)やCRM(クラスタ・リソース・マネージャ)，クラスタやリソースのさまざまな監視，複数のリソース･スクリプト，複数のフェールバック，障害ノードの強制停止など，豊富な機能がサポートされています．

なお，本章ではHeartbeatのリソース・オブジェクト(適用対象)として，仮想的なクラスタIPアドレスとWWWサーバ・アプリケーションを題材にして解説していますが，Heartbeatはこれらを含むさまざまなリソースに対してHA機能を提供するばかりでなく，ほかのHAツールと組み合わせて，さらに高度かつ応用範囲の広いRAS機能を高めることができます．例えば，LVS(Linux Virtual Server)やDRBD(Distributed Replicated

(*1.1) 本書では，Heartbeatは全体やシステム，一般名称など，HAプロジェクト記述に準拠する場合に使用し，heartbeatは具体的なプログラムやプロセスなど，特定のものに使用している．

(*1.2) http://linux-ha.org/．(日本語)http://linux-ha.org/ja/HomePage_ja

(*1.3) フェールオーバは，稼動中のノードが何らかの原因で停止したとき，他のノードが処理中のタスク(サービス)を引き継ぐこと．フェールバックは，元々動作していたサーバが停止状態から復帰した際に，改めて処理を引き継ぐこと．

(*1.4) Shoot The Other Node In The Head．1.2.5項を参照．

(*1.5) 「Linux高信頼サーバ構築ガイド-シングルサーバ編」で使用したシステムを本書「クラスタリング編」で続けて使用する場合は，「シングルサーバ編」での設定のいくつかは停止または変更しておくほうがわかりやすい．

- watchdogを止めておく　：chkconfig watchdog off
- daemontoolsを止めておく：/etc/inittabの最後の行
  →##STOP##SV:123456:respawn:/command/svscanboot　(ブート時のsvscan起動)
- daemontoolsで起動するサーバ･アプリケーションを元に戻す
  apache：chkconfigで自動起動設定する→chkconfig --level 35 httpd on
  (ただし，Apacheのクラスタ操作を行う場合にはoffにする)
- セキュアOS-SELinux　：ブート時パラメータ→selinux=0

Block Device)などです．それらについては，以降のそれぞれの章で解説します．

表1.1に，Heartbeatで使用する技術用語を簡単にまとめます．

表1.1　Heartbeatで使用する技術用語

用　語	意　味
クラスタ	複数のシステムを相互に接続して仮想的に1台のシステムとして動作する仕組み
(クラスタ)ノード	クラスタリングを実行するシステム
クラスタ・サイズ	クラスタを構成するノード数
CIB(Cluster Information Base)	クラスタ情報ベース クラスタの構成や動作，サービスなどを記述した設定ファイル
ハートビート	各ノードの相互接続性を確認するために送受信されるheartbeatパケット
リソース	Heartbeatの対象となるアプリケーションやサービス，IPアドレスなど
リソース・グループ	グループ化された複数のリソース
RA(Resource Agent)	リソース・エージェント CIBに記述されたリソースに対して，起動や停止などの動作を記述したスクリプト，あるいはその実行主体
ネイティブ・リソース	クラスタ内の特定のリソース・エージェントのうちの既成のもので，「class(クラス)」で識別する3種類のスクリプトがある (Heartbeat-V2，推奨順) • OCF(Open Cluster Framework) RA：OCFプロジェクト仕様(注1)に基づくもので，スクリプトは「/usr/lib/ocf/resource.d/」(Heartbeatの場合は，/usr/lib/ocf/resource.d/heartbeat)下に用意されている．RAへは「OCF_RESKEY_*スクリプトのパラメータ名*」というパラメータ名で渡される． なお，resource記述では，「class=OCF，provider=heartbeat，type=*スクリプト名*」となる． 以下のスクリプトがある． /usr/lib/ocf/resource.d/heartbeat AudibleAlarm　IPaddr　ManageVE　SysInfo　db2　oracle ClusterMon　IPaddr2　Pure-FTPd　VIPArip　drbd　oralsnr Delay　IPsrcaddr　Raid1　WAS　eDir88　pgsql Dummy　IPv6addr　SAPDatabase　WAS6　ids　pingd EvmsSCC　LVM　SAPInstance　WinPopup　iscsi　portblock Evmsd　LinuxSCSI　SendArp　Xen　ldirectord　rsyncd Filesystem　MailTo　ServeRAID　Xinetd　mysql　tomcat ICP　ManageRAID　Stateful　apache　o2cb • LSB(Linux Standard Base) RA：LSB仕様(注2)に基づくもので，「/etc/init.d/」下に用意されている．スクリプトのパラメータは指定できない． • Heartbeat RA heartbeatスクリプトで，「/etc/heartbeat/resource.d/」または「/etc/ha.d/resource.d/」の下に用意されている．パラメータはポジショナル($n)であり，named値=*ポジション番号*($nのn)となる．以下のスクリプトがある． /etc/heartbeat/resource.d/ なし /etc/ha.d/resource.d/ AudibleAlarm　IPaddr2　LinuxSCSI　ServeRAID　db2 Delay　IPsrcaddr　MailTo　WAS　hto-mapfuncs Filesystem　IPv6addr　OCF　WinPopup　ids ICP　LVM　Raid1　Xinetd　ldirectord IPaddr　LVSSyncDaemonSwap　SendArp　apache　portblock

CRM(Cluster Resource Manager)	クラスタ・リソース・マネージャ クラスタ・ノードのリソースの登録や変更などの管理を行うモジュール
フェールオーバ	クラスタ内の現在アクティブなノードが停止したときに別ノードに切り替わること
フェールバック	フェールオーバ後，フェールオーバ前の元のノードが再起動したときに元のノードに切り替わること
アクティブ／パッシブ	稼動／待機
仮想IPアドレス	クラスタ全体を指す仮想のIPアドレス
通信トポロジ	クラスタを構成するノード間の通信形態
構成ファイル	Heartbeatの以下の3種類の設定ファイル /etc/ha.d/authkeys：クラスタ・メンバの認証で使用するメンバ・ノードの認証情報. /etc/ha.d/ha.cf：heartbeatの全般動作の構成・設定情報. /var/lib/heartbeat/crm/cib.xml：クラスタのサービスとデフォルト・オーナの指定を行う，リソースの構成・設定情報
DC (Designated Coordinator)	指定コーディネータ．各ノードのCRMデーモンの中のマスタでスレーブからの登録や変更などの要求に対応して，保持しているCIBの更新やそのコピーのスレーブへの配布を行う

(注1) http://www.opencf.org/cgi-bin/viewcvs.cgi/specs/ra/resource-agent-api.txt?rev=HEAD

(注2) http://www.linuxbase.org/spec/refspecs/LSB_3.0.0/LSB-Core-generic/LSB-Core-generic/iniscrptact.html

1.2.1　Heartbeatの稼働環境と特徴

Heartbeatは，**図1.1**に示すように，一つのネットワーク・セグメント内で複数ノードによるクラスタ・システムを実現します．そして，そのクラスタ・システムにおいてもっとも基本的なHAインフラ・サービスとして動作します．

Heartbeatの基本的な機能や特徴は，以下のようなものです．

(1) 複数ノード間で，システム全体やアプリケーションの稼働状況の監視と動作制御を行う．

(2) 稼働監視は，システムで使用しているインターフェースとは別ルート，別メディアでも可能である(*1.6)．

(3) システムやアプリケーション，IPアドレスなどのリソースを管理する．

(4) リソースごとにRAを使用する．

(5) RAの設定はCIB設定ファイルで行い，CRMによって管理される．

(6) RAは3種類のスクリプトのいずれかを使用して動作する．

(*1.6) 通信方法で単一障害点(Single Point of Failure；SPOF．アプリケーションとHeartbeatが使用するコンポーネントが同じ場合に，一つの障害でシステム全体が障害となること)を防ぐために重要である．単一障害点を回避することは，HAの基本機能である．別メディアとして，LANやシリアル回線などを選択する．なお，LANを選択した場合，複数のNICが同じTCP/IPスタックである，IPパケット・フィルタ(ファイアウォール：UDP/694ポート発着信許可)の影響を受けやすい(一つのミスや設定が二つのNICに同じような影響を与えやすい)などの問題がある．そのため，シリアル回線を選択することが推奨される．このシリアル回線は高速であることが望ましい．

(7) DRBD(*1.7)やLVS(*1.8)などとも連携する．

なお，本章で確認するリソースは，IPアドレスとWWWサーバ・アプリケーションの2種類だけですが，RAスクリプトを使用してさまざまなリソースに対処することができます．

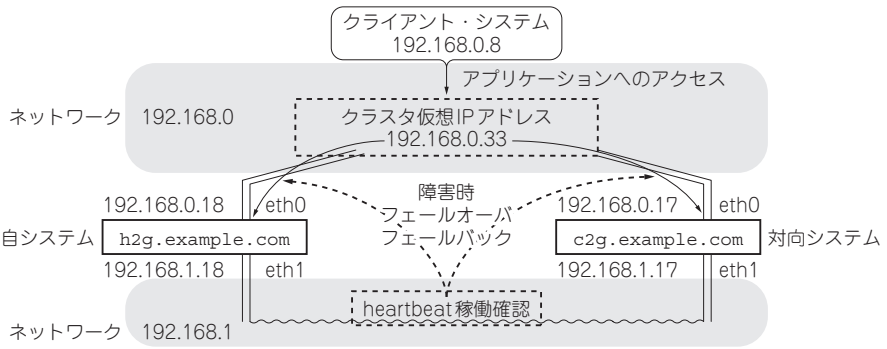

図1.1 クラスタ・ノードのフェールオーバ/フェールバック環境図

1.2.2 インストールと障害時の自動フェールオーバ/フェールバック動作の確認

インストールを行うパッケージは，以下の五つです．

heartbeat	：Heartbeat本体
heartbeat-gui	：GUIクライアント
heartbeat-stonith	：STONITH(障害ノードを強制停止)
heartbeat-ldirectord	：LVSのサービスを監視するデーモン
heartbeat-pils(*1.9)	：リソースなどのプラグインのためのサービス

なお，ここで示す設定は，単一の仮想IPアドレスでクラスタを利用する，ノード全体のシステムとしてのフェールオーバおよびフェールバックを自動的に行う設定です．つまり，単一IPアドレスをクラスタ・ノード間で切り替える(フェールオーバおよびフェールバック)処理の例で，さまざまなアプリケーションを含むシステム全体を切り替える設定です．

また，ネットワーク構成は，**図1.1**のように二つのシステム(*1.10)がアプリケーション用のネットワーク(192.168.0)とHeartbeat稼働確認用のネットワーク(192.168.1)に接続

(*1.7) DRBD；Distributed Replicated Block Device．分散デバイス・ミラーリング．

(*1.8) LVS；Linux Virtual Server．Linux仮想サーバ．

(*1.9) Plug-in and Interface Loading System.

(*1.10) 二つのシステム(h2g.example.comとc2g.example.com)は，それぞれ一つのNICに対する一つのクラスタ仮想IPアドレス(192.168.0.33)でフェールオーバ/フェールバックを行い，別のNICでHeartbeatの稼働確認を行う．

2.1 フロントエンド・システムPound

Pound(*2.1)は，スイス・チューリッヒの小企業Apsis GmbHが開発した，リバース・プロクシ(不特定多数のクライアントからの要求に対し，特定サーバの負担軽減やアクセス制限をするために設置されるプロクシ・サーバ)やロード・バランサ(負荷分散)，SSLラッパ(クライアントとHTTPS通信を行う機能)などの機能を含む，httpdサーバのフロントエンド・システムです(**図2.1**)．

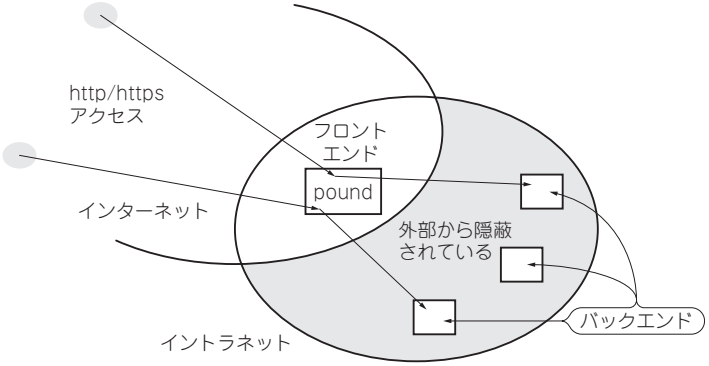

図2.1　Pound(http/httpsフロントエンド・システム)

2.1.1　既存フロント・システムの問題点とPoundの機能

ロード・バランサやプロクシ，リレー，フィルタリングなどのフロントエンド機能は，Apacheなどのアプリケーション・サーバやSquid，各種ラッパ(stunnelやtcp_wrappersなど)，スーパ・サーバなどを使用して，多機能かつ多様に実現することができます．

しかし，多機能ゆえに複雑であり，セキュリティやトラブルの穴も多く，ボトルネックにもなりやすくなります(**図2.2**)．

(*2.1) http://www.apsis.ch/pound/index_html
本書では，パッケージなどの名前の場合はPoundと大文字で記述し，プログラム/モジュール・ファイルなどの名前の場合にはpoundと小文字で記述する．

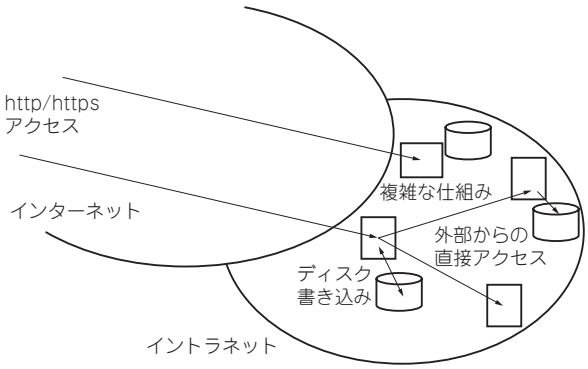

図2.2　フロントエンド・システムの一般的な仕組み

ここで取り上げるPoundは，そのようなフロントエンドとは大きく異なり，簡単なつくりでバックエンド・サーバに中継できるフロントエンド・システムです．

Poundのセキュリティ面では，特に特徴的なことがあります．多くのシステムのセキュリティ・ホール対策で中心となる問題は，「いかにしてファイル・システムへの不正な書き込みを阻止するか」ですが，Poundはファイル・システムへの書き込みを行いません．ログファイルもSYSLOGに依頼しています．

フロントエンド機能は，システムやネットワークが大きくなればなるほどボトルネックになりやすく，設定や動作のチェックに大きな労力が必要になります．また，セキュリティの脆弱性も大きくなります．

さらに，多くのフロントエンドは防護すべき対象のセキュリティ属性について制限はしていますが，自身内部に保持するので，その「手の内」が外部から見えやすくなっています．

以上のことを考慮すれば，簡単な設定でロード・バランサやフィルタ，リダイレクタ，リレーなどの機能を軽量に行い，防護すべき手の内を外部から隠蔽するPoundの役割を理解できるでしょう．

2.1.2　Poundの特徴

Poundのもっとも大きな特徴は，中継する宛て先のバックエンド・システム（httpdサーバ）を外部から隠蔽することです．外部からは，直接バックエンド・システムへアクセスする「道」がありません．

また，設定が簡単でありながら，ロード・バランサやフィルタ，リダイレクタ，セッション保持などの基本的な機能が備わっています．

さらに，ファイル・システムへの書き込みを行わないので，セキュリティ上の大きな危険性を排除しています．

Poundの特徴を列挙すると，以下のようになります（**図2.3**）．

(1) Poundは，自身のコンテンツやキャッシュを持たない(セキュリティ侵害の芽をつむ)
(2) 軽量かつ小さなプログラムである(ボトルネックを回避する)
(3) SSL起動時のSSL証明書の読み込み以外は，ディスクとはI/Oでつながない
(4) クライアント・アクセスのリバース・プロクシとして，外部から隠蔽されたバックエンド・サーバに対して中継する(リバース・プロクシ機能)
(5) クライアント・リクエストを，そのセッションを維持しながらバックエンド・サーバに負荷分散させる(ロード・バランサ機能)
(6) バックエンド・サーバのフェールを感知し，フェール時にはリカバリするまで中継を停止する(フェールオーバ機能)
(7) バックエンド・サーバとの接続状態を監視・保持しながら，クライアントとバックエンド間のHTTPデータ送受信を行う(HA機能)
(8) クライアント・リクエストのURLに対応したバックエンド・サーバに中継する
(9) HTTP/HTTPSリクエスト・フォーマット(RFC仕様準拠)を検証して適正なもののみ中継する
(10) HTTPのSSLラッパとして機能し，平文のバックエンドHTTPサーバに中継する
(11) バックエンド・サーバに中継(リダイレクト，分散)させる場合，さまざまな条件を設定することができる

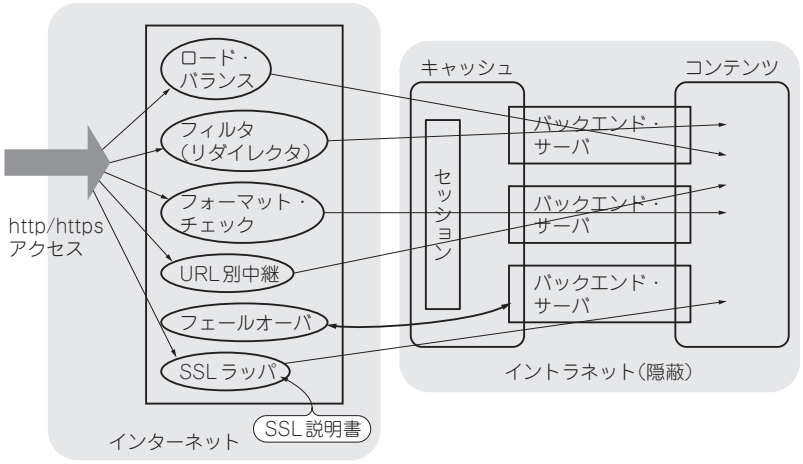

図2.3　Poundの機能

WWWサーバのVirtual HostとPoundのVirtual Host

Poundは，ある意味ではWWWサーバなどと同じようなVirtual Host機能を持っていますが，それらと決定的に違うのは，

(1) WWWサーバ(たとえばApache)の`httpd.conf`の「Virtual Host」設定は，1台のWWWサーバ上で複数の仮想ドメイン・サーバを実現する，いわば論理的なVirtual Hostである

(2) PoundのVirtual Host設定は，一つの入り口(プロクシ/リスナ)から複数のWWWサーバ上の個々の実ドメイン・サーバにアクセスを分散させる物理的な負荷分散Virtual Hostである

ということです．

コラム2.1　Poundで使用する用語

(1) セッション

poundは，クライアントとバックエンド・サーバとの接続を「セッション(session)」として保持している．

(2) オブジェクト

3種類の設定オブジェクトがある．

①リスナ(Listener)・オブジェクト

- クライアントからの受信要求の定義．HTTPリスナとHTTPSリスナの二つがある
- アドレスとポートのほか，HTTPS要件の指定が必要

②サービス(Service)・オブジェクト

- リクエストと応答の仕組み，およびセッション・メカニズムを定義
- リスナ・オブジェクト内またはグローバル(トップレベル)で指定される
- 受信リクエストのURLやヘッダに対応するバックエンドに中継される
- セッション(session)指定がある場合，そのクライアントからのそれ以降のリクエストは，常に同じバックエンドからの応答となる

③バックエンド・オブジェクト

- リクエストを受け付ける実際のWWWサーバ．pound自体は，バックエンドとの間を中継するだけである
- バックエンドの受け取り
- 三つのタイプのバックエンド
  - レギュラー(regular)：リクエスト-応答を行う
  - リダイレクト(redirect)：poundはバックエンドにアクセスすることなく，リダイレクト応答を返す
  - エマージェンシー(emergency)：ほかの全バックエンドが応答しないとき(dead)に使用するバックエンド
- 一つのservice内で複数のバックエンドを指定した場合には，それらの間でロード・バランスされる
- deadバックエンドは定期的に稼動しているかどうかをチェックし，応答があればpoundは回復したとしてリクエスト-応答を再開する
- どのバックエンドも応答しない(指定なし，またはdead)場合には，poundはほかのサービスをチェックせずに，“503 Service Unavailable”を返す
- poundとクライアント間のプロトコルに関係なく，poundとバックエンドの間はHTTPで行われる

3.1 HA/負荷分散クラスタリング・インフラ LVS(Linux Virtual Server)

Linuxシステムにおける負荷分散ソリューションの一種であるLVS(Linux Virtual Server)(*3.1)は，前章のPoundと同様な負荷分散機能をインフラ・レベル(OSI階層のレイヤ4)で持っています．さらに，信頼性を高めるために二つのバーチャル・フロントエンド(アクティブとバックアップの二つがあり，これをLVSルータと呼んでいる)間で相互の稼働監視サービス，およびフロントエンドから実際のバックエンド(複数の実サーバ)への稼働監視サービスを提供することにより，HA(High Availability，高可用性)を実現しています．

すなわち，LVSを構成している稼働監視(ハートビート)機構により，LVSルータのフォルト時のフェールオーバ(LVSルータ間)や，障害のあるリアル・サーバの切り離しと正常時の復帰という，LVSクラスタ全体における負荷分散とHA機能を実現し，持続的に信頼性のあるLVSバーチャル・クラスタリングを行います．

ハートビートは，LVS自体のものを使用する以外に，第1章で解説したHeartbeatやKeepalived(*3.2)，Ultra Monkey(*3.3)などのHAサービス・プログラムを使用しても実現が可能です．こうした，LVSと別のHAサービスを組み合わせて，「HA/負荷分散ク

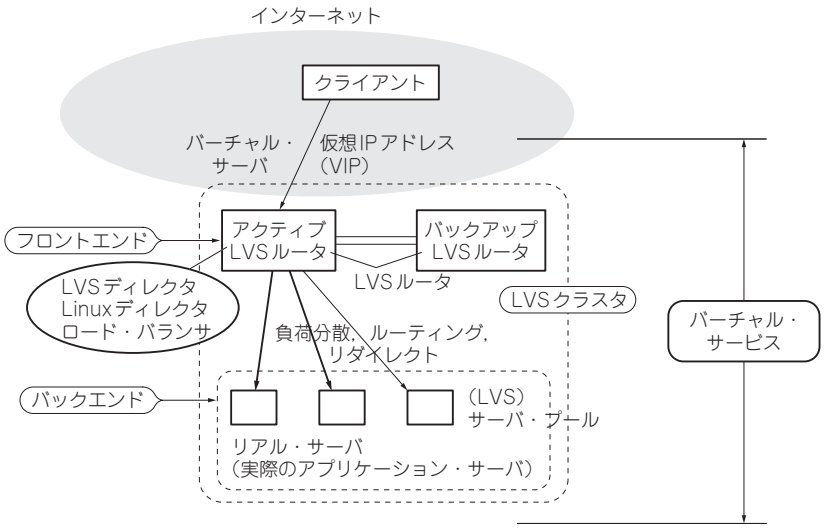

図3.1　LVSの構成

(*3.1) http://www.linux-vs.org/
(*3.2) http://www.keepalived.org/
(*3.3) http://www.ultramonkey.org/

ラスタ」システムを構築する方法については，第4章で解説します．

なお，LVSではバーチャル・サービスとして，HTTPやHTTPSだけでなく，SMTPやFTP，POP3など，さまざまなアプリケーションに適用できます．

本章では，LVSクラスタリングについて，その物理的・論理的構成，負荷分散のスケジューリング，パケット転送(ルーティング)方式，LVSバーチャル・クラスタの設定と実行の例，そして，LVSのためのHA手法について解説しています．

なお，LVSルータやリアル・サーバの全体を「LVSクラスタ」，リアル・サーバ全体を「(LVS)サーバ・プール」，アクティブLVSルータをプライマリ(またはマスタ)LVSルータ，あるいは「ディレクタ」(LVSディレクタ，Linuxディレクタ，ロード・バランサなど)とも呼びます(**図3.1**)．

CentOS 5では，LVSはLinuxカーネルの中に組み込まれているので，本章の説明でもこれをそのまま使用します．

ドキュメントについては，(*3.4)と(*3.5)を主に参考にしています．

3.2 LVSの物理的構成

LVSの利用環境は，**図3.2**に示すようなものになります．外部ネットワークとのフロントエンドに位置するアクティブLVSルータは，クライアントからのアクセス・リクエストを内部のリアル・サーバ(実サーバ)に送ります．

LVSルータが受け付けたIPアドレスは，仮想(バーチャル)IPアドレス(VIP，実際のLVSルータが切り替わるので「浮動IPアドレス」とも呼ばれる)であり，アクティブ・ルータがフォルトのときにはハートビートによる監視で検出された後，ARPリフレッシュされて，バックアップのLVSルータに継承されます．

アクティブLVSルータのバーチャル・サービスの主要な機能は，リアル・サーバ間の負荷分散とリアル・サーバ上で稼動しているサービスの稼働状況の監視の二つです．LVSの負荷分散は，「OSI7階層モデルのレイヤ4」スイッチングということもできます．さらに，LVSルータ間でのハートビートとフェールオーバも主要な機能です．

負荷分散を行うアルゴリズムには八つの方式があり，詳細は「3.4 負荷分散スケジューリング」で説明しています．

(*3.4) Linux Virtual Server Administration
http://www.centos.org/docs/5/html/5.2/Virtual_Server_Administration/index.html
http://www.centos.org/docs//5/html/5.2/pdf/Virtual_Server_Administration.pdf

(*3.5) Red Hat Linux Advanced Server インストールガイド
http://www.jp.redhat.com/manual/DocAS21/RH-DOCS/rhl-ig-as-x86-ja-2.1/index.html

アクティブLVSルータは，クライアントからのリクエストを対応するサービスを行うリアル・サーバに転送しますが，そのサービスはTCPあるいはUDP上のアプリケーション・サービスです．

アクティブLVSルータからリアル・サーバ・プールへのデータ・ルーティング，つまりパケット転送方式には，NAT(Network Address Translator，ネットワーク・アドレス変換機能)，トンネル，ダイレクト・ルーティングの3種類があります．詳細は，「3.5 パケット転送方式」で説明しています．

なお，LVSルータの物理的インターフェースにエリアス・アドレスを割り当てて，それぞれ個別のサービス(HTTPやFTPなど)を提供することも可能です．

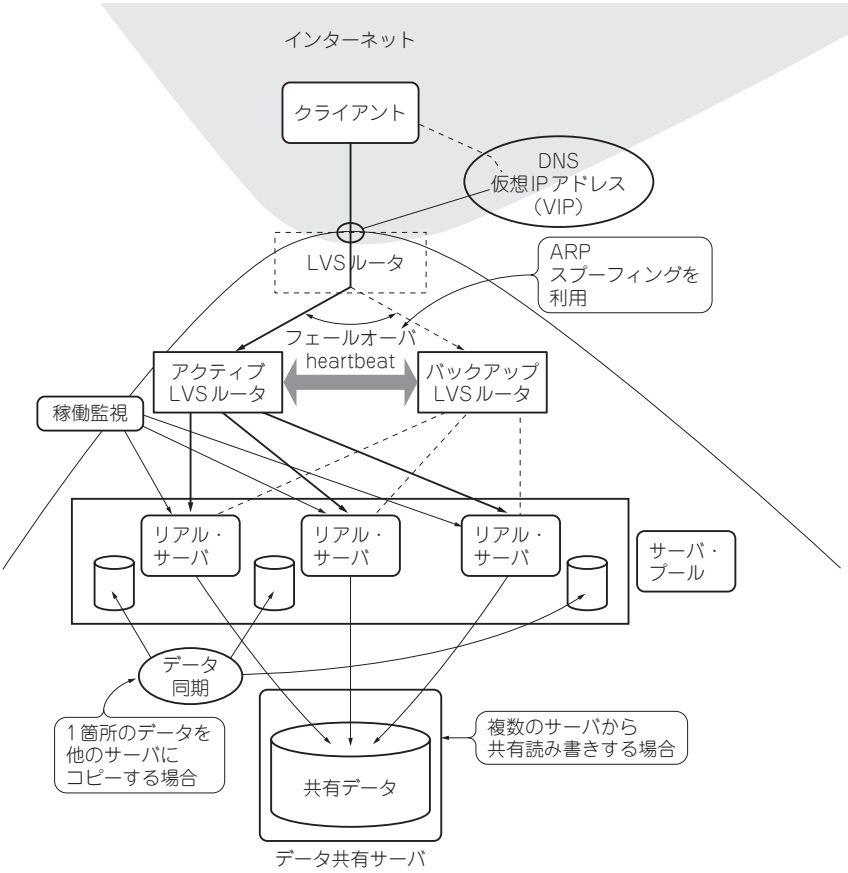

図3.2　LVSの利用環境

3.2.1　リアル・サーバ間でのデータの複製と共有

さらに，LVSのための付随的な機能として，リアル・サーバ間のデータの複製と共有が考えられます(**図3.3**)．ただし，これらの機能はLVSには組み込まれていないので，別のパッケージを利用する必要があります．

一つのリアル・サーバで更新されたデータを，サーバ・プール内の他のリアル・サーバ上で複製反映させるには，管理者がシェル・スクリプトを作成して送信したり，自動的にrsync(第5章の5.1節参照)で同期を取るといった方法があります．

一方，データの共有を行う場合には，ネットワーク内に設置したデータ共有サーバに対してNFS(Network File System)やsambaなどを使用して利用することになります．

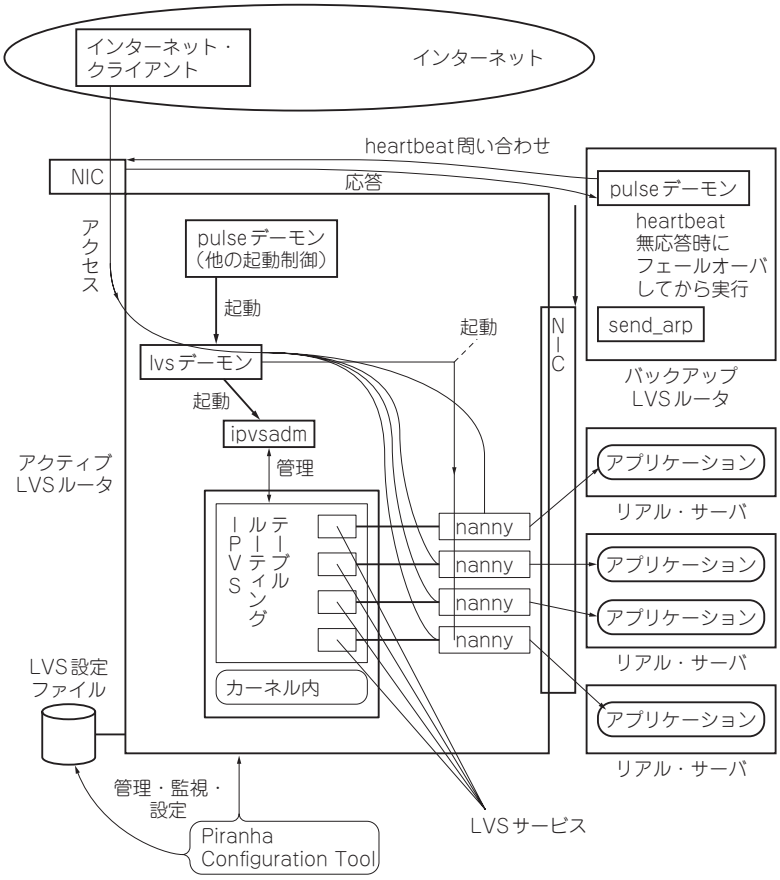

図3.3　LVSのコンポーネント構成

4.1 LVSをベースにしたその他のHA/負荷分散クラスタ

本章では，前章で紹介したLVS/HAクラスタ（HA機能付きLVSクラスタ）の中で，主にHA機能と操作性に重点を置いて，別の方法で実現するパッケージとその組み合わせについて解説します．

その代表的なものは，HeartbeatやPiranha（piranha-gui），Keepalived，Ultra Monkeyなどです．Piranhaは，第3章の手動による設定をWebベースで管理するツールです．その他のツールは，いずれもLVSディレクタ間のハートビート監視（キープアライブ）やLVSディレクタとリアル・サーバ間の稼働監視に，pulseやnannyなどのモジュールではなく専用のプログラム（パッケージ）を使用しています．

HA/負荷分散クラスタを実現するには，以下のような方法があります．

- ipvsadm + Heartbeat + mon
- Piranha
- Keepalived
- Ultra Monkey
- Heartbeat + mon + coda分散ファイル・システム
- Heartbeat + ldirectord

4.2 ipvsadm＋Heartbeat＋monによる方法

これは，IPVSルーティング・テーブルの直接設定はipvsadmで行い，LVSディレクタの監視/自動フェールオーバ/自動フェールバックはHeartbeatで，リアル・サーバ監視はmonで，という組み合わせにより行うLVS/HAです．

ここでのLVS/HAテスト実行環境は，以下のとおりです．

- 転送方式　　：LVS-NAT方式
- クライアント　　：192.168.3.12
- LVS仮想IPアドレス　　：192.168.3.181/eth1：0
- LVS-NAT内側IPアドレス：192.168.0.181/eth0：0
- LVSルータ　　：h2g（192.168.3.18/192.168.0.18）
  c2g（192.168.3.17/192.168.0.17）

- リアル・サーバ：f400（192.168.0.3），dgx（192.168.0.27）
  いずれもゲートウェイは192.168.0.181を設定
- クライアント　：192.168.3.12，Windowsインターネット・エクスプローラ/DOS-ftp
  ゲートウェイは，192.168.3.181を設定

なお，リアル・サーバではhttpサーバとftpサーバを起動し，クライアントからすぐに接続先を確認できるように，httpサーバでは個別のindex.htmlページを，ftpサーバでは個別のグリーティング・バナー（接続時の表示メッセージ）を，それぞれ表示するようにしています．実際の環境では，これらは全く同一にして，さらに第3章で紹介したツールなどを使って同期を取って作成・更新する必要があります．

4.2.1　準備――パッケージの導入と設定

最初に，Heartbeatの導入手順と設定方法，ipvsadmシェル・スクリプトの作成方法，monの導入手順と設定方法について説明します．

Heartbeatの導入手順と設定方法

LVSディレクタ間の監視と，自動フェールオーバおよび自動フェールバックを行うHeartbeatの導入と設定および実行を行う方法は，第1章の1.2節で説明しているので，ここでは本節で必要となるCIB設定ファイルの作成と設定，およびHeartbeatの設定ファイルについて説明します．

リスト4.1に示すように，h2g上で置き換え（Replace）用CIB設定ファイル（**リスト4.2**）を作成してから，h2gおよびc2g上で，/var/lib/heartbeat内の初期CIB関連設定ファイルを削除し，heartbeat（初期化）を起動した後，heartbeatの状態を表示して，両LVSディレクタがオンライン状態になったのを確認してから，CIB設定ファイルを置き換えて更新します．さらに，heartbeatの状態を表示して，仮想IPアドレスの起動を確認し，⑧ifconfigで，LVS-NAT両側の仮想IPアドレスの設定確認を行います．eth0:0が192.168.0.181で，eth1:0が192.168.3.181となっています．

リスト4.1　heartbeatの事前設定および起動

```
[root@h2g ~]# vi /tmp/cib_tmp2.xml              ①置き換え用CIB設定ファイル（リスト4.2）の作成

[root@h2g ~]# rm -f /var/lib/heartbeat/* /var/lib/heartbeat/*/*
                                                ②初期CIB関連設定ファイルを削除
rm: cannot remove `/var/lib/heartbeat/cores': ディレクトリです
rm: cannot remove `/var/lib/heartbeat/crm': ディレクトリです
rm: cannot remove `/var/lib/heartbeat/pengine': ディレクトリです
rm: cannot remove `/var/lib/heartbeat/cores/hacluster': ディレクトリです
rm: cannot remove `/var/lib/heartbeat/cores/nobody': ディレクトリです
```

```
rm: cannot remove `/var/lib/heartbeat/cores/root': ディレクトリです
[root@h2g ~]#
[root@h2g ~]# service heartbeat start              ③heartbeat起動（初期化）
Starting High-Availability services:
[  OK  ]
[root@h2g ~]#

しばらく，heartbeatの起動待ち

[root@h2g ~]# crm_mon                              ④heartbeat状態表示
Defaulting to one-shot mode
You need to have curses available at compile time to enable console mode
============
Last updated: Thu Jun  4 18:13:26 2009
Current DC: h2g.example.com (ea0325b7-b2c3-41bb-a405-bcd25f84c8c5)
2 Nodes configured.
0 Resources configured.
============
Node: c2g.example.com (9bbe03e2-89b6-43d1-b705-699288cfdaba): online
Node: h2g.example.com (ea0325b7-b2c3-41bb-a405-bcd25f84c8c5): online
[root@h2g ~]# ls -al /var/lib/heartbeat/crm
合計 24
drwxr-x--- 2 hacluster haclient 4096  6月  4 18:10 .
drwxr-xr-x 5 root      root     4096  6月  4 18:09 ..
-rw------- 2 hacluster haclient  832  6月  4 18:10 cib.xml
-rw------- 2 hacluster haclient  832  6月  4 18:10 cib.xml.last
-rw-r--r-- 2 hacluster haclient   32  6月  4 18:10 cib.xml.sig
-rw-r--r-- 2 hacluster haclient   32  6月  4 18:10 cib.xml.sig.last

[root@h2g ~]# cibadmin -R -x /tmp/cib_tmp2.xml     ⑤CIB設定ファイルの置き換え更新
[root@h2g ~]# ls -al /var/lib/heartbeat/crm
合計 24
drwxr-x--- 2 hacluster haclient 4096  6月  4 18:15 .
drwxr-xr-x 5 root      root     4096  6月  4 18:09 ..
-rw------- 2 hacluster haclient 2283  6月  4 18:15 cib.xml
-rw------- 2 hacluster haclient 2283  6月  4 18:15 cib.xml.last
-rw-r--r-- 2 hacluster haclient   32  6月  4 18:15 cib.xml.sig
-rw-r--r-- 2 hacluster haclient   32  6月  4 18:15 cib.xml.sig.last
[root@h2g ~]# crm_mon                              ⑥heartbeat状態表示
Defaulting to one-shot mode
You need to have curses available at compile time to enable console mode
============
Last updated: Thu Jun  4 18:16:33 2009
Current DC: h2g.example.com (ea0325b7-b2c3-41bb-a405-bcd25f84c8c5)
2 Nodes configured.
1 Resources configured.
============
Node: c2g.example.com (9bbe03e2-89b6-43d1-b705-699288cfdaba): online
```

```
Node: h2g.example.com (ea0325b7-b2c3-41bb-a405-bcd25f84c8c5): online
Resource Group: ip_group
    ip_resource1    (heartbeat::ocf:IPaddr):    Started h2g.example.com
                                                ⑦仮想IPアドレス1起動
    ip_resource2    (heartbeat::ocf:IPaddr):    Started h2g.example.com
                                                  仮想IPアドレス2起動
[root@h2g ~]# ifconfig -a                       ⑧LVS-NAT両側の仮想IPアドレスの設定確認
eth0      Link encap:Ethernet  HWaddr 00:E0:18:EF:B8:E8
          inet addr:192.168.0.18  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::2e0:18ff:feef:b8e8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:53994 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48759 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6235918 (5.9 MiB)  TX bytes:6536214 (6.2 MiB)
          Interrupt:217
eth0:0    Link encap:Ethernet  HWaddr 00:E0:18:EF:B8:E8
          inet addr:192.168.0.181  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:217
eth1      Link encap:Ethernet  HWaddr 00:04:2E:01:EB:0E
          inet addr:192.168.3.18  Bcast:192.168.3.255  Mask:255.255.255.0
          inet6 addr: fe80::204:2eff:fe01:eb0e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:320 errors:0 dropped:0 overruns:0 frame:0
          TX packets:818 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:95959 (93.7 KiB)  TX bytes:205559 (200.7 KiB)
          Interrupt:209 Base address:0xc000
eth1:0    Link encap:Ethernet  HWaddr 00:04:2E:01:EB:0E
          inet addr:192.168.3.181  Bcast:192.168.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:209 Base address:0xc000
…(以下省略)…
```

リスト4.2の置き換え用CIB設定ファイル(cib_tmp2.xml)は，第1章の**リスト1.5**をベースに作成しています．

この設定要件は，以下のとおりです．

- 192.168.3.181をeth1に，192.168.0.181をeth0に，それぞれ仮想IPアドレス生成
- これをh2g(優先：スコア＝200)とc2g(スコア＝100)でフェールオーバ，フェールバック

詳細は，①二つのIPリソース(「“ip_resource1”と“ip_resource2”」をグループ化して，②IPリソース1を，“192.168.3.181”(③LVS-NAT外側仮想IPアドレス)と使用インターフェース(④NIC=eth1)で設定し，⑤IPリソース2を，“192.168.0.181”(⑥LVA-NAT内側仮想IPアドレス)と使用インターフェース(⑦NIC=eth1)で設定しています．

5.1 データ・ファイル同期コマンド rsync

rsync(*5.1)はrcpやrshなどのように，BSD系のrコマンドの一つですが，リモート・システム間の処理を行うほかのrコマンドとは異なり，データの送受信だけでなく「同期」の機能が付け加えられています．ここでいう同期とは，「二つのシステム間のデータ・ファイルの違いをなくす」ことです．つまり，お互いに全く同じ(情報の)データ・ファイル・システムを持つことになります．

さらに，rsyncの大きな特徴として，両端システムのデータ・ファイルの差異だけを送るという「rsyncアルゴリズム」を使用することにより，高速・高効率な送受信を可能にしています．これが，他のftpやrcp，scpなどと異なるところです．

5.1.1 rsyncの特徴

rsyncには，以下のような長所と短所があります．

●長所

- ディレクトリ・ツリーやファイル・システム全体の更新が可能
- オプションとして，シンボリック/ハード・リンク，ファイル所有者，パーミッション，デバイスや時間などの属性情報を保持可能
- インストールに特別な権限が不要
- 内部パイプライン処理により複数ファイルの待ち時間を短縮可能
- 転送インフラにrsh，sshトンネル，そして，TCPソケットを使用可能
- ミラーリング用に匿名rsyncをサポート
- 低速リンクでも高速・高効率転送が可能
- 信頼性が高い
- 同じrsync操作を何回行っても問題ない
- ファイルの小さな変更は小さな処理
- 簡単な操作設定
- apacheやftpなど，ユーザIDを含む属性特性を指定した「モジュール」による処理が可能
- ピア・ツー・ピアとデーモン(サーバ)モードのいずれも可能

(*5.1) メインサイト　http://rsync.samba.org/
日本語サイト　http://www.infoscience.co.jp/technical/rsync/index.html

●短所

- 未だ分析不能なバグが残っている
- プログラムもプロトコルも単一の非対話型1方向転送が前提
- 全転送関連ファイルのリストがメモリを消費するので，大規模ツリーでは足を引っ張る
- SSHパスワード認証では，操作のソケット生成のたびに問題が起こることがある
- 名前が変更されたファイルは処理されず，古いファイルが削除され，消去されたものから新規作成される
- エラー・メッセージがわかりにくい
- オプションがわかりにくい．「-a」オプションが一般的
- ディレクトリ名とその内容を区別する最後の「/」に注意が必要

●用途

- 一つの大きな魅力は，一つ起動すると後続がパイプライン処理で行われ，ラウンドトリップ遅延が小さいという点で，ストリーミングへの適用性がある

5.1.2　他の転送プログラムとの比較

データ・ファイルの転送プログラムにはrcpやrsh，あるいはftpなどがあります．しかし，rcpやrshはセキュリティ上(例えば，.rhostを指定するだけでパスワードなしで接続できる)の問題があり，ftpの場合にもSSHトンネル経由のパッシブ転送を使わなければセキュリティを確保できません．しかし，SSHトンネルを経由するftpパッシブ転送ではデータ・チャネルの制限(複数を使用するためには複数を設定するという限界)があります．さらに，rcpやftp，scpなどでは，「データ同期の失敗」(新しいものを古いものに置き換えてしまうなど)も起こりがちです．

また，SSHトンネルとNetBIOSゲートウェイによるファイル転送は，セキュリティ上はしっかりしていても，複雑な仕組みになってしまいます．

そんな中で，rsyncも転送インフラとしてrshを使用するとセキュリティ上の心配が出てきます．そこで，やはりユーザ認証鍵を使用したSSHインフラによるrsyncを使うのが良い方法といえます．

ここでは，SSHユーザ認証鍵の設定について説明しませんが，相手方で生成したプライベート鍵(公開鍵も同時に生成し，authorized_keysに格納しておく)を使用するとか，SSHバージョン2専用(DSA鍵使用)とするとか，パスワード認証を認めない，などの基本的な制限を施しておいて，SSHトンネル経由でrsync送受信を行い，rsyncのデータ同期の優位性を，セキュリティを確保したうえで発揮させます．

なお，rsync転送インフラのデフォルトはSSHです．また，ここでは，rshのセキュリティ上の脆弱性(.rhostした場合はパスワードなしでOK．それ以外でもパスワードだけで入れる)を考慮してrshdは入れていません．

6.1 ネットワーク・リソース監視（snmp＋mrtg, RRDtool, Cacti）

本節では，ネットワーク管理を行うsnmp（snmpエージェント）からネットワークのさまざまなリソース情報を取得して，グラフで可視化し確認するシステムについて解説します(*6.1)．

図6.1に示すように，ネットワーク・リソースの監視を受ける被監視側システムでは，リソース情報を吸い上げるsnmpd（snmpサーバ，snmpエージェント）が稼働しています．監視側では，mrtgやRRDtool，Cactiなどのsnmpを利用（snmpマネージャ）するシステムが被監視側システムのsnmpエージェントからリソースを取得してWWWブラウザ用のHTMLドキュメントを生成します．そして，これを端末からWWWブラウザで監視することになります．

The Multi Router Traffic Grapher（MRTG）は，snmpマネージャとして動作して，ネットワークの負荷を監視するツールです．MRTGは，現在のネットワークのトラフィックの状態を示すグラフィック・イメージを含むHTMLページを生成します．

RRDtool（Round Robin Database tool）も同様ですが，RRDtool自体はrrdファイルを作成するだけで，関連するシェル・スクリプトやCGIプログラムなどでhtml文書化します．RRDtoolもmrtgも同じ作者（`http://tobi.oetiker.ch/hp/`）のものです．

Cactiはmrtgと同様に，システムのパフォーマンスやステータスなどの情報をグラフ化するツールです．

なお，これらのパッケージはCentOS release 5.3（Final）用を使用しています．

snmpを利用したネットワーク・リソース監視機能をインストールするには，まず監視側（snmpマネージャ．ここではh2g上）でsnmp関連のパッケージをインストールした後，被監視側（snmpエージェント，snmpサーバ．ここではc2g上）で，snmp関連のパッケージをインストールします．そして，被監視側でsnmpd設定ファイルを作成した後，snmp監視コマンド（snmpwalk）でネットワーク・リソースの監視をテストします．

(*6.1) 参照URL
- Net-SNMP　`http://net-snmp.sourceforge.net/`
  - Download　`http://www.net-snmp.org/download/`
    `ftp://ftp.net-snmp.org/pub/sourceforge/net-snmp/`
  - Web page　`http://www.net-snmp.org/`
  - Project Wik　`http://www.net-snmp.org/wiki/`
  - Sourceforge Project page　`http://sourceforge.net/projects/net-snmp`
- mrtg　`http://oss.oetiker.ch/mrtg/`
  - 日本語公式サイト　`http://www.mrtg.jp/`
- RRDtool　`http://oss.oetiker.ch/rrdtool/`
- Cacti　`http://www.cacti.net/`

その後，監視側でmrtgやRRDtool，Cactiなどのネットワーク・リソースの取得・表示・分析パッケージをインストールして，これらによりネットワーク・リソースの状況を確認します．

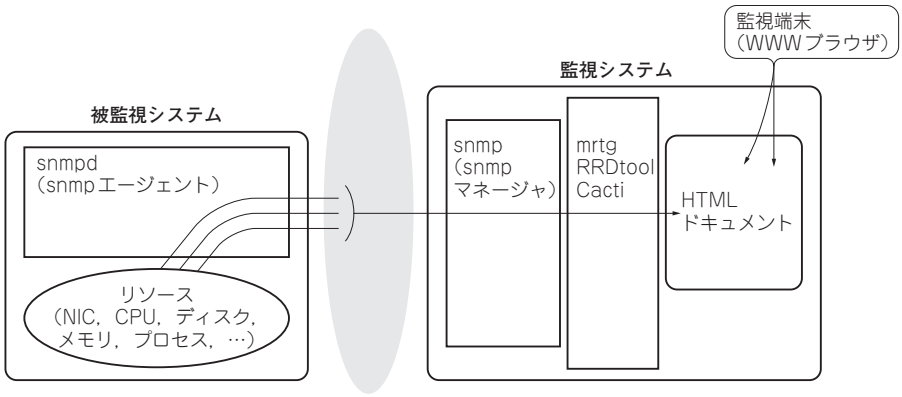

図6.1　snmp＋mrtg/RRDtool/Cactiによるネットワーク・リソースの監視

6.1.1　ネットワーク・リソース監視側へのsnmp＋mrtgのインストール

snmpを利用したネットワーク・リソース監視のインストールの手順を，**リスト6.1**に示します．まず，監視側(snmpマネージャ．ここではh2g)で，snmp関連のパッケージをインストールします．なお，ここではCacti/RRDtool関係のパッケージも一緒にインストールしています．

まず，①net-snmp関係のyumインストールを行います．snmpはNet-SNMPです．②でNet-SNMP(主snmp)の情報を確認した後，Cacti/RRDtoolのインストールを行いますが，これらは第4章で説明したmonと同様に，DAG/rpmforgeを利用してyumインストールするためのrpmforge-releaseパッケージをダウンロードした後，③yumインストールします．そして，④rpmforgeでCactiパッケージをyumインストールしますが，このとき，⑤のようにRRDtoolも一緒にインストールされます．その他，PHPやPHP-MySQL，PHP-SNMP関係も更新されます．

最後に，⑥Cacti情報を確認し，さらに⑦RRDtool情報の確認を行います．