LinuC

高信頼システム構築標準教科書

― 仮想化と高可用性 ―

（Ver.1.0.1）

LPI-JAPAN

■まえがき

このたび、特定非営利活動法人エルピーアイジャパンは、Linux/OSS 技術者教育に利用していただくことを目的とした教材、「高信頼システム構築標準教科書　― 仮想化と高可用性 ―」を開発し、Web 上にて公開し（URL: http://lpi.or.jp/linuxtext/ha.shtml ）、無償提供することとなりました。

この「高信頼システム構築標準教科書　― 仮想化と高可用性 ―」は、大手 IT ベンダーをはじめとする多くの企業からの、「Linux/OSS を使った高信頼システムを構築するための実践的なガイドブックが欲しい」という要望に応えて開発されました。

クラウドサービスやプライベートクラウドの利用が拡大する中、クラウド基盤をはじめとするミッションクリティカルシステムでの Linux/OSS のニーズはますます高まっています。中でもクラウド基盤構築の中核技術である『仮想化技術』と、サーバ間連携技術など信頼性の高いシステムを構築するための『高可用性技術』は、IT 技術者にとって必須の技術となっています。本教材は、このような技術の習得に役立つ実践的な内容となっています。

また、本教材は、クラウドエンジニアのスキルを認定する「LPIC レベル 3 304 試験（LPI-304 Virtualization & High Availability Exam）』の教育および学習にも役立てていただけます。

公開にあたっては、クリエイティブ・コモンズ・ライセンスに基づき、公開されています。

本教材は、最新の技術動向に対応するため、随時アップデートを行っていきます。

また、テキスト作成やアップデートについては、意見交換の Wiki サイトで、誰でもオープンに参加できます。

Wiki サイトの URL　http://www.lpi.or.jp/linuxtext/ha/

■執筆者・制作者紹介

恒川 裕康　株式会社デージーネット　代表取締役　（執筆）

高可用性サーバや仮想サーバは、様々な複雑な技術的要素を組み合わせて実現されています。そのため、こうしたサーバをきちんと作るためには、表面的な技術よりも、どのような仕組みで動作しているかという基礎的な知識をきちんと押さえる必要があります。本書は、こうした技術の「教科書」という位置づけです。技術の最新性や細部にこだわるのではなく、根底にある仕組みや考え方を重視し、それが理解しやすい素材を選んで取り上げたつもりです。この教科書が、強固なサーバを作ろうとしている方々の何らかの指針となれば幸いです。

＜略歴＞　1990 年台初めから、商用 UNIX の移植業務に携わり、UNIX/Linux を使った ISP などのネットワーク構築業務を行う。1995 年から UNIX/Linux を使った ISP などのネットワーク構築業務を行う。1999 年にデージーネットを設立し、代表取締役に就任。現在は、主に xSP へのコンサルティングなどを行う。

＜著書＞　「ネットワークサーバ構築ガイドシリーズ（共著）」（秀和システム 2002-2009）、「オープンソースでメシが食えるか!?―成功するシステム構築のための OSS 活用術」（秀和システム 2008）

松田 神一　（査読担当）

インターネットとそれを利用したサービスが、止まることの許されない重要な生活基盤となった現在、高信頼性システムを構築する技術は IT 技術者にとって非常に重要なものとなりました。この教科書では、高信頼性システムについての理論から実践まで数多くの実施例を含めて解説しているので、読んで理解し、そして実際に試してみることで技術を習得することができると思います。

鎌滝 雅久　OpenOffice.org 日本ユーザー会　理事長　（スタイル・レイアウト担当）

本教科書は、オープンソースのオフィススイート OpenOffice.org のワードプロセッサ機能を利用して作成しました。わたしは書式やレイアウトの管理を簡単に行えるスタイルを担当しました。OSS の普及に OpenOffice.org が一役買えれば幸いです。

木村 真之介　（編集協力、意見交換用 Wiki サイト作成）

エンジニアの皆様にとって高信頼システムは関心の高い分野かと思います。本教材を通じて皆様の学習のお役に立てれば幸いです。また、皆様の意見交換用の Wiki サイト(http://www.lpi.or.jp/linuxtext/ha/)を開設しました。本教材に関するご意見・ご質問・誤植等の報告はぜひ Wiki サイト(http://www.lpi.or.jp/linuxtext/ha/)にお寄せください。

伊本 貴士　（企画協力）

このテキストは、クラウド時代のエンジニアとして必要となる技術が一通り学べるように企画しました。このテキストを読む事で、クラウドや大規模なシステム構築を学ぶ最初のステップとしていただければと思います。また、オープンソースでここまで出来るんだと感じていただければ幸いです。

■著作権

■使用に関する権利

表示

本教材は、特定非営利活動法人エルピーアイジャパンに著作権が帰属するものであることを表示してください。

改変禁止

本教材は、改変せず使用してください。本教材に対する改変は、特定非営利活動法人エルピーアイジャパンまたは特定非営利活動法人エルピーアイジャパンが認める団体により行われています。

フィードバックは誰でも参加できるメーリングリストで行われていますので、積極的にご参加ください。

http://list.ospn.jp/mailman/listinfo/linux-text

非営利

本教材は、営利目的（※）以外で教材として自由に利用することができます。

教材として営利目的での利用は、特定非営利活動法人エルピーアイジャパンによる許諾が必要です。

本教材を利用した教育において、本教材自体の対価を請求しない場合は、営利目的の教育であっても基本的に使用できます。

その場合も含め、**LPI-Japan** 事務局までお気軽にお問い合わせください。

（※）営利目的の利用とは以下のとおり規定しております。

営利企業において、当教材の複製を用いた研修や講義を行うこと、または非営利団体において有料セミナー等に利用すること

本教材の使用に関するお問合せ先

特定非営利活動法人エルピーアイジャパン（LPI-Japan）事務局

〒102-0082　東京都千代田区一番町 15　一番町コート 6F

TEL：03-3261-3660

FAX：03-3261-3661

E-Mail：info@lpi.or.jp

■この教科書の目的

本書は、Linux および OSS を使って、高信頼システムあるいは仮想化環境を構築したい方を対象とした教科書です。

LPIC レベル 3 の 304 試験の学習範囲に含まれる、仮想化および高可用性についての知識を学習し、またそのようなシステムを実際に構築することを目的としています。

LPIC レベル 2 相当の知識がある方を前提としています。

■使用している環境

Linux には様々なディストリビューションが存在しますが、本書では CentOS5.5 を使用しています。

その他のディストリビューションでも、多くの内容がそのまま当てはまりますが、パス名、ファイル名や設定の方法などが一部、異なる場合があります。

各実施例で使用したソフトウェアは、執筆時点における最新のバージョンを使っていますが、その後の機能強化や変更により、ファイル名、コマンド名、使用方法、実行結果などが本書の記述と異なる場合があります。

■本書で用いる表記

本書に掲載されているコマンドの実行例において、プロンプトが'#'になっているのは root 権限で実行するもの、プロンプトが'$'になっているのはユーザ権限で実行するものです。

例)

```
# modprobe kvm                          ← root ユーザで実行
$ qemu-img create guest1.img 4GB    ← 一般ユーザで実行
```

本文中でファイル名やディレクトリ名を記載するとき、原則としてディレクトリ名(例えば/usr/bin/)にはパス名の最後に'/'を付加して表記し、ファイル名(例えば/usr/bin/ls)と区別しやすいようにしてあります。

目次

1章 高信頼システムの概要

高信頼システムについて学習するにあたり、その基礎的な知識や考え方を学習します。システムの信頼性を数値化する方法、信頼性を向上するための考え方、信頼性をチェックする監視の方法などについて学びます。

1.1　高信頼システムとは、どんなシステムか？

コンピュータは、複雑に電子機器や装置を組み合わせて作成された精密機器です。単独で働くだけでなく、複数のコンピュータや機器と組み合わせてシステムとして動作することもあります。

１つ１つのコンピュータでは、部品の故障や外部条件の変化などで、故障や停止の可能性があります。高信頼システムとは、こうしたコンピュータやアプリケーションを適切に組み合わせることで、通常よりも高い信頼性を確保したシステムのことです。

1.1.1　高信頼システムの必要性

従来は、学術研究や企業の会計などの限られた用途にしか利用されていなかったコンピュータですが、近年になって、社会の中の様々な分野で利用されるようになってきました。また、インターネットの普及に伴い、研究やビジネスだけでなく、コミュニケーションやエンターテイメントなどの幅広い分野で利用されるようになってきています。

こうした状況の中、用途によっては絶対に止まることが許されない重要なコンピュータシステムが存在します。例えば、金融、医療サービス、公共サービスなどで利用されているコンピュータが停止すれば、人命を含む大きな被害が出ることが予想されます。また、インターネット上のシステムのように多数の人が利用するサービスも、システムが停止すれば多くの人が不便な思いをしなければなりません。

一般の企業で使われている業務システムやコミュニケーションツールのメールシステムの場合でも、システムが停止すると業務が止まってしまうかもしれません。例えば、インターネットの通信販売システムが停止すれば、受注量に大きな影響が出る可能性もあり大きな損害となり得ます。このように、24 時間 365 日停止することができないコンピュータシステムの性格を**ミッションクリティカル**と呼びます。最近では、社会の中の至る所で、ミッションクリティカルなコンピュータシステムが必要となり、高信頼なシステムが求められるようになってきているのです。

このような技術が求められるもう 1 つの背景には、「コンピュータは止まりやすい」という現実があります。例えば、表 1-1 は 2006 年に米国のガートナー社が公表したデスクトップパソコンとノートパソコンの年間故障率です。これによれば、一般のデスクトップパソコンが 1 年間に故障する割合は約 5%もあり、さらに利用年数が長くなると故障率が高くなることが分かります。これは、20 台に 1 台のパソコンが何らかの問題で故障することを示しています。この年間故障率が 5%のパソコンを 20 台使ってシステムを作ると、確率的には、年に 1 度はシステムのどれかのパソコンが壊れることになります。一般に、サーバと呼ばれるコンピュータハードウェアは、もっと故障率は小さく 1%程度と言われていますが、それでも 100 台のコンピュータでシステムを作れば、年に 1 台程度が故障することになります。

▼表 1-1:デスクトップとノート・パソコンの平均年間故障率（単位：%）

	2005-06 年購入のシステム	**2003-04 年購入のシステム**
デスクトップ		
1 年	5	7
4 年	*12	15
ノート・パソコン		
1 年	15	20
4 年	*22	28

注記：*は予測
出典：Gartner 社 Dataquest（2006 年 6 月）

コンピュータシステムの故障は、実際にはハードウェアの故障だけが原因ではありません。次のような様々な要素があります。

- 外部要因（ほこり、温度変化など）による故障
- 停電などによる停止
- コンピュータの OS の不具合
- コンピュータ上のアプリケーションプログラムの不具合
- コンピュータ部品の不良による故障
- コンピュータ部品の耐用年数の経過による故障
- 地震や水害などの物理的環境要因
- オペレーションミスなどの人的ミスによる停止

どのような重要なシステムであっても、こうした壊れやすいコンピュータを利用することになります。そのため、システムの用途に応じて信頼性を考慮したシステムの設計が必要になるのです。

1.1.2　システムの稼働率

コンピュータシステムは、様々な原因で停止する可能性があります。そのため、システム全体の安全性を検討する場合には、単純なハードウェアの故障率だけでは考慮不足だと言われています。例えば、1 日に平均 24 万円（年間 8,760 万円）の売上を上げることのできるインターネット通信販売のシステムを作る場合を考えてみましょう。このシステムが故障して、復旧までに 3 日掛かった場合には、72 万円もの損害が出る可能性があります。しかし、復旧に半日しか掛からなければ、損害は 12 万円で済みます。このように、システムの安全性を検討する場合には、故障率だけではなく、障害から回復するために必要な時間（修理時間）も考慮する必要があるのです。

一般には、システムの安全性は、信頼性と保守性の両面から考える必要があると言われています。信頼性と保守性は、次のように定義されます。

- MTBF（Mean Time Between Failure）　～　システムが故障してから次に故障するまでの時間

$$MTBF = \frac{\text{稼働時間の合計}}{\text{故障回数}}$$

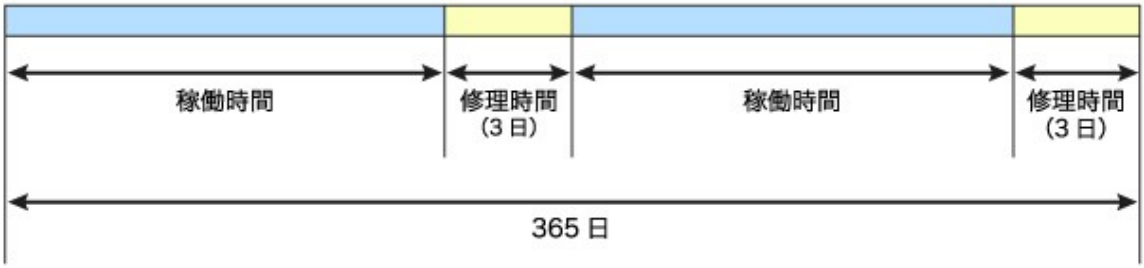
- MTTR(Mean Time To Repair）　～　故障したシステムの復旧に要する時間

$$MTTR = \frac{\text{修理時間の合計}}{\text{故障回数}}$$

これに対して、システム全体の安全性は稼働率と呼びます。稼働率は、次のような式で定義されます。

$$\text{稼働率} = \frac{MTBF}{MTBF + MTTR}$$

例 1-1:年（３６５日）に2回故障し、1回の修理に3日かかるシステム



稼働日数＝365−6＝359(日)
稼働時間＝359×24＝8616(時間)
稼働回数＝2(回)

$$MTBF = \frac{8616}{2} = 4308 \text{（時間）}$$

修理日数＝3+3＝6(日)
修理時間＝6×24＝144(時間)
修理回数＝2回

$$MTTR = \frac{144}{2} = 72 \text{（時間）}$$

したがって

$$\text{稼働率} = \frac{MTBF}{MTBF+MTTR} = \frac{4308}{4308+72} = 0.98356\ldots$$

稼働率は、約98.3%であることがわかります。

例 1-2:月（30 日）に 1 回故障し、1 回の修理に 1 時間かかるシステム

稼働時間＝30(日)×24−1＝719(時間)
稼働回数＝1(回)

$$MTBF = \frac{719}{1} = 719 \text{（時間）}$$

修理時間＝1(時間)
修理回数＝1(回)

$$MTTR = \frac{1}{1} = 1 \text{（時間）}$$

したがって

$$\text{稼働率} = \frac{MTBF}{MTBF+MTTR} = \frac{719}{719+1} = 0.99861\ldots$$

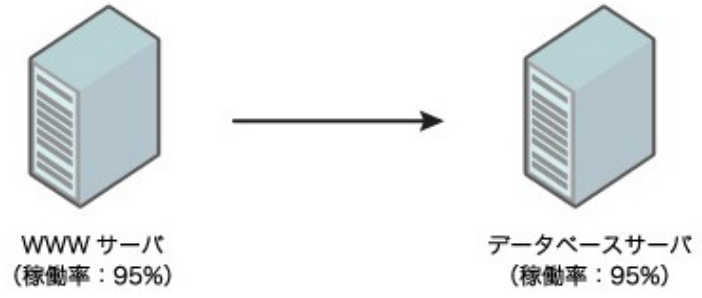
稼働率は、約 99.8% であることがわかります。

1.1.3　シンプレックスシステム

稼働率は、ある瞬間にそのコンピュータが動作している確率を示しています。複数のコンピュータを組み合わせてシステムを作った場合には、一般的な確率計算でシステムの稼働率を計算することができます。

例えば、図 1-1 のように稼働率 95%の WWW サーバと、稼働率 95%のデータベースサーバの 2 台を使ってシステムを作った場合を考えてみます。

図 1-1:シンプレックスシステムの例



WWW サーバとデータベースサーバのどちらが止まっても、システム全体の機能が維持できない場合には、システムの稼働率は各サーバの稼働率を乗じたものになります。この例では、次のように計算することができます。

稼働率＝WWW サーバの稼働率　×　データベースサーバの稼働率
＝0.95 × 0.95
= 0.9025

このように 95%の稼働率のサーバ 2 台を組み合わせた場合の稼働率は約 90%となり、各コンピュータの稼働率よりも低くなります。

このシステムは、システムを構成する要素の 1 つでも停止すると、全体の機能を維持することができません。このようなシステムを**シンプレックスシステム**と呼びます。

1.1.4　信頼性を向上させる手法

最近のコンピュータシステムは、たくさんの機能を組み合わせて構成することが多くなりました。そのため、役割に応じてサーバを 1 台だけ用意するシンプレックスシステムでは、機能が複雑になり、利用するコンピュータの台数が増えるほどシステムの稼働率が低下してしまいます。しかしそれでは困りますので、システムの信頼性を向上するためにいろいろな手法が使われています。

$$\text{稼働率} = \frac{MTBF}{MTBF + MTTR}$$

稼働率は、このような式で表されるわけですから、稼働率を上げるには MTBF（平均故障間隔）を長くする方法と、MTTR（平均修理時間）を短くする方法の 2 つの方法があります。

MTBF を長くする

稼働率をあげるために、もっとも単純な方法は MTBF を長くすることです。そのためには、ハードウェア、OS、アプリケーションなどのすべての部分に信頼性を向上するための工夫を行う必要があります。

ハードウェアの信頼性を少しでも高くするために、どのハードウェア部品に障害が発生した際にも停止することなく稼動を続けられるようにした **FT サーバ**（Fault Tolerant server）という特殊なサーバが利用されることがあります。FT サーバでは、電源、CPU、メモリ、ハードディスク、ネットワークインタフェース、PCI バス、などの部品をすべて多重化します。また各部品が故障しても無停止で交換できるように設計されています。ただし、FT サーバは、通常のサーバハードウェアに比べるとかなり高価です。

最近は、通常の PC サーバでも、様々な部品を二重化することができたり、稼働中に部品交換ができるようになってきています。また、コンピュータ部品の故障としては初期不良がもっとも多いという特徴に着目し、**エージング**という方法が使われることがあります。エージングは、「ならし運転」という意味で、一定期間の稼働を行い安定して動作することが確認できてから、本格的な利用を行おうという考え方です。

ハードウェアによる信頼性だけではなく、OS レベルやアプリケーションレベルでの信頼性も同時に考える必要があります。障害の発生原因はサーバハードウェアだけとは限らないため、FT サーバなどによってハードウェアの信頼性がほぼ 100%だとしても、サービスやシステムという視点で見たときに完全な信頼性があるわけではないからです。そのため、次のような様々な観点から OS やアプリケーションの信頼性を高める工夫を行う必要があります。

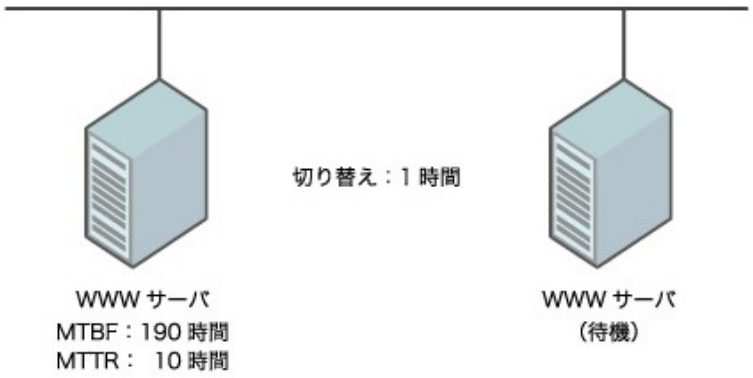
- 障害を考慮したアプリケーションの設計
- 開発段階でのソースコードレベルでのレビューの実施
- 開発したソフトウェアの詳細な試験の実施
- ソフトウェアやシステムのレベルで可能な障害対策の実施
- ソフトウェアを組み合わせてシステムを作成した後の機能試験の実施
- 限界時の動作を確認する負荷試験の実施

MTTR を短くする

FT サーバのような特殊で高価なハードウェアを使わない限り、ハードウェアの信頼性をある程度以上に向上することはできません。また、ハードウェア以外に起因する障害の可能性も決して 0 にはなりません。したがって、信頼性の高いシステムを設計しようとすれば、MTTR をできるだけ短くする努力も欠かせません。

もっとも単純に MTTR を短くする方法は、2 台のコンピュータをあらかじめ用意しておき、障害が起きたときに手動で切り替えるという方法です。切り替え用のサーバを用意し電源を入れずに待機させるのを**コールドスタンバイ**、いつでも切り替えができるように電源を入れた状態で待機させるのを**ホットスタンバイ**と呼びます。例えば、図 1-2 のように 2 つの WWW サーバを用意して片側でサービスを稼働させ、障害が起きたときにサービス稼働サーバを切り替えます。

図 1-2: アクティブ・スタンバイシステムの構成例



障害が起きたときに、待機しているサーバに切り替えれば、修理時間は切り替えの時間だけとなります。この切り替え時間を**フェイルオーバタイム**と呼びます。フェイルオーバータイムは通常の修理時間よりも短いため、MTTR を短縮することができます。フェイルオーバータイムが 1 時間と仮定すると、稼働率は次のようになります。

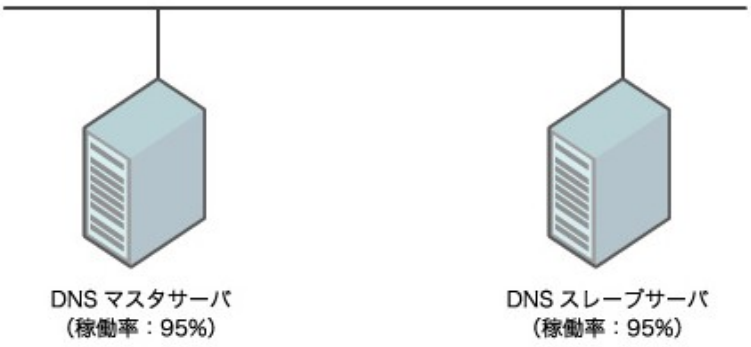
$$\text{稼働率} = \frac{MTBF}{MTBF + MTTR} = \frac{190}{1+190} = 0.9947..$$

稼働率は、95%から 99.5%へと大きく向上しました。このように、障害が発生したときの切り替え用の待機系サーバを用意したシステムのことを、デュプレックスシステムと呼びます。また、通常は 1 台が稼働（active）状態で、1 台は待機（standby）状態であることから、このようなシステムを**アクティ**

ブ・スタンバイシステムと呼ぶこともあります。

システムの設計や用途によっては、待機系サーバへの切り替えを瞬時に行うことができるものもあります。その例の 1 つが、DNS サービスです。DNS サービスは、マスタサーバとスレーブサーバを作ることで完全に二重化できるため、どちらのサーバが停止しても、全体としてはサービスを継続して提供することができます。

図 1-3:アクティブ・アクティブシステムの構成例



両方のサーバが同時に止まらない限りサービスを提供し続けることができますので、次のように稼働率を計算することができます。

稼働率＝1 ー（1 ー マスタサーバの稼働率）×　（1 ー スレーブサーバの稼働率）
　　＝1 ー (1 ー 0.95) × (1 ー 0.95)
　　＝0.9975

稼働率は約 99.8%となり、それぞれのサーバの稼働率よりも格段によくなります。このように、複数のコンピュータを並列に動作させておき、システムの障害が発生すると自動的に切り替わるシステムのことを**デュアルシステム**と呼びます。また、この例のように、両方のサーバが同じ役割を担っていて、どちらかが停止した場合には片系でサービスを継続できるシステムを、アクティブ・スタンバイシステムに対してア**クティブ・アクティブシステム**と呼ぶことがあります(図 1-3)。

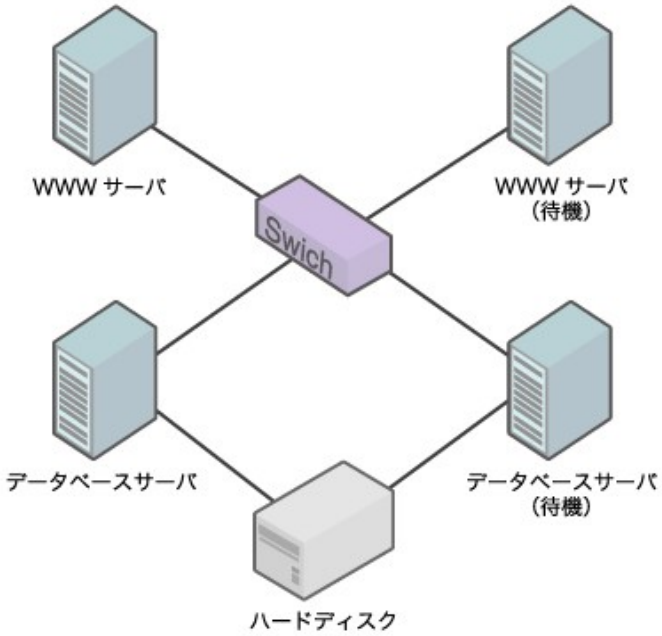
1.1.5　フォールトトレランス

デュプレックスシステムやデュアルシステムのように、システムのどのコンピュータが停止しても、継続してサービスが提供できるような機能や能力のことを、**フォールトトレランス（fault tolerance）**と呼びます。前述した FT サーバは、1 台のコンピュータの中ですべての部品を多重化することでフォールトトレランスを実現しようとしたサーバシステムです。

このような特殊で高価なハードウェアを使わなくても、2 台（あるいは複数）のコンピュータを使ってシステムの信頼性を高めることができます。これを**二重化**（あるいは**冗長化**）と呼びます。さらに、同じ機能のコンピュータを集めることを**クラスタリング**と呼び、クラスタリングにより作られたシステムを**クラスタ（クラスタシステム）**と呼びます。一般的には、二重化、冗長化、クラスタリングは、どれも同じような意味で使われています。

なお、複数の機能のコンピュータを組み合わせてフォールトトレランスを保ったシステムを作るためには、各機能をすべて冗長化する必要があることに注意してください。

図 1-4:単独障害点の例



例えば、図 1-4 のシステムは、WWW サーバもデータベースサーバもすべて二重化されていて、フォールトトレランスであるように見えます。しかし、これではシステム全体としてフォールトトレランスであるとは言えません。スイッチやハードディスクが故障した場合には、システム全体が止まってしまう可能性があるためです。

このように、故障するとシステムが止まってしまう欠陥箇所のことを、**単独障害点(Single point of failure)**と呼びます。フォールトトレランスを維持してシステムを作るには、このような単独障害点がないように設計しなければならないのです。

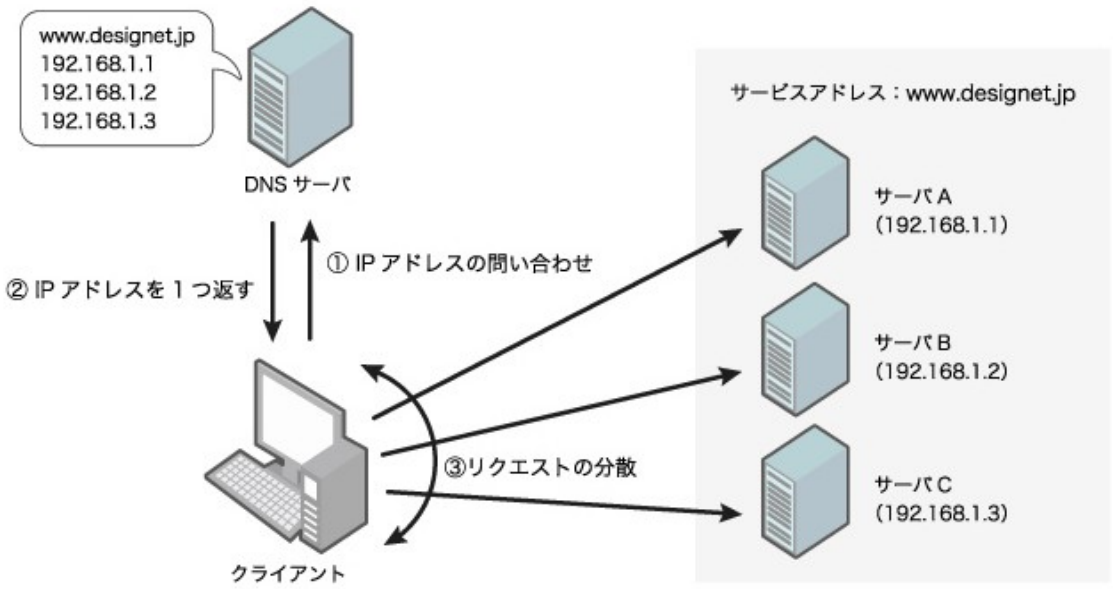
1.1.6　負荷分散構成(ロードシェアシステム)

システムの障害のうち、コンピュータの故障の次に多いのが、能力不足によるものです。例えば、一度に 100 人のユーザにしか対応できないシステムを 1000 人が利用しようとすれば、システムは正常に働かないでしょう。

このような状況を打開するために**負荷分散(ロードシェアリング)**という方法が使われます。負荷分散には、次のようないくつかの方法があります。

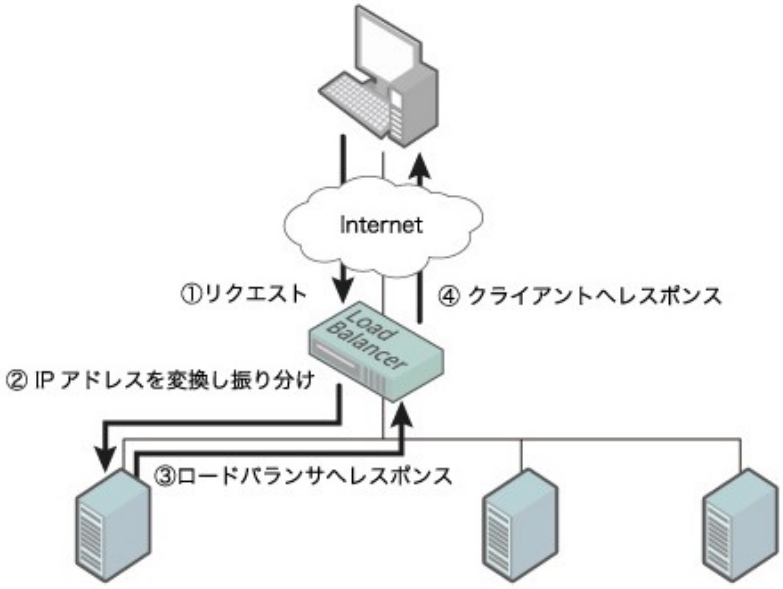
- DNS 型
  IP ネットワークでは、サービスを利用するときに DNS を参照することを利用して負荷分散を行います。クライアントは、実際のサービスを利用する前に DNS サーバへの問い合わせを行い、www.designet.jp のようなサービス名称から IP アドレスを調べます。このときに、DNS サーバが複数のサーバの IP アドレスの中から 1 つを選んで応答を返します。利用するサーバが動的に切り替わるため、リクエストを複数のサーバ間で分散することができます(図 1-5)。

図 1-5:DNS 型負荷分散



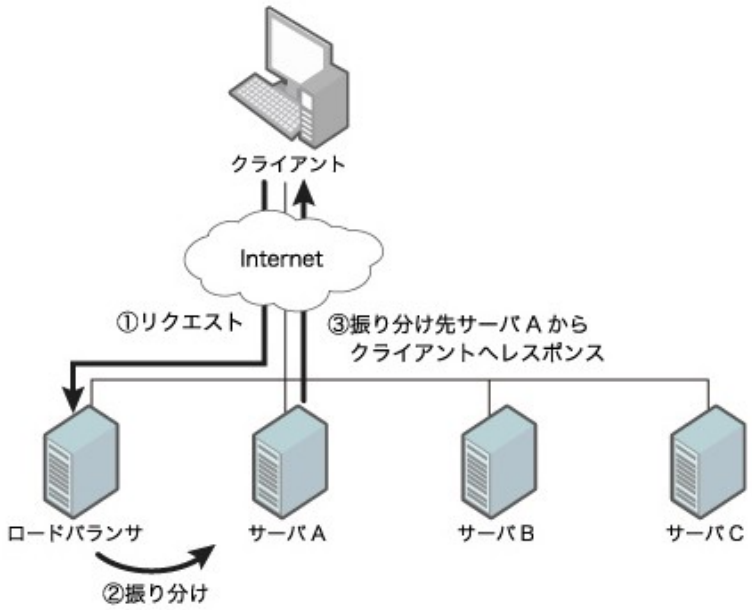
- アドレス変換（NAT）型
  リクエストを分散させる専用のサーバや機器（ロードバランサ）によって、リクエストを分散させる方式の 1 つです。クライアントは、ロードバランサに付けられた IP アドレス（代表 IP アドレス）に対してリクエストを送ります。ロードバランサがリクエストを受けとると、宛先の IP アドレスを処理を行うサーバのアドレスに変換してサーバに送付します。この方式では、すべての通信がロードバランサを経由することになるため、ロードバランサの処理性能に十分に注意する必要があります(図 1-6)。

図 1-6:図 1-1. アドレス変換（NAT）型負荷分散



- ダイレクトルーティング型
  アドレス変換型と同様にロードバランサを利用する負荷分散方式です。クライアントが、代表IP アドレスに対してリクエストを送ると、ロードバランサはリクエストをサーバに送付しますが、このときに IP アドレスではなく MAC アドレスを変更してサーバに送付します。この構成の場合には、クライアントからのリクエストはロードバランサを経由して届けられますが、レスポンスパケットはサーバからクライアントへ直接送られます。したがって、アドレス変換型に比べると、処理性能の点で有利です。ただし、各サーバに代表 IP アドレスを処理できるような設定がされていなければならなかったり、必ず全サーバが同一の物理ネットワーク内になければならないという制約があります(図 1-7)。

図 1-7: ダイレクトルーティング型負荷分散

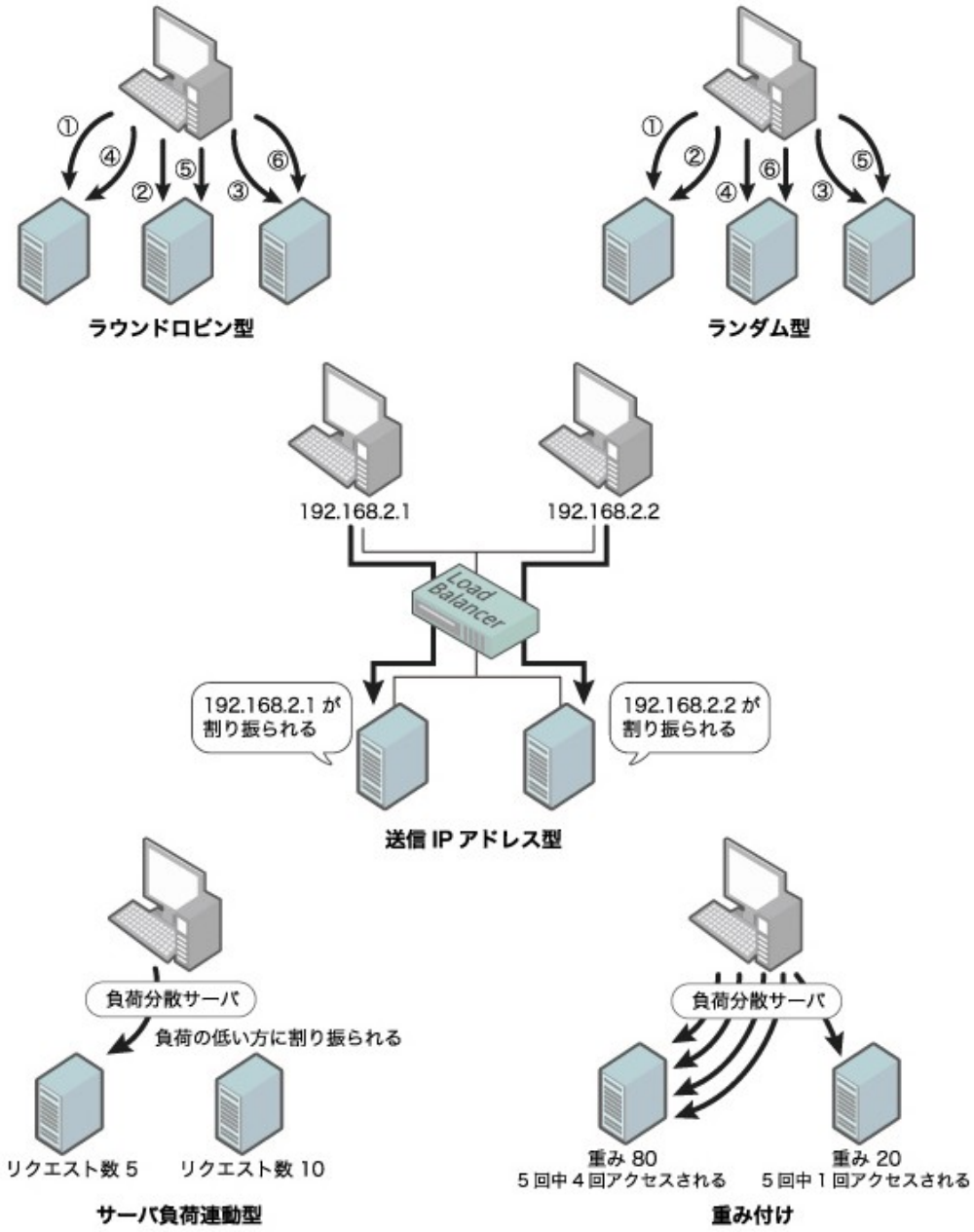


実際の負荷分散では、複数のサーバにどのようにリクエストを分散させるのかということも重要なポイントになります。主に次のような方式があります(図 1-8)。

- ラウンドロビン型
  1→2→3→1→2→3...というように、各サーバに順番にリクエストを割り振ります。
- ランダム型
  乱数を使って、完全にランダムにリクエストを割り振ります。
- 送信元 IP アドレス型
  送信元の IP アドレスにしたがってリクエストを割り振ります。同じクライアントが、必ず同じサーバを利用するような割り振りをしたい場合に使います。
- サーバ負荷連動型
  処理能力の余裕のあるサーバに割り振ります。サーバの負荷の計測方法にもいくつか種類があります。処理中のリクエスト数、CPU 利用率、ロードアベレージなどです。
- 重み付け

すべてのサーバの能力が同じでない場合に、能力に応じてリクエストが割り振られる頻度を調整する機能です。ラウンドロビン型、ランダム型、送信元 IP アドレス型などと組み合わせて使われます。

図 1-8: リクエスト分散のイメージ



なお、ロードバランサは専用機器として販売されているものが一般的ですが、Linux サーバで構築することもできます。Linux サーバで構築した場合には、必ずしも専用のシステムとする必要はなく、WWW サーバやメールサーバの機能の一部として実現することも可能です。

1.1.7　負荷分散を使ったフォールトトレランス

DNS サーバやロードバランサが、正常に稼働しているサーバだけにリクエストを割り振ることで、フォールトトレランスを実現することができます。その場合には、DNS サーバやロードバランサからサーバの稼働状況を調べる仕組みが必要となります。次のようないろいろな方式が使われています。

- リクエスト処理状況での調査
  実際に割り振ったリクエストの通信開始から終了までの処理が、一定時間内に完了しているかを調べます。
- ICMP レベルでの調査
  ping コマンドと同様の機能（ICMP echo request/ICMP echo replay）を使って、サーバが稼働しているかを調べます。
- TCP ポートレベルでの調査
  TCP のコネクション開設要求を送って、一定時間内にコネクション開設処理が完了するかどうかでサーバの状況を調べます。
- プロトコルレベルでの調査
  特定のプロトコルで通信を行い、サーバの状況を調べます。例えば、WWW サーバであれば HTTP で実際にリクエストを送り正常に応答があること、POP サーバであればユーザ名とパスワードを送り正常に認証処理ができることなど、実際の処理のレベルまで調べます。

Linux サーバでロードバランサを構築すると、自作のプログラムなどでサーバの稼働状態を調査することも可能ですので、より複雑な管理を行うこともできます。

1.2　システムを監視する

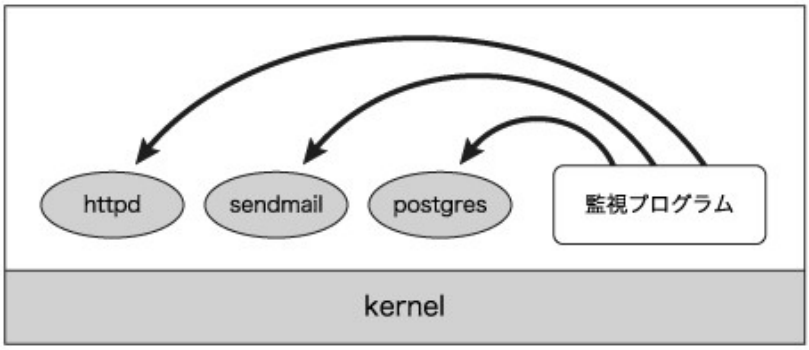
注意深くシステムを冗長化し、フォールトトレランスを保持したシステムを作成したつもりでも、システムが常に正常に稼働し続けるとは限りません。MTTRを短くするためには、できるだけ早くシステムの異常を検知して対策を行う必要があります。そのため、システム管理者は常にシステムの状態を監視する必要があります。

1.2.1　監視の種類

システムの監視には、次のような様々な方法があります。

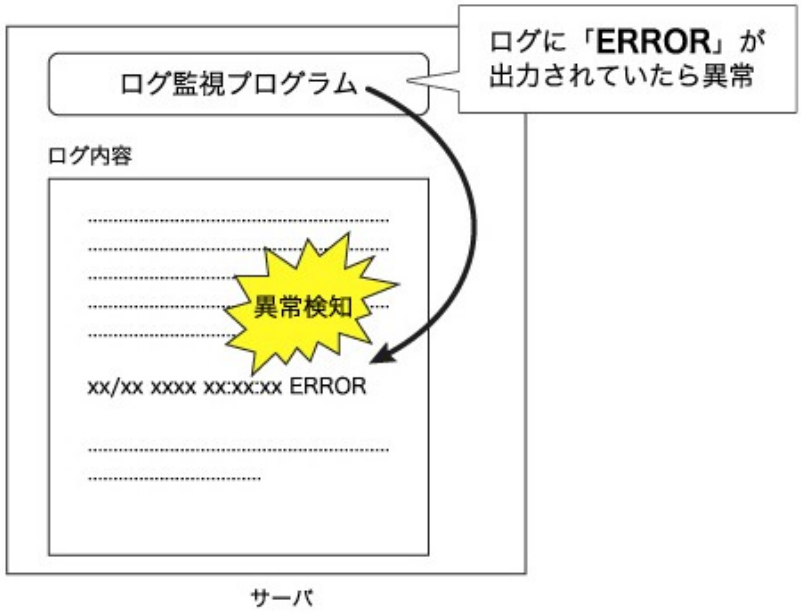
- 自システム内を監視する
  ソフトウェアが正しく動作しているかを、システム内で監視しようとするものです。
  - プロセス監視
    ソフトウェアの動作に必要なすべてのプロセスが正常に動作しているかを検査します(図1-9)。

図 1-9: プロセス監視のイメージ



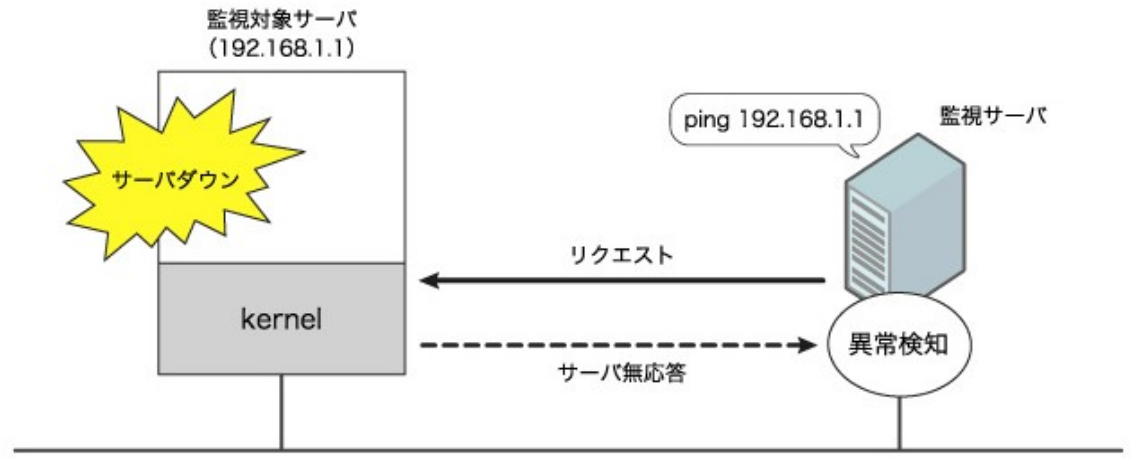
  - ログ監視
    システムやアプリケーションソフトウェアが記録するログを監視し、エラーメッセージ、警告メッセージ、特定のメッセージなどがログに出力されていないかを検査します(図1-10)。

図 1-10:ログ監視のイメージ



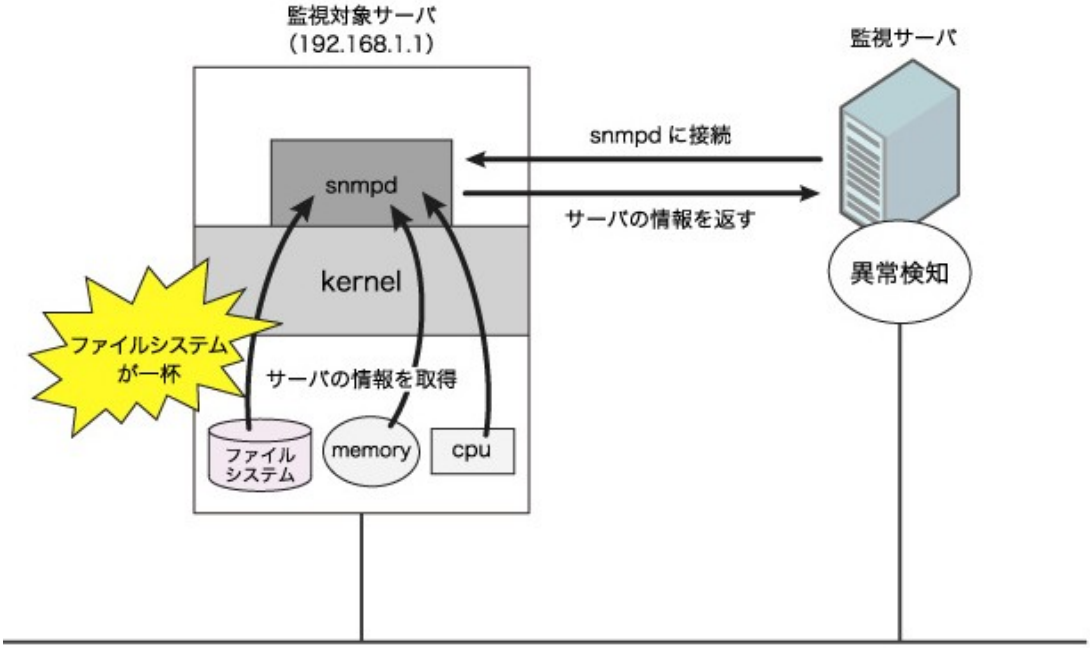
- ネットワーク上の他システムから監視する
  ソフトウェアが正しく動作しているかを、ネットワーク上の他のサーバから監視します。
  - システム稼働監視（PING 監視）
    ping コマンドと同様の機能（ICMP の echo request/echo reply）を利用して、ネットワーク上の他のノードが稼働しているかを検査します。OS のネットワーク機能が正しく動作しているかを検査することができます(図 1-11)。

図 1-11:ping 監視



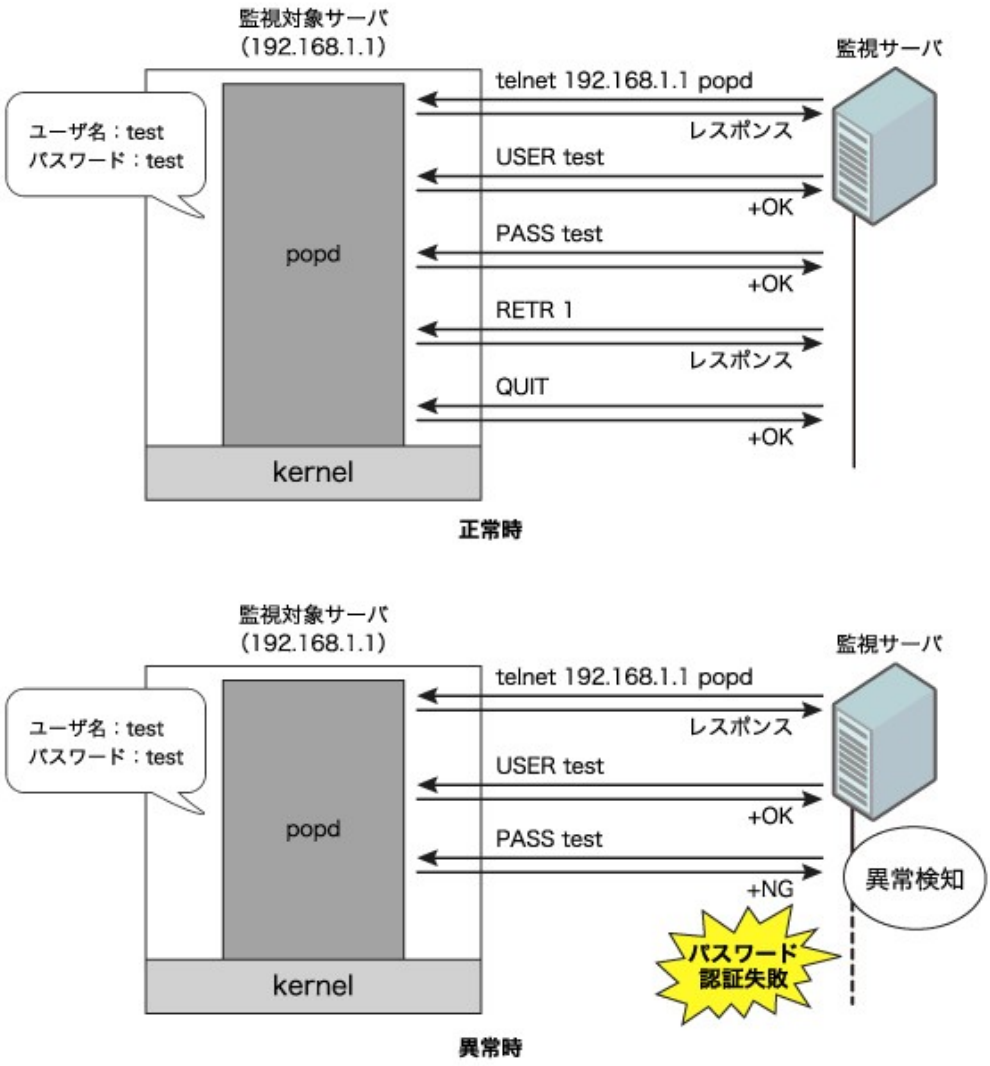
- SNMP監視
  SNMP（Simple Network Management Protocol）を使って、ネットワーク上の他のノードのリソースの状態などを検査します。一般的には、他ノードのプロセスの状況、ファイルシステムの利用率、メモリ利用率、CPU利用率などの基礎的な情報を調べることができます(図 1-12)。

図 1-12:SNMP監視のイメージ



- ポート監視
  TCPポートへの接続要求を行って、システムのWWWやメールのようなネットワークサービスが受け入れ可能な状態にあるかどうかを検査します。
- アプリケーションレベルの監視
  アプリケーションレイヤーのプロトコルや、アプリケーション独特の手順を使って、サービスが正常に稼働しているかどうかを調べます。例えば、次のような手法が使われています(図 1-13)。
  - 実際にWWWサーバのページをダウンロードすることで、WWWサーバが応答を正常に返しているかを検査する。
  - メールサーバにPOPログインしてみて、POPサーバが稼働していて認証ができることを検査する。メールサーバにメールを送ってみて、POPでメールを取ってみることで正常にメールを配信できているかを検査する。
  - データベースに接続して検索を実施してみて、想定された応答があることを調べる。

図 1-13:アプリケーションレベルの監視のイメージ



一般に、システム稼働監視のような低レベルな監視ほど、仕組みが単純で導入が簡単です。それに対して、アプリケーションが正常に動作しているか、データベースの検索が正常に行えているかというような高レベルな監視は、仕組みも複雑で導入も簡単ではありません。

1.2.2　監視と通知

システムの障害にいち早く気がつくためには、システムの障害時にシステム管理者へどのようにして通知するかということが非常に重要です。一般的には、次のような方法が使われています。

1. ランプの点灯

   データセンターなどで大量のコンピュータがある状況では、障害を起こしている装置がどれかを見つけるだけでも大変です。最近の PC サーバには、システムの稼働状態が正常であることを示す LED が付いているものが多くなってきました。こうした LED 機能を利用して外部の人に障害を通知します。

   システム管理者が常にコンピュータを目視で確認していることはできませんので、その他の

方法と組み合わせて利用することが多いようです。

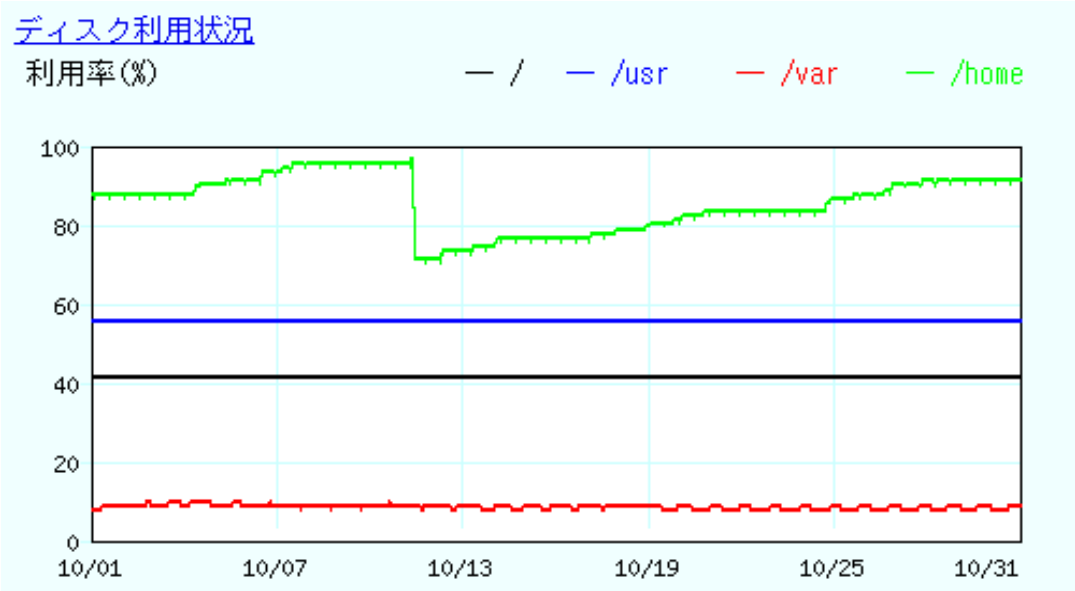
2. 音による通知
   システム管理者が近くにいる場合には、ブザー音、警告音などを鳴らすことで障害を音で知らせるのが有用です。
3. 電話による通知
   システム管理者が遠くにいる場合には、モデムを通じてシステム管理者へ電話によって通知するという方法があります。ただし、障害の内容などの詳細な情報を伝えるためには、特殊な音声発生装置などが必要になります。
4. メールによる通知
   電子メールによって障害を知らせる方法です。最近では、携帯電話のメールサービスのリアルタイム性が向上していますので、携帯電話へメールで通知するという方法がよく使われるようになってきました。メール本文に障害の詳細な内容などを記載して通知することで、システム管理者に多くの情報を送ることができます。

1.2.3　状態管理

システムの監視を行っていると障害の発生をいち早く知ることができます。しかしながら、監視ではシステムの障害そのものを防ぐことはできません。そのため、システムの状態を定期的に調査し、システムが定常状態にあるかどうかを知ることも重要です。

例えば、長い間メールサーバを使っていると、サーバに保管されているメールが徐々に増えてメールを保管するファイルシステムが一杯になり、新しいメールが届かなくなる可能性があります。しかし、ファイルシステムの利用量が少しずつ増えていることに事前に気がつくことができれば、障害が発生する前に対処することができます(図 1-14)。

図 1-14: ディスク利用率のグラフ



特に、Linuxをはじめとするオープンソースソフトウェアを使ってシステムを構築した場合には、状態管理は非常に重要です。

製品のソフトウェアの場合には、厳密に利用条件が決まっていることが多く、また他のソフトウェアと同時に利用することを認めていないことが多いため、ほとんどの場合にシステムの稼働状態はソフトウェアメーカーやソフトウェア開発者の想定の範囲内です。
しかし、オープンソースソフトウェアを使って作成したシステムでは、ハードウェアも導入するソフトウェアもシステム構築者が選択したオリジナルのシステムとなります。そのため、システムの定常状態を知る人は世界に一人もいません。それは、自分で調べるしかないのです。

2章 Linux サーバ1台の稼働率を上げる設計

システム全体の稼働率を向上するためには、まずはサーバ１台１台の稼働率を向上する必要があります。この章では、単独のサーバで行うことのできる対策について学習します。

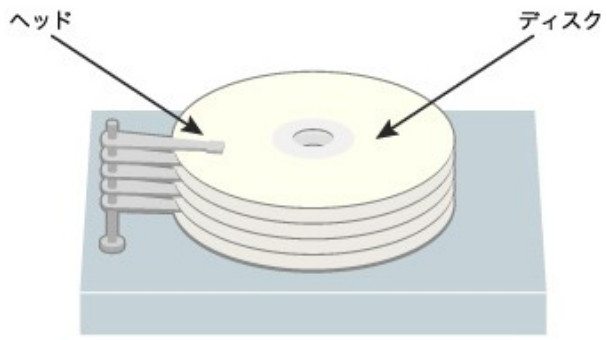
2.1　RAID によるディスクの冗長化

RAID（Redundant Arrays of Inexpensive Disks）は、ディスク障害に対応するために一般的に広く利用されている技術です。本節では、RAID について詳しく学習します。

2.1.1　RAID の概要

ハードディスクは、円盤型のディスクに磁気を使って情報を記録する媒体です。円盤状のディスク上に磁気を使って書かれた情報を、**ヘッド**と呼ばれる読み取り装置を使って読み取ります。ディスクの円盤を回転させることで、ヘッドで読み取る情報の位置を制御します。1 枚のディスクと 1 個のヘッドでは処理できる情報量が少ないため、通常のハードディスクでは何重にもディスクが重ねられた構造になっています(図 2-1)。

図 2-1:ハードディスクの構造



このように、ハードディスクはモーターを使った機械そのもので、小さな空間にディスクとヘッドが収められた精密機器です。経年劣化によりモーターの回転軸の微妙な摩擦の状況が変化したり、外部からの衝撃によりヘッドとディスクの位置関係が変わったり、ディスク上の磁気を記録する物質の密度が変化すると、情報が正常に読み取れなくなってしまうことがあります。

最近では、電源を OFF にすると自動的にヘッドがしまわれたり、媒体上の磁気を記録する物質を均一にする技術が進んだことで、以前に比べると障害は発生しにくくなっています。しかし、依然として、コンピュータの部品の中でもっとも故障率が高いのはハードディスクであると言われています。

一方で、ハードディスクは情報を格納するという重要な機能を持った装置です。万一故障すると、記録した情報がすべて失われてしまう可能性もあり、影響は深刻です。そのため、ハードディスクの故障によりコンピュータが停止したり、情報が失われたりするのを避けるために考案されたのが RAID です。

RAID は、1987 年に米国 University of California, Berkeley の David A Patterson 氏、Garth Gibson 氏、Randy Katz 氏の 3 人が考案したハードディスクの高速化と冗長化のための仕組みです。Linux では、データが失われるのを最小限に抑えるために、この RAID の中のいくつかの方式を使うことができます。RAID には、表 2-1 のような方式があります。このうち、RAID0～5 が当

初から考えられた方式で、それ以降はより安全性を高めるために近年になって考案された方式です。

▼*表 2-1:RAID レベルと動作*

レベル	Linux	動作
RAID0	○	複数のディスクにデータを均等に割り振り、同時に並行して読み書きを行う方式です。ディスク処理を高速化するもので、**ストライピング**とも呼ばれます。最低 2 台のディスクが必要になり、データ容量は全ディスク容量の合計となります。複数のディスクをまとめて 1 つのディスクに見せるため、データの冗長性はありません。
RAID1	○	2 つのディスクに同じデータを書き込む方法です。**ミラーリング**とも呼ばれます。片方のディスクが壊れても、もう片方のディスクで処理を継続できます。データ容量は、全ディスク容量の半分になります。処理を高速化することはできませんが、最低 2 台で信頼性を確保できるため、よく利用される方法です。
RAID2	×	RAID0 を発展させ、ハミングコードと呼ばれる誤り訂正符号を同時に記録するものです。同時にデータの分散も行うことで、冗長性と高速化の両方を狙ったものですが、実用化されていません。
RAID3	×	RAID0 を発展させ、データを分散させた上で、誤り訂正符号のみを記録する特別なディスクを使う方式です。最低 3 台のディスクが必要です。1 つのディスクが故障してもデータを復元することができますが、誤り訂正符号を記録するディスクが分散されていないため、高速化の効果は高くありません。
RAID4	○	データの分散をビット単位ではなくブロック単位で行う以外は、RAID3 とほぼ同様の方式です。
RAID5	○	RAID3 に対して、誤り訂正符号も分散して書き込む方式です。データの冗長性と高速化の両方を実現できることから、よく利用されます。最低 3 台のディスクが必要で、ディスク台数を N として、ディスク容量は 1 台のディスクの容量の(N-1)倍になります。RAID0 と同様によく利用される方式です。
RAID6	○	RAID5 の誤り訂正の情報をさらに二重化する方式です。一度に 2 台のディスクが壊れてもデータを復旧できるという特徴があります。RAID5 よりも 1 台余分にディスクを追加する必要があります。
RAID1+0	○	RAID0 と RAID1 を組み合わせて記録することで、高速化と冗長性の両方を確保しようとする方式です。ミラー化ストライピングや RAID10 とも呼ばれます。

なお、Linux には**リニアモード**という方式がサポートされています。リニアモードは、2 つのディスクを論理的に 1 つに見せることでディスク容量を増やす方式です。ストライピングとは異なり、データをディスク上に分散しませんので、高速化の効果も冗長性もありません。したがって、RAID とは区別して考える必要があります。データ容量がディスクの総ディスク容量になることから、大きなディスク領域を確保するために使われます。

2.1.2　ソフトウェア RAID とハードウェア RAID

RAID には、ハードウェア的に行われる**ハードウェア RAID** と、ソフトウェア的に行われる**ソフトウェア RAID** があります。ハードウェア RAID は、RAID コントローラと呼ばれるハードウェア制御用の特殊な装置を使って実現します。そのため、コンピュータの購入時に RAID コントローラを実装した機器を選ぶ必要があります。最近のサーバコンピュータには、最初から RAID コントローラを実装している機器が多くなりました。そのため、OS のレベルからは単純に 1 つのディスクに見えます。Linux からハードウェア RAID を使うためには、RAID コントローラ用のデバイスドライバが必要となります。サーバの選択時には、Linux のデバイスドライバを入手することができるのかを考慮して機器を選ぶ必要があります。

これに対して、ハードウェアのサポートなしに利用することができるのが、ソフトウェア RAID です。Linux はソフトウェア RAID の機能をサポートしており、表 2-1 のように RAID0, RAID1, RAID5, RAID6 などの方式を利用することができます。

一般的には、ハードウェア RAID はコントローラ上の専用の CPU でデータ処理を行うため、ソフトウェア RAID に比べて高速であると言われています。また、Linux 上の複雑な設定などが必要ないため、手軽に利用することができます。最近では、Linux がサポートする RAID コントローラの種類が充実してきたことから、ハードウェア RAID が使われる機会が多くなりました。

【SATA による RAID サポート】

最近のコンピュータのハードディスクの規格として SATA と SAS がよく使われます。

SATA（Serial Advanced Technology Attachment）は、シリアル ATA とも呼ばれ、主にデスクトップコンピュータをターゲットにしたハードディスクの規格です。これに対して、**SAS**（Serial Attached SCSI）は、**SCSI**（Small Computer System Interface）の伝送方式をシリアル化したものです。SCSI には CPU の負荷軽減や MTTR を短くするための様々な工夫が盛り込まれていることから、サーバコンピュータでは SAS がよく利用されています。しかし、SAS に対応したハードディスクは SATA に比べて複雑なため価格が高く、エントリークラスの低価格サーバでは、SATA を採用した機器も増えています。

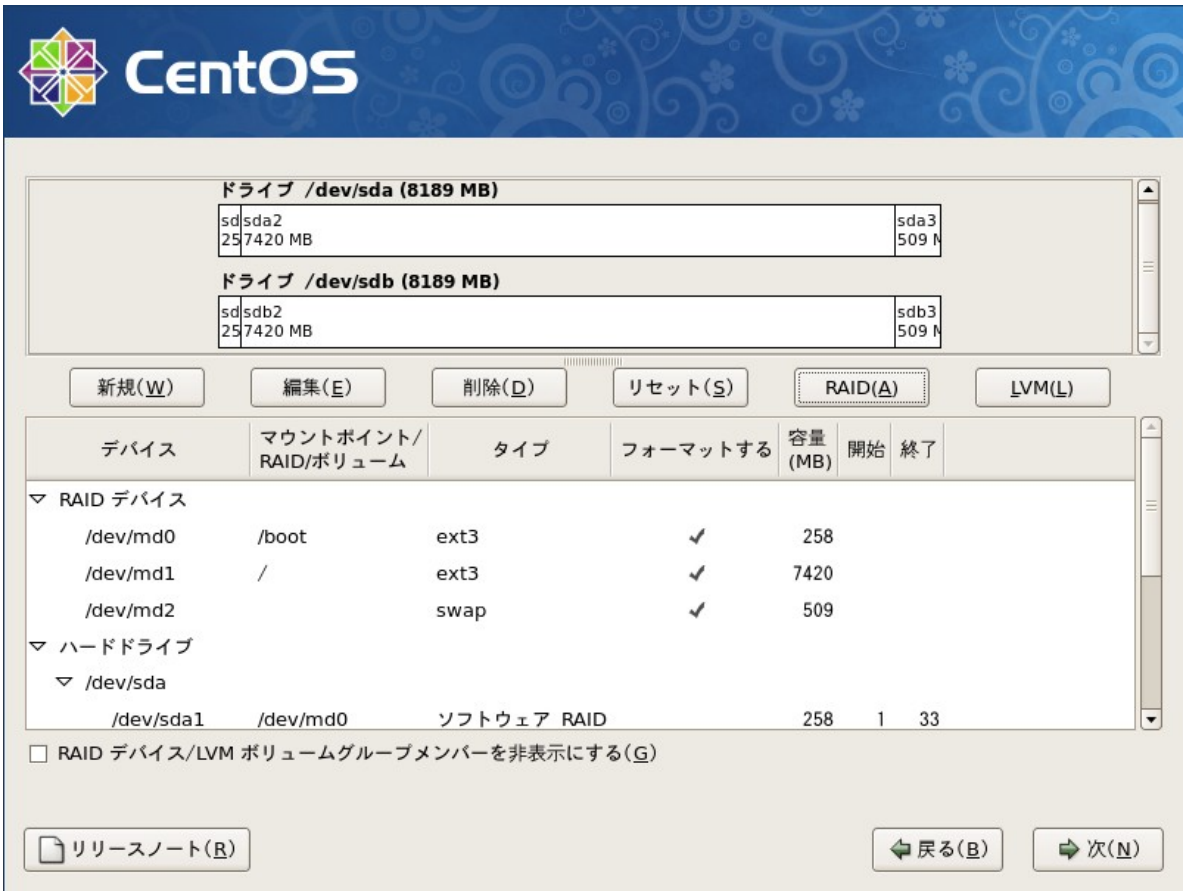
SATA のコントローラの中には標準で RAID1 をサポートする製品があります。ただし、SATA コントローラによる RAID サポートは、実際にはデバイスドライバで行われているソフトウェア RAID の場合が多いようです。Linux でも **Device Mapper** の機能を使って SATA のソフトウェア RAID を利用するモジュール（**dm-raid**）がサポートされています。ただし、この機能はあくまでソフトウェア RAID であることに注意が必要です。SATA は、もともと SAS に比べて CPU の負荷が高い規格です。そのため、SATA とソフトウェア RAID を組み合わせたシステムでは、多くの CPU リソースがディスク処理に占有されてしまい、性能の劣化を招くことが少なくありません。

ソフトウェア RAID を構成する場合には、ディスクの種類にも十分に配慮する必要があるのです。

2.1.3　Linux での実装

Linux では、カーネルモジュールの MD（Multiple Device）がソフトウェア RAID をサポートしています。ほとんどの Linux ディストリビューションでは、MD は標準的に有効になっていて、インストーラで RAID を構成することができます(図 2-2)。

図 2-2:OS インストーラでの RAID 構成イメージ



コマンドラインで RAID を管理する場合には、mdadm コマンドで行います。また、RAID の構成情報は/proc/mdstat を通じて参照することができます。

▼*RAID1 を構成した場合の例*

```
# mdadm -C /dev/md0 -l raid1 -n 2 /dev/sdb1 /dev/sdc1
mdadm: array /dev/md0 started.
# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 sdc1[1] sdb1[0]
      1044096 blocks [2/2] [UU]
            [================>.....]    resync  =  76.5%  (800000/1044096)  finish=0.0min
speed=266666K/sec

unused devices: <none>
```

この例では、RAID のデバイスが正常に構成されディスクの同期処理が行われていて、同期処理の進捗状況が報告されています。

▼*同期処理完了後の状態*

```
# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 sdc1[1] sdb1[0]
      1044096 blocks [2/2] [UU]          ←　ディスクの状態
```

[UU]のように表示されているのは、ディスクの状態を示しています。どちらかのディスクが異常状態になっている場合には、[_U]のように表記されます。

RAID1では、ディスクの異常時に自動的に切り替えを行うための**スペアデバイス**を付けてRAIDを構築することもできます。スペアデバイスを付けてRAID1を構築する場合には次のように”-x 1”というオプションを使ってスペアデバイスを追加します。

▼*スペアデバイスを利用する構成例*

```
# mdadm -C /dev/md0 -l raid1 -n 2 -x 1 /dev/sdb1 /dev/sdc1 /dev/sdd1
mdadm: array /dev/md0 started.
```

スペアデバイスの状態もmdadmコマンドで確認できます。

▼*スペアデバイスの状態表示例*

```
# mdadm --misc -D /dev/md0                          ← アレイの詳細情報表示
/dev/md0:
        Version : 0.90
  Creation Time : Thu Dec  9 16:29:38 2010
     Raid Level : raid1
     Array Size : 1044096 (1019.80 MiB 1069.15 MB)
  Used Dev Size : 1044096 (1019.80 MiB 1069.15 MB)
   Raid Devices : 2
  Total Devices : 3
Preferred Minor : 0
    Persistence : Superblock is persistent

    Update Time : Thu Dec  9 16:29:45 2010
          State : clean
 Active Devices : 2
Working Devices : 3
 Failed Devices : 0
  Spare Devices : 1

           UUID : e4905a18:b2362238:e2c610ba:65cc7e0b
         Events : 0.2

    Number   Major   Minor   RaidDevice State
       0       8       17        0      active sync   /dev/sdb1
       1       8       33        1      active sync   /dev/sdc1
```

```
      2       8       49        -       spare   /dev/sdd1        ←　スペア
```

サーバの起動時に、RAIDが自動的に有効になるようにするためには、/etc/mdadm.confにRAIDの情報を設定しておく必要があります。RAIDの情報は、mdadmコマンドで確認することができます。

▼*mdmonitorサービスの設定例*

```
# mdadm -E -scan -v
ARRAY /dev/md0 level=raid1 num-devices=2 UUID=e4905a18:b2362238:e2c610ba:65cc7e0b
   spares=1   devices=/dev/sdd1,/dev/sdc1,/dev/sdb1
```

この情報を元に、/etc/mdadm.confを作成します。

▼*/etc/mdadm.confの設定例*

```
DEVICE /dev/sd*[0-9]
ARRAY /dev/md0 level=raid1 num-devices=2 spares=1
   UUID=e4905a18:b2362238:e2c610ba:65cc7e0b
   devices=/dev/sdd1,/dev/sdc1,/dev/sdb1
```

また、ディスクの障害などを検知するためは、mdmonitorサービスを利用します。例えば、次のように/etc/mdadm.confに設定を行いmdmonitorサービスを起動しておけば、障害発生時にメールで通知を受けることができます。

▼*mdmonitorサービスの設定例（/etc/mdadm.conf）*

```
MAILADDR admin@designet.jp　←　メール送信先を設定
```

障害が発生すると、次のようなメールが送られてきます。

▼*mdmonitorから届くメールの例*

```
Subject: Fail event on /dev/md0:(ホスト名)


This is an automatically generated mail message from mdadm
running on (ホスト名)

A Fail event had been detected on md device /dev/md0.

It could be related to component device /dev/sdb1.

Faithfully yours, etc.
```

```
P.S. The /proc/mdstat file currently contains the following:

Personalities : [raid1]
md0 : active raid1 sdb1[2](F) sdc1[3] sdd1[1]
      1044096 blocks [2/1] [_U]
           [==>..................]    recovery  =  12.6%  (132608/1044096)  finish=0.1min
speed=132608K/sec

unused devices: <none>
```

2.2　論理ボリューム

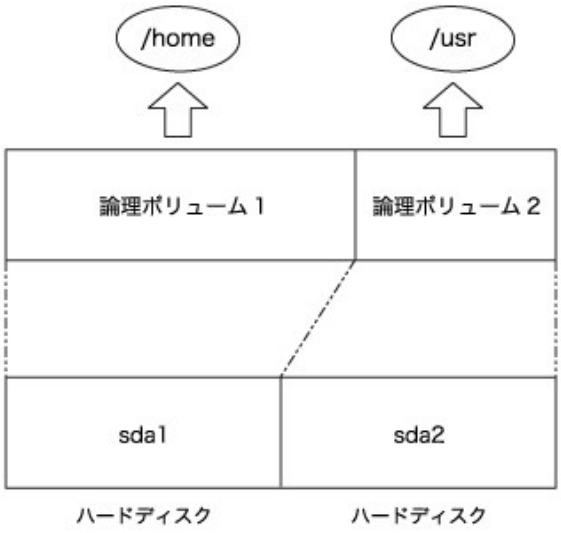
論理ボリューム管理（**LVM**:Logical Volume Manager）は、ハードディスクなどの記憶媒体の物理的な状態を隠蔽し、論理的なイメージで管理するための技術です。本節では、論理ボリュームについて学習します。

2.2.1　論理ボリュームの概要

最近のサーバでは、取り扱うデータの量が著しく増加する傾向があります。そのため、実際にシステムを利用しはじめてから、当初に想定していたよりも多くのディスク容量が必要となる場合も少なくありません。こうした場合には、新しいディスクを接続してディスク容量を増やす必要があります。しかし、ファイルシステムを作り直して従来のデータをすべてコピーするという作業を行うためには、長いシステム停止の時間が必要となります。このようなシステム停止の時間は、システムの稼働率を下げる原因となってしまいます。このような問題を解決するために利用されるのが論理ボリュームです。論理ボリュームを使うと、次のようなメリットがあります。

- ディスクサイズを越える大規模ファイルシステムを作成できる
  論理ボリュームを使うと、複数個のディスクにまたがる大きなファイルシステムを作成することができます。例えば、図 2-3 のように二つのディスクを集めて、大きな論理ボリュームを作ることができます。さらに、その論理ボリュームを分割してファイルシステムを作成することもできます。

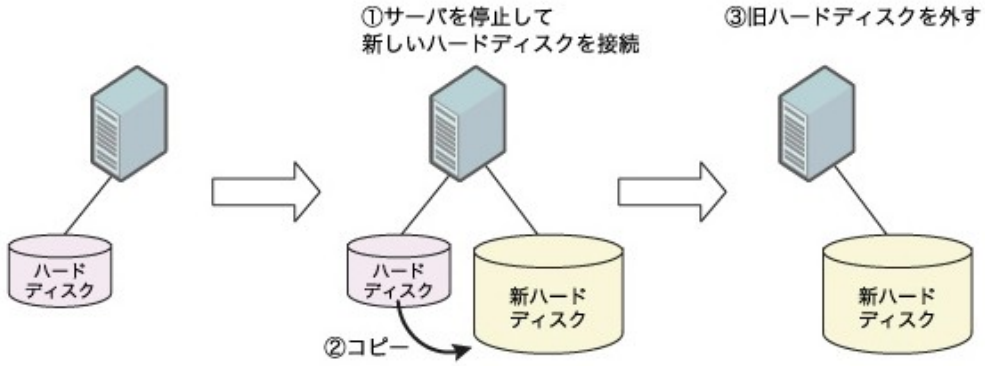
図 2-3:LVM の機能構成



- ファイルシステムが拡張できる
  システムを再起動することなく、論理ボリュームの大きさを変更することができます。図 2-4

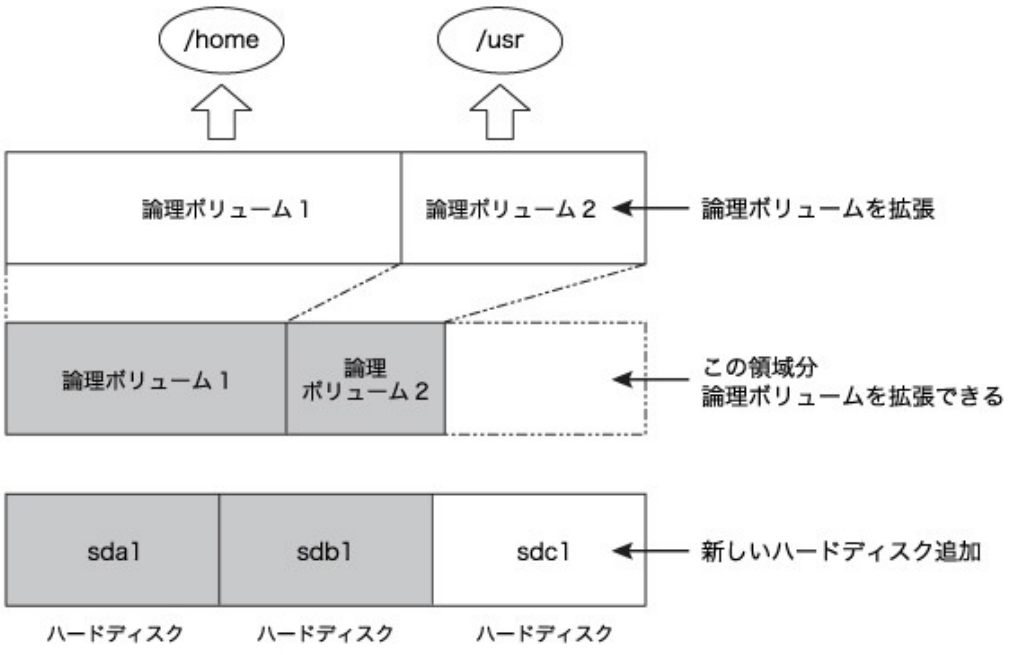
は、従来のディスク増設の方法です。ディスク容量が不足した場合には、このように新しい大きなディスクを接続して新たなパーティションを作成し、古いディスクからデータをコピーする必要がありました。

図 2-4:従来のディスク増設



論理ボリュームを使うと、図 2-5 のように、単純にディスクを増設して論理ボリュームに加えることができ、ファイルシステムを簡単に拡張することができます。

図 2-5:LVM を使ったディスク増設



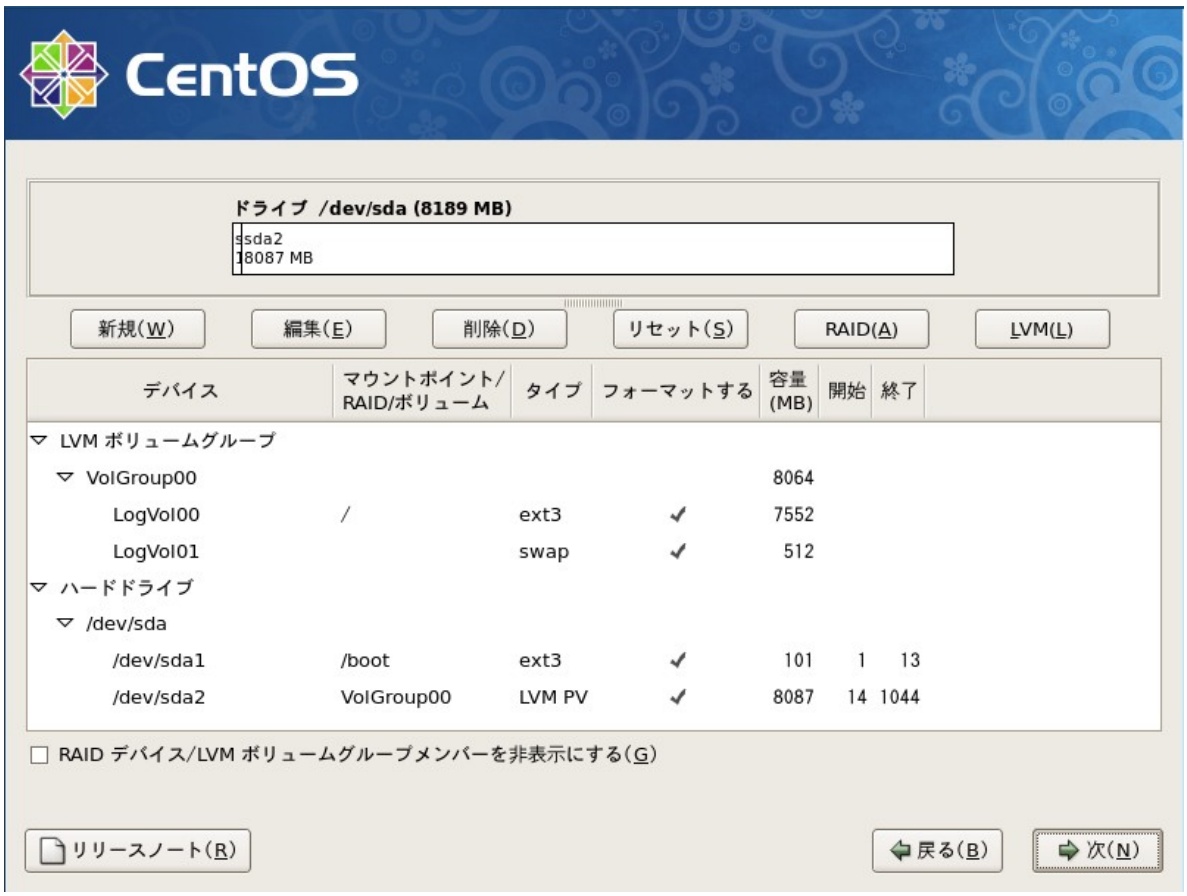
- 障害時の対応が容易である
  論理ボリュームに新たなディスクを追加し、故障したディスクを取り外すことで、調子の悪いハードディスクを簡単に取り外すことができます。

- スナップショット
  論理ボリュームのある一瞬の状態を保管することができます。この機能を使うと、システムを停止することなく、安全にバックアップを取得することができます。

2.2.2　Linuxでの実装

Linuxでは、標準的にLVMをサポートしています。多くのLinuxディストリビューションではLVMの機能が標準的に有効になっていて、インストーラでもLVMを構成することができます(図 2-6)。

図 2-6:OSインストーラでのLVM構成イメージ

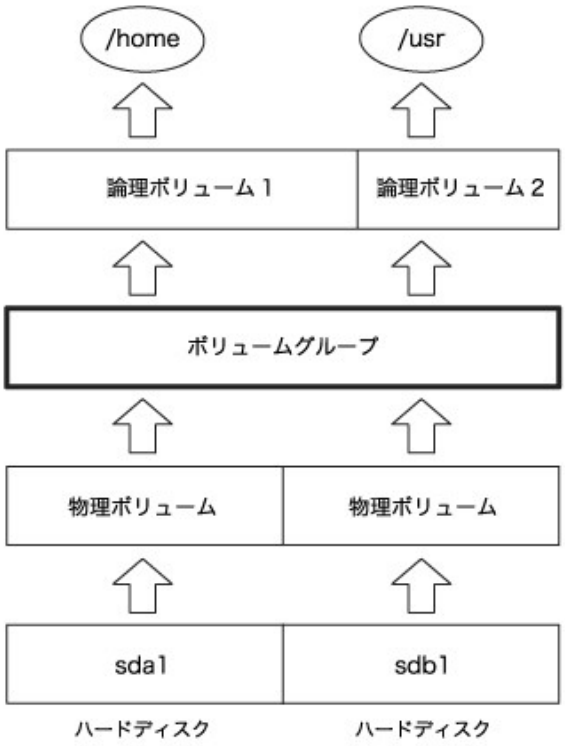


LVMは、図 2-7のように、次のようないくつかの階層からできています。

- 物理ボリューム
  LVMでは物理的な記憶媒体を**物理ボリューム**として管理します。物理ディスクのパーティション単位で物理ボリュームを作成することができます。
- ボリュームグループ
  複数の物理ボリュームを集めて、グループ化したものを**ボリュームグループ**として管理します。これは、論理的な1つのディスクとなります。
- 論理ボリューム

ボリュームグループを複数に分割したものを論理ボリュームと呼びます。物理ディスク上に作成するパーティションと同じ意味を持ちます。

図 2-7:LVM の構造



物理ボリュームの作成

物理ボリュームの作成は、pvcreate コマンドで行います。引数に物理パーティションのデバイス名を指定します。 物理ボリュームの一覧は pvs コマンドで取得することができます。また詳細は、pvdisplay コマンドで確認することができます。

▼*物理ボリュームの作成例*

```
# pvcreate /dev/sdb1                                  ←/dev/sdb1 に物理ボリュームを作成
  Physical volume ”/dev/sdb1” successfully created
# pvcreate /dev/sdc1                                  ←/dev/sdc1 に物理ボリュームを作成
  Physical volume ”/dev/sdc1” successfully created
# pvs                                                   ←物理ボリュームの一覧を確認
  PV         VG   Fmt  Attr PSize   PFree
  /dev/sdb1       lvm2 --   499.98M 499.98M
  /dev/sdc1       lvm2 --   499.98M 499.98M
# pvdisplay                                               ←物理ボリュームの詳細を確認
  ”/dev/sdb1” is a new physical volume of ”499.98 MB”
  --- NEW Physical volume ---
  PV Name               /dev/sdb1
  VG Name
```

```
  PV Size               499.98 MB
  Allocatable           NO
  PE Size (KByte)       0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               1dJCe1-dYfu-Di1H-dN0r-MOY8-UFqb-6OYun7

  "/dev/sdc1" is a new physical volume of "499.98 MB"
  --- NEW Physical volume ---
  PV Name               /dev/sdc1
  VG Name
  PV Size               499.98 MB
  Allocatable           NO
  PE Size (KByte)       0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               rJV7t8-MR2p-MPdd-jQiU-O9T6-C3OH-n3snUP
```

ボリュームグループの作成

ボリュームグループの作成は、vgcreate コマンドで行います。引数に複数の物理ボリュームを指定します。ボリュームグループの一覧は、vgs コマンドで取得することができます。また、pvdisplay コマンドで確認すると、物理ボリュームがどのボリュームグループに所属しているのかを確認することができます。

▼ *2 つの物理ボリュームでボリュームグループを作成した例*

```
# vgcreate vg01 /dev/sdb1 /dev/sdc1              ←vg01 というボリュームグループを作成
  Volume group "vg01" successfully created          /dev/sdb1 と/dev/sdc1 を組み込む
# vgs                                            ←ボリュームグループの一覧を確認
  VG   #PV #LV #SN Attr   VSize   VFree
  vg01   2   0   0 wz--n- 992.00M 992.00M
# pvdisplay                                      ←物理ボリュームの内容を確認
  --- Physical volume ---
  PV Name               /dev/sdb1
  VG Name               vg01                     ←ボリュームグループ名が vg01 になっている
  PV Size               499.98 MB / not usable 3.98 MB
  Allocatable           yes
  PE Size (KByte)       4096
  Total PE              124
  Free PE               124
  Allocated PE          0
  PV UUID               1dJCe1-dYfu-Di1H-dN0r-MOY8-UFqb-6OYun7
```

```
  --- Physical volume ---
  PV Name               /dev/sdc1
  VG Name               vg01        ←ボリュームグループ名がvg01になっている
  PV Size               499.98 MB / not usable 3.98 MB
  Allocatable           yes
  PE Size (KByte)       4096
  Total PE              124
  Free PE               124
  Allocated PE          0
  PV UUID               rJV7t8-MR2p-MPdd-jQiU-O9T6-C3OH-n3snUP
```

論理ボリュームの作成

lvcreateコマンドで、ボリュームグループを分割して、論理ボリュームを作成することができます。引数には、作成する論理ボリュームのサイズ、論理ボリュームの名称、ボリュームグループを指定します。作成した論理ボリュームの一覧は、lvsコマンドで取得できます。作成した論理ボリュームには、“/dev/ボリュームグループ名/論理ボリューム名”という書式のデバイスファイルができます。lvdisplayコマンドに、そのデバイス名を指定することで詳細を確認することができます。

▼*ボリュームグループに論理ボリュームを作成*

```
# lvcreate -L 500M -n lv01 vg01                    ←論理ボリュームを作成
  Logical volume ”lv01” created
# lvcreate -L 268M -n lv02 vg01                    ←論理ボリュームを作成
  Logical volume ”lv02” created
# lvs                                              ←論理ボリュームの一覧を確認
  LV   VG   Attr   LSize   Origin Snap%  Move Log Copy%  Convert
  lv01 vg01 -wi-a- 500.00M
  lv02 vg01 -wi-a- 268.00M
# lvdisplay /dev/vg01/lv01                         ←論理ボリュームの内容を確認
  --- Logical volume ---
  LV Name                /dev/vg01/lv01
  VG Name                vg01
  LV UUID                Z2jB3R-9A7T-iMtg-v9Lo-Iz8K-bhUE-CbueBo
  LV Write Access        read/write
  LV Status              available
  # open                 0
  LV Size                500.00 MB
  Current LE             125
  Segments               2
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
```

ファイルシステムの作成

論理ボリュームを作成した時にできたデバイスファイルを指定してファイルシステムを作成することができます。

▼*論理ボリュームにファイルシステムを作成*

```
# mkfs -t ext3 /dev/vg01/lv01
mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
128016 inodes, 512000 blocks
    :
# mount -t ext3 /dev/vg01/lv01 /data          ←作成したファイルシステムを/data にマウント
```

2.2.3　LVM を使ったディスク管理

LVM の機能を使い、論理ボリュームに対してファイルシステムを作成すると、次のようなことを実現することができます。

- 論理ボリュームを拡張・縮小することができる。
- ディスクの故障時には、物理ボリュームを論理ボリュームから取り除くことができる。
- スナップショットを作ることができる。

物理ボリュームの追加

vgextend コマンドを使えば、システムを停止することなく物理ボリュームをボリュームグループに追加することができます。

▼*ボリュームグループに物理ボリュームを追加*

```
# pvcreate /dev/sdd1                                  ←/dev/sdd1 に物理ボリュームを作成
  Physical volume ”/dev/sdd1” successfully created
# vgextend vg01 /dev/sdd1                           ←/dev/sdd1 をボリュームグループに追加
  Volume group ”vg01” successfully extended
```

さらに、システムを停止することなく論理ボリュームを拡張し、動的にファイルシステムも拡張することも可能です。論理ボリュームの拡張は、lvextend コマンドで行います。ファイルシステムの拡張は、動的リサイズに対応したファイルシステムのみで行うことができます。ext2、ext3、ext4、reizerfs などが動的リサイズに対応しています。ext3 でファイルシステムのリサイズを行うためには、resize2fs コマンドを使います。

▼*論理ボリュームを拡張*

```
# lvextend -L+512M /dev/vg01/lv01          ←論理ボリュームを 512M 拡張
```

```
  Extending logical volume lv01 to 1012.00 MB
  Logical volume lv01 successfully resized
# resize2fs /dev/vg01/lv01                                    ←ファイルシステムを拡張
resize2fs 1.39 (29-May-2006)
Filesystem at /dev/vg01/lv01 is mounted on /data; on-line resizing required
Performing an on-line resize of /dev/vg01/lv01 to 1036288 (1k) blocks.
The filesystem on /dev/vg01/lv01 is now 1036288 blocks long.
```

論理ボリュームの縮小

他の論理ボリュームのサイズが不足した場合には、余裕のある論理ボリュームを縮小して、その分を容量の追加にまわしたい場合があります。LVMでは、こうした用途を想定して、論理ボリュームを縮小することができます。論理ボリュームの縮小はlvreduceコマンドで行います。ただし、論理ボリュームを縮小する場合には、あらかじめ論理ボリューム上に作成されているファイルシステムのサイズを縮小しておく必要があります。ファイルシステムの縮小の方法はファイルシステムの種類によってことなりますが、ext2、ext3などでは一度マウントを解除してfsckを実行し、ディスク上の問題を取り除いた状態で実施する必要があります。

▼*論理ボリュームを縮小*

```
# umount /dev/vg01/lv01                                       ←　マウント解除
# fsck -f /dev/vg01/lv01                                      ←　強制的にfsckを実行
fsck 1.39 (29-May-2006)
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/vg01/lv01: 11/258064 files (9.1% non-contiguous), 43602/1036288 blocks
# resize2fs /dev/vg01/lv01 256M                               ←　サイズ変更
resize2fs 1.39 (29-May-2006)
Resizing the filesystem on /dev/vg01/lv01 to 262144 (1k) blocks.
The filesystem on /dev/vg01/lv01 is now 262144 blocks long.
# mount -t ext3 /dev/vg01/lv01 /data                          ←　再マウント
# lvreduce -L256M /dev/vg01/lv01                              ←　論理ボリュームの縮小
  WARNING: Reducing active and open logical volume to 256.00 MB
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce lv01? [y/n]: y                   ←　yを入力
  Reducing logical volume lv01 to 256.00 MB
  Logical volume lv01 successfully resized
```

エラーディスクの交換

LVMをうまく使うと、ディスク上のデータを退避し、ディスクを安全に交換することができます。ハー

ドディスクの交換は、次のような手順で行います。

- システムの停止し、ハードディスクを追加（ホットスワップディスクの場合は不要）
- 新しいハードディスクを必要なサイズのパーティションに分割
- 物理ボリュームを作成
- 新しく作った物理ボリュームを、交換するディスクと同じボリュームグループに追加
- 交換するディスクの物理ボリューム上のデータを新しい物理ボリュームに移行
- 交換するディスクの物理ボリュームを、ボリュームグループから削除します

▼*物理ボリューム上のデータを移行*

```
# modprobe dm-mirror                                    ←　モジュールの読み込み
# pvmove  /dev/sdb1
  /dev/sdb1: Moved: 100.0%
# vgreduce vg01 /dev/sdb1                               ←　物理ボリュームの取り外し
  Removed ”/dev/sdb1” from volume group ”vg01”
```

この手順は、論理ボリューム上のファイルシステムをマウントしたまま行うことができます。ホットプラグ対応のディスクを使っていれば、システムを停止することなく安全にディスクを交換することが可能です。

スナップショット機能

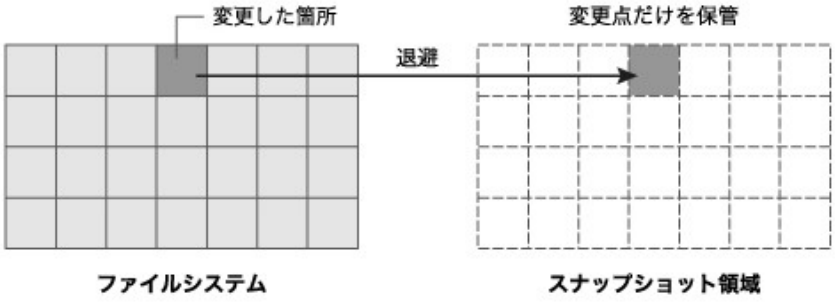
Linux の LVM は、スナップショット機能を持っています。これは様々な用途でデータのバックアップを取得する時に有用で、システムの稼働率を向上することができます。

例えば、ファイルシステムやデータベースなどは、ディスク上の複数の情報を一貫して書き換える必要があるため、システムの稼働中には完全なバックアップを取得することができません。そのため、完全なバックアップを取得するにはシステムを完全に停止する必要があります。これは、稼働率を低下させる要因となります。

しかし、スナップショット機能では、システムの運用中のある一瞬のファイルシステムの状態を保管することができます。この機能を利用することで、システムを停止することなく安全にバックアップを取得することができます。

LVM のスナップショット機能は、全ディスクを退避するのではなく、ファイルシステムへのデータ更新が行われたときに、古いデータをスナップショット領域に退避していくという仕組みで動作します。そのため、スナップショットを取得するために利用するディスク容量も節約することができます(図 2-8)。

図 2-8:スナップショットの変更管理



▼*スナップショットの利用例*

```
# vgs                                              ← ボリュームグループの空きを確認
  VG   #PV #LV #SN Attr   VSize   VFree
  vg01   2   2   0 wz--n- 992.00M 480.00M          ←Vfree のサイズを確認
# lvcreate -s -L 256M -n snap01 /dev/vg01/lv01     ←スナップショットを snap01 という名前
  Logical volume "snap01" created                    で作成
# lvs
  LV     VG   Attr   LSize   Origin Snap%  Move Log Copy%  Convert
  lv01   vg01 owi-ao 256.00M
  lv02   vg01 -wi-a- 256.00M
  snap01 vg01 swi-a- 256.00M lv01     0.0          ← snap01 が作成されている
# mount -r /dev/vg01/snap01 /snapshot              ← /snapshot に snap01 をマウント
# tar cvf /dev/rmt/0 /snapshot                     ← tar コマンドでバックアップ
# umount /dev/vg01/snap01                          ← マウント解除
# lvremove /dev/vg01/snap01                        ← スナップショットを削除
Do you really want to remove active logical volume snap01? [y/n]: y      ← y を入力
  Logical volume "snap01" successfully removed
```

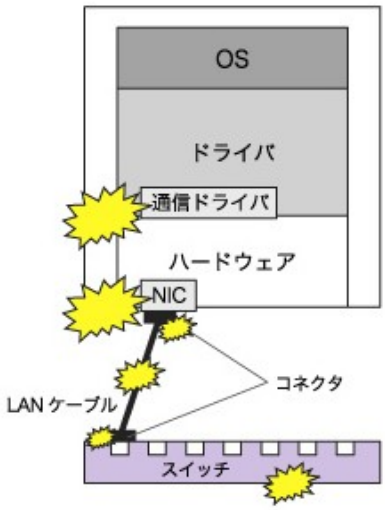
2.3　ネットワークインタフェースの冗長化

ハードディスクに次いで故障のしやすい箇所は、ネットワークインタフェースです。特にネットワーク上でサービスを提供しているサーバでは、故障の影響は深刻です。たとえ CPU、メモリ、ハードディスクなどのメインの機能が正常に動作していても、ネットワークインタフェースが故障してしまえば何のサービスも提供できなくなってしまいます。

ネットワークインタフェースの故障には様々な要因があります(図 2-9)。

- 通信ドライバの不良
- ネットワークカードの故障
- ネットワークインタフェースのコネクタ部の故障
- LAN ケーブルの不良
- 接続先のスイッチの故障
- スイッチとの通信方式のネゴシエーションの失敗

図 2-9:ネットワークの故障要因

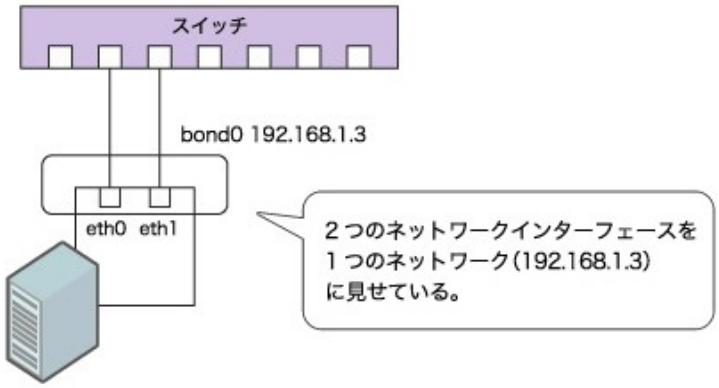


こうしたケースに対応するために、Linux は**ボンディング**（bonding）と呼ばれるネットワークインタフェースを冗長化する機能をサポートしています。

2.3.1　ボンディングの概要

ボンディングは結合インタフェースとも呼ばれ、複数のネットワークインタフェースを仮想的な 1 つのデバイスとして利用できるようにする技術です。障害時の切り替えだけでなく、複数のケーブルを使って、大容量通信を実現する用途でも利用することができます(図 2-10)。

図 2-10:ボンディング



ボンディングでは、複数のインタフェースを束ねた仮想的なインタフェースを作ることができます。このようなインタフェースは、**ボンディングインタフェース**と呼ばれ、bond0, bond1 のように名称が付けられます。1 つのボンディングインタフェースには、**スレーブインタフェース**と呼ばれる複数の物理インタフェースを参加させることができます。

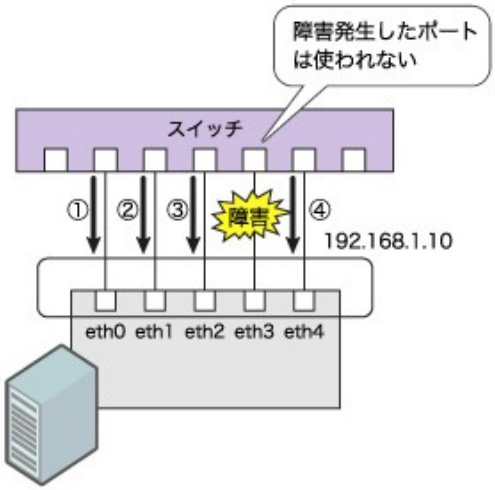
2.3.2　ボンディングの種類

Linux は、ボンディングに対応していて、次のような種類の結合をサポートしています。

- **アクティブ・バックアップ型**（active-backup）
  複数のネットワークインタフェースのうちの 1 つだけをアクティブインタフェースとして利用します。その他のインタフェースは、バックアップ用となり、通常の状態では利用されません。もし、アクティブインタフェースに何らかの障害が発生すると、バックアップインタフェースの中の 1 つが替わりに使われます。したがって、すべてのネットワークインタフェースが故障しない限り、通信を継続することができます。

- **ラウンドロビン型**　（balance-rr）
  複数のネットワークインタフェースすべてを順番に利用します。どれか 1 つのインタフェースに障害が発生した場合には、そのインタフェースを使わなくなります。すべてのネットワークインタフェースが故障しない限り、通信を継続することができます。また、物理的に複数のネットワークインタフェースを同時に利用しますので、通信帯域もインタフェース数に応じて増加します。
  また、ラウンドロビン型を発展させて、宛先によって利用するインタフェースを固定する XOR 分散利用(balance-xor)型、すべてのインタフェースに同時にパケットを送るブロードキャスト(broadcast)型などもあります。
  これらのラウンドロビン型を利用するためには、接続するスイッチが**トランキング(Trunking)**と呼ばれる機能をサポートしていなければなりません(図 2-11)。

図 2-11:トランキングの場合のシステム構成例



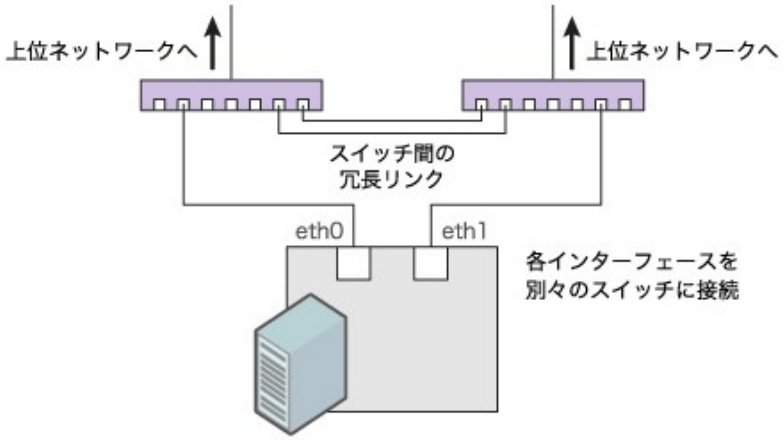
- **ロードバランス型**
  ラウンドロビン型と同様に、すべてのインタフェースを利用して通信を行います。各インタフェースの利用状況に応じて、どのインタフェースにパケットを送信するかを決定します。受信を単一のインタフェースで行う送信ロードバランス型 (balance-tlb)、受信側の負荷分散も実現する**適応分散型** (balance-alb)などがあります。ラウンドロビンでは、すべてのインタフェースが同一条件で利用されるため、各インタフェースの通信速度が同じでないとうまく動作しませんが、このモードではインタフェースの利用状況に応じて通信が行われるため、異なる速度のインタフェースを混在させることができます。また、送信ロードバランス型はスイッチが特別な機能をサポートしていなくても利用することができます。

- **IEEE 802.3ad 動的リンク型** (802.3ad)
  IEEE 802.3ad は、**リンクアグリゲーション**(Link Aggregation)と呼ばれ、多くのスイッチで実装されています。このモードでは、全インタフェースを使って通信します。接続しているスイッチが 802.3ad を実装していなければなりません。すべてのネットワークインタフェースが故障しないかぎり、通信を継続することができます。

ボンディングとネットワークの冗長構成

一見すると、アクティブ・バックアップ型は普段利用しないインタフェースがあるため、非効率にも見えます。ただ、図 2-12 のように各インタフェースを別々のスイッチに接続することで、ネットワーク全体の冗長構成を行うことができます。また、どのようなスイッチに対しても有効に利用することができます。

図 2-12:複数のスイッチを使ってボンディング



2.3.3　Linux での実装

最近では多くの Linux ディストリビューションで、ボンディングの機能を利用できるようになっています。Linux でボンディングを使うためには、次のような手順が必要です。

- ボンディングドライバ(bonding)を有効にする。
- ボンディングインタフェースを定義する。
- スレーブインタフェースを定義する。

ボンディングドライバの有効化

ボンディングドライバの有効化は、/etc/modprobe.conf で行います。次の例のように、ボンディングインタフェースを bonding モジュールの別名として登録します。

▼ */etc/modprobe.conf*

```
alias bond0 bonding
```

ボンディングインタフェースの定義

bond0 というインタフェースの設定ファイルを/etc/sysconig/network-scripts/に作成し、通常のネットワークインタフェースと同様の IP アドレス設定を行います。BONDING_OPTS に、追加でボンディングのオプションを設定することができます。次は、bond0 のインタフェース設定ファイルの例です。

▼ */etc/sysconfig/network-scripts/ifcfg-bond0*

```
DEVICE=bond0
BOOTPROTO=static
ONBOOT=yes
```

```
TYPE=Ethernet
IPADDR=192.168.1.161
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
BONDING_OPTS="miimon=100 mode=active-backup"　　←オプション設定
```

この例では、active-backup 型を設定しています。また、miimon というオプションを設定しています。MII は、最近ではほとんどのネットワークカードのドライバがサポートしている機能で、インタフェースのリンク状態などを管理することができます。この例では、100ms 毎にインタフェースのリンク状態を確認するようにしています。

スレーブインタフェースの定義

通常の物理インタフェースの設定ファイルでは、そのインタフェースがどのボンディングインタフェースに参加するかを指定します。次は、その設定例です。

▼ */etc/sysconfig/network-scripts/ifcfg-eth0*

```
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
TYPE=Ethernet
MASTER=bond0
SLAVE=yes
```

▼ */etc/sysconfig/network-scripts/ifcfg-eth1*

```
DEVICE=eth1
BOOTPROTO=none
ONBOOT=yes
TYPE=Ethernet
MASTER=bond0
SLAVE=yes
```

ボンディング状態の確認

ボンディングの状態は、/proc/net/bonding/bond0 を経由して確認することができます。

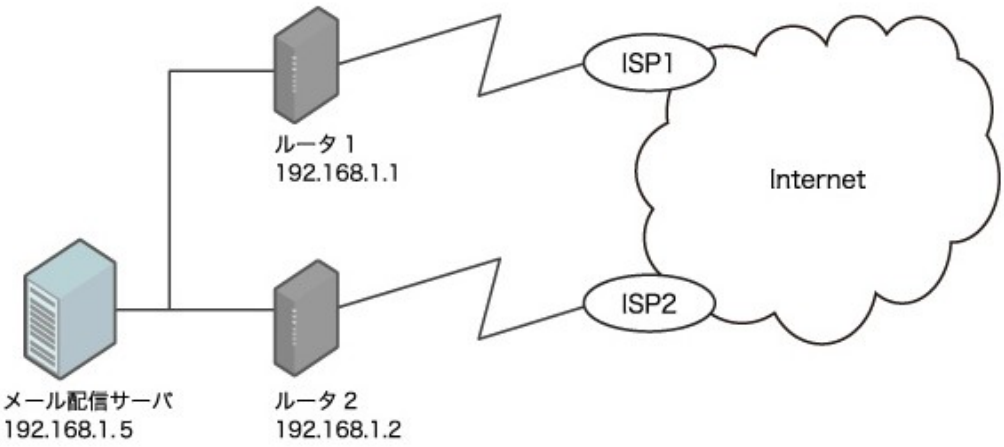
```
$ cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.2.4 (January 28, 2008)

Bonding Mode: load balancing (active-backup)
MII Status: up
MII Polling Interval (ms): 0
```

```
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: eth0                          ← スレーブインタフェースの名前を確認
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:50:56:00:00:02

Slave Interface: eth1                          ← スレーブインタフェースの名前を確認
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:0c:29:55:ee:e0
```

2.4 通信経路の冗長化

サーバのネットワークインタフェースが正常に働いていても、ネットワークの出口にあたるルータが故障すると、サーバの通信ができなくなってしまう場合があります。こうした故障は、サーバの稼働率とは関係なく発生する可能性があります。図 2-13 は、こうした場合を想定したシステムの構成例です。

図 2-13:デフォルトゲートウェイの冗長化構成例



この例では、メール配信サーバはルータ 1、ルータ 2 を介してインターネットと接続されています。通常の Linux のネットワーク設定では、ルータ 1 かルータ 2 のどちらかしかゲートウェイとして設定することができません。そのため、デフォルトゲートウェイに指定してあるルータが故障すると、メールの配信が行えなくなります。

2.4.1 アドバンスト IP ルーティング

Linux には、**アドバンスト IP ルーティング(Linux Advanced IP Routing)**と呼ばれる機能がサポートされていて、複雑なルーティング設定が行えるようになっています。次のような機能がサポートされています。

- ポリシールーティング
  通信プロトコルや宛先などのパケットの内容によって、あらかじめ決めたポリシーにしたがった経路制御を行う機能を**ポリシールーティング**と呼びます。パケットを選別するためのルール(**セレクタ**)と対応する動作(**アクション**)を使って経路制御のポリシーを定義することができます。また、経路制御のルールを、**ルーティングポリシーデータベース**(Routing Policy DataBase: RPDB)として管理します。
- マルチパスルーティング
  一つの宛先に対して複数の経路を持つことを**マルチパスルーティング**と呼びます。複数の経路を同時に使って負荷分散したり、経路を冗長化することができます。

- マルチルーティングテーブル
  複数の経路制御テーブルを管理し、ポリシールティングのアクションによって使い分けることができます。
- クラスベースキュー
  パケットをいくつかのクラスに分けて管理する機能です。クラスごとに、優先制御や帯域管理を行うことができます。

2.4.2　デフォルトゲートウェイの冗長化

アドバンスト IP ルーティングのマルチパスルーティングの機能を利用することで、デフォルトゲートウェイを冗長化することができます。

次の例は、デフォルトゲートウェイを 2 つ設定する場合のコマンド例です。

▼*2 つのゲートウェイを指定（標準的なデフォルトゲートウェイの設定を行っていない場合）*

```
# ip route add default nexthop via 192.168.1.1 nexthop via 192.168.1.2      ←設定
# ip route show                                                             ←確認
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.5
169.254.0.0/16 dev eth0  scope link
default
        nexthop via 192.168.1.1  dev eth0 weight 1
        nexthop via 192.168.1.2  dev eth0 weight 1
```

このように設定しておくと、通常は最初に指定したルータ 192.168.1.1 がデフォルトゲートウェイとして使われます。そして、192.168.1.1 のルータが ARP に応答しなくなると、192.168.1.2 が替わりに使われるようになります。そのため、実際の切り替わりは ARP テーブルがクリアされてからとなります（通常 15 分程度）。

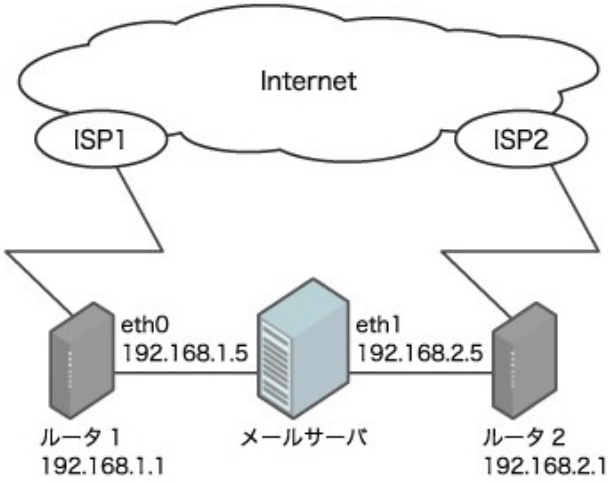
なお、このようなデフォルトゲートウェイの切り替えで二重化できるのは、サーバから外部への通信のみです。例えば、メール配信専用のサーバや、SNMP を使った監視サーバなど、自分から外部への通信を行うサーバでは、このような仕組みを有効に利用することができます。

2.4.3　外部からサーバへの通信の二重化

外部からサーバへのアクセスを二重化するための設定はより複雑です。例えば、図 2-14 のようにサーバの 2 つのインタフェースに別々にインターネットへの接続ができるルータが接続されている構成を考えてみます。

- 1 台のメールサーバが ISP1、ISP2 に接続されている
- ISP1 には mail1.example.com、ISP2 には mail2.example.com という名称で公開されている
- DNS の MX レコードの設定で、両方のサーバが指定されている

図 2-14:外部サーバへの通信の二重化



こうした構成のサーバでは、ISP1 を経由して、eth0 のインタフェース(192.168.1.5)に着信したメールのセッションについては、ルータ 1 を経由して応答を返す必要があります。同様に、eth1 のインタフェース(192.168.2.5)に着信したメールのセッションについては、ルータ 2 を経由して応答を返す必要があります。

このような場合のルーティングの設定も、アドバンスト IP ルーティングのポリシールーティングの機能を利用すれば実現することができます。ポリシールーティングの設定は次のような順で行います。

- 経路テーブルの登録
- 経路テーブルの設定
- セレクタの設定

経路テーブルの登録

経路テーブルの登録は、/etc/iproute2/rt_tables で行います。次は、ISP1、ISP2 のそれぞれの経路を管理するための経路テーブルを作成する例です。

▼*経路テーブルの設定：経路番号 100、101 を isp1、isp2 として登録する*

```
#
# reserved values
#
#255    local
#254    main
#253    default
#0      unspec
#
# local
#
#1      inr.ruhep
```

```
100     isp1                                                    ←  追加
101     isp2                                                    ←  追加
```

経路テーブルの設定

各経路テーブルへのルーティング経路の設定は、ip routeコマンドで行います。次は、isp1、isp2のそれぞれの経路テーブルにデフォルトゲートウェイの設定を行う例です。

▼*デフォルトゲートウェイの設定*

```
# ip route add default via 192.168.1.1 table isp1
# ip route add default via 192.168.2.1 table isp2
```

一般的には、ローカルリンク上のネットワークも経路テーブルに設定しておく必要があります。次は、その設定例です。同一ネットワーク上の通信は、ゲートウェイを使わないように設定しています。

▼*同一ネットワークへのルーティング設定*

```
# ip route add 192.168.1.0/24 dev eth0 table isp1
# ip route add 192.168.2.0/24 dev eth1 table isp2
```

セレクタの設定

それぞれの経路テーブルをどのようなときに使うかというルール（セレクタ）を設定します。次は、192.168.1.5を送信元とする場合にはisp1を、192.168.2.5を送信元とする場合にはisp2を使うように設定する場合の例です。

▼*セレクタの設定*

```
# ip rule add from 192.168.1.5 table isp1 priority 100
# ip rule add from 192.168.2.5 table isp2 priority 101
```

設定の確認

次の例のようにip route show, ip rule showコマンドで設定の確認を行うことができます。

▼*経路設定の確認*

```
# ip route show table isp1                                  ←経路テーブル isp1 の確認
192.168.1.0/24 dev eth0  scope link
default via 192.168.1.1 dev eth0
# ip route show table isp2                                  ←経路テーブル isp2 の確認
192.168.2.0/24 dev eth1  scope link
default via 192.168.2.1 dev eth1
# ip rule show                                              ←セレクタの確認
0:      from all lookup 255
100:    from 192.168.1.5 lookup isp1
101:    from 192.168.2.5 lookup isp2
32766:  from all lookup main
32767:  from all lookup default
```

3章 複数台のサーバによる高信頼性システムの設計例

システムの稼働率を向上させるには、サーバ単体の対策では限界があります。しかし、稼働率の比較的高いサーバを複数台組み合わせることで、より高い稼働率を実現することができます。この 章では、複数のサーバを用意して、必要に応じてリクエストを受け取るサーバを切り替えることで稼働率を向上するための仕組みの実例を学習します。

3.1　DNS による負荷分散

DNS サーバには、アドレスの問い合わせに対して、複数のサーバ IP アドレスを返却する機能があります。この機能を利用して、サーバのリクエストを分散させることができます。

3.1.1　DNS ラウンドロビン

▼*DNS ラウンドロビンの設定例（ゾーンファイル）*

```
www                     IN      A     192.168.1.5
                        IN      A     192.168.1.6
                        IN      A     192.168.1.7
```

これは、DNS マスタサーバのゾーンファイルで、複数のサーバアドレスを返却する設定の例です。www というリソースレコードに対して、192.168.1.5, 192.168.1.6, 192.168.1.7 という 3 つの IP アドレスが返却されます。

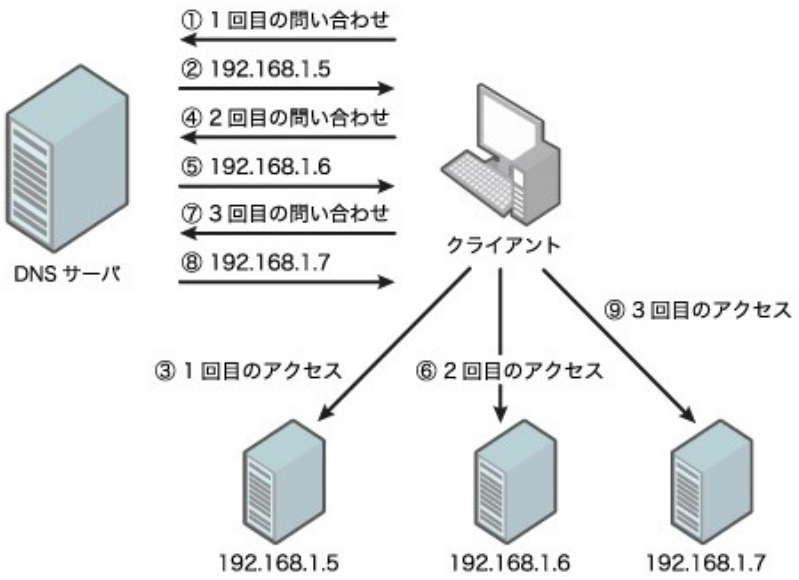
実際に DNS サーバへの問い合わせを行うと、DNS サーバは 3 つのサーバのすべての IP アドレスを返却します。この時に返却する IP アドレスは、表 3-1 のように問い合わせのたびに順序が入れ替わります。多くのアプリケーションでは、複数の IP アドレスが返却されても先頭の 1 つ目の IP アドレスしか利用しないため、DNS サーバの応答に合わせて順次利用するサーバが変わります。

▼*表 3-1:IP アドレスの返却*

1 回目	2 回目	3 回目
192.168.1.5	192.168.1.6	192.168.1.7
192.168.1.6	192.168.1.7	192.168.1.5
192.168.1.7	192.168.1.5	192.168.1.6

DNS サーバの標準的な設定では、IP アドレスを図 3-1 のように順番に変更していきます。そのため、クライアントから見ると、毎回違うサーバを使うことになります。このような方法を **DNS ラウンドロビン**と呼びます。

図 3-1:DNS ラウンドロビンのイメージ



3.1.2　DNS ラウンドロビンの問題点

DNS ラウンドロビンは、サーバの負荷を分散するためには非常に有用です。障害によってサーバが停止した場合には、該当サーバをリソースレコードから取り除くだけで、そのサーバへのアクセスを抑制することができます。DNS の設定だけで、障害が発生したサーバを取り除けるため、MTTR を短縮することができます。しかも、簡単に導入できるというメリットがあります。

しかし、次のような問題があります。

- DNS キャッシュの問題
  DNS の応答は、クライアントや各組織の DNS キャッシュサーバによって一定時間キャッシュされます。そのため、毎回必ず違うサーバにアクセスする訳ではありません。また、DNS サーバ側で設定を変更しても、一定期間は情報が更新されない可能性があります。
- サーバ稼働状態の問題
  何らかの障害でサーバが停止しても、DNS サーバはそれを感知しません。そのため、障害があるサーバのアドレスを返却し続けます。

つまり、サーバの障害が発生した場合の MTTR は、次のようになります。

MTTR ＝ 障害検知時間＋DNS レコード変更時間＋DNS キャッシュ時間

DNS キャッシュが存在するため、その情報が行き渡るまでにはしばらく時間がかかります。そのため、システム全体の MTTR は、DNS キャッシュの時間よりも必ず長くなってしまいます。DNS のキャッシュ時間は、DNS サーバ側のリソースレコード毎に設定ができますので、短めの時間を設定す

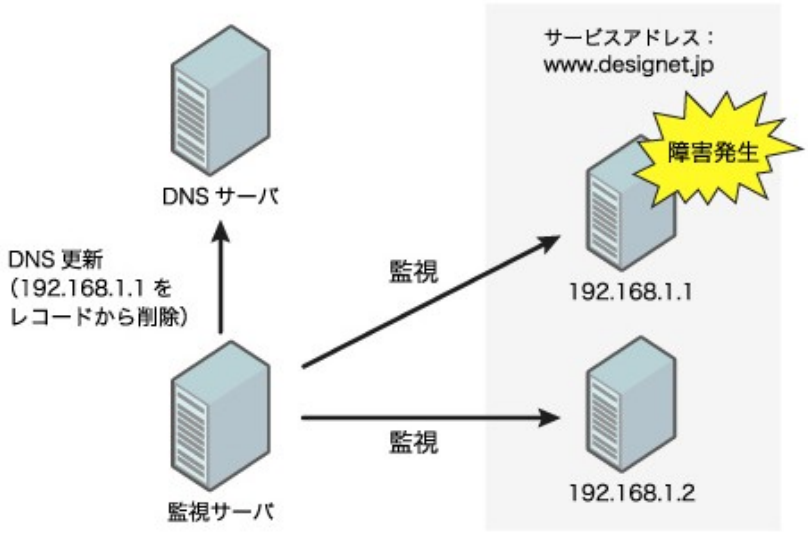
る必要があります。

3.1.3　DNS バランス

DNS ラウンドロビンでサーバを切り替えるシステムで稼働率を高くするためには、DNS キャッシュ時間だけでなく、障害検知時間と DNS レコード変更時間をできるだけ短くする必要があります。そのためには、サーバの稼働状況を時々確認し、問題がある場合にはそれを取り除く処理を少しでも早く行わなければなりません。

DNS サーバとしてもっともよく使われている BIND では、ダイナミック DNS をサポートしています。この仕組みを使えば、必要に応じて DNS レコードを動的に変更することができます。図 3-2 は、こうした仕組みを使ったシステム構成の例です。

図 3-2:DNS バランスのイメージ



この例では、監視サーバからサービスの稼働状況を定期的に調査し、障害を少しでも早く検知できるようにします。また、障害を検知したら、ダイナミック DNS を使って該当サーバを自動的に DNS レコードから削除します。このようなダイナミック DNS を使った切り替えの仕組みを **DNS バランス**と呼びます。

BIND でダイナミック DNS を有効にするには、マスタサーバのゾーンの設定に allow-update ステートメントを追加し、DNS レコードの更新を実施することができるサーバを登録します。なお、ダイナミック DNS を有効にしたドメインのリソースレコードの変更は、必ず nsupdate コマンドなどを使ってダイナミック DNS の手順で実施する必要があります。リソースデータベースファイルを直接変更しないように注意してください。

▼*ゾーンのダイナミック DNS を有効にする（/etc/named.conf）*

```
        :
zone "designet.jp" IN {}
        type master;

        file "designet.jp.db";
        allow-update { 192.168.1.5; };                    ← （追加）
        allow-transfer { 192.168.10.155; };
        notify yes;
};
        :
```

▼*nsupdate によるリソースレコードの更新*

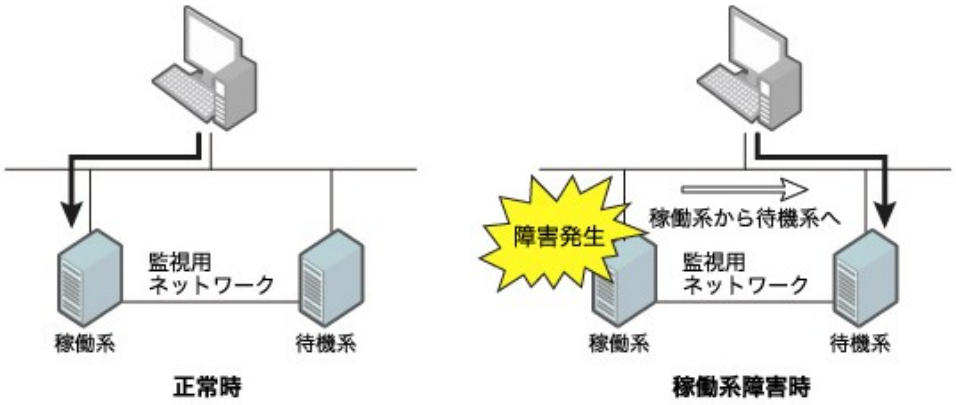
```
$ nsupdate
> server 192.168.1.5
> update delete www.designet.jp
> update add www.designet.jp 10 IN A 192.168.1.2
> update add www.designet.jp 10 IN A 192.168.1.3
> update add www.designet.jp 10 IN A 192.168.1.4
> send
> quit
```

BIND では、残念ながら DNS バランスを実現するための障害検知の仕組みやリソースレコードを自動的に更新するプログラムは用意されていません。これらのソフトウェアは、自分で用意する必要があります。

3.2 アクティブ・スタンバイクラスタリング

アクティブ・スタンバイクラスタリングは、2 台のサーバを利用したデュプレックスシステムの一種です。通常の状態では、2 台のうちの 1 台だけがサービスを提供し、もう 1 台はバックアップサーバとして待機しています。障害が発生した場合には、バックアップサーバに切り替えてサービスを提供します(図 3-3)。

図 3-3:アクティブ・スタンバイクラスタリングのイメージ

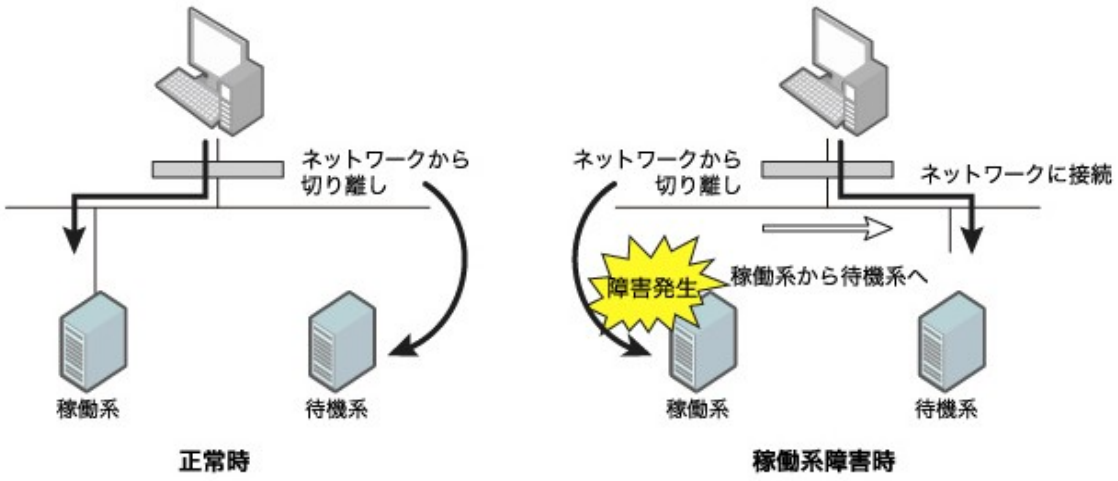


3.2.1 コールドスタンバイ

アクティブ・スタンバイのシステムの中で、もっとも単純な方法は、まったく同じサーバを 2 台用意しておく方法です。この場合には、1 台でサービスを提供し、もう 1 台は電源を OFF にするか、ネットワークから切り離しておきます。障害が発生した場合には、もう 1 台のサーバの電源を ON にしたり、ネットワークを継ぎ替えるたりすることで、物理的にサーバを入れ替えます(図 3-4)。

非常に単純ですが、物理的にハードウェアを入れ替えることで障害からの復旧を行うことができますので、MTTR を短縮することができます。

図 3-4:コールドスタンバイのイメージ



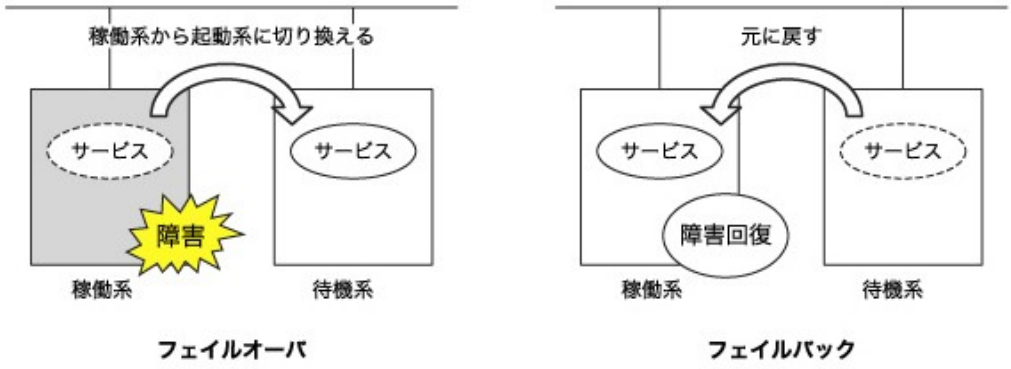
しかし、次のような問題もあります。

- スタンバイ機は通常はまったく動作していませんので、障害が発生したときになって電源を入れてみると、正常に動作しない場合があります。
- サーバに対する変更がある場合には、スタンバイ機も同様に設定を変更して、確実に動作するように確認を行っておく必要があります。システムの入れ替えには、物理的な配線の変更などの作業が伴います。そのため、システム障害を検知してから復旧までには、作業時間が必要です。

3.2.2　アクティブスタンバイ

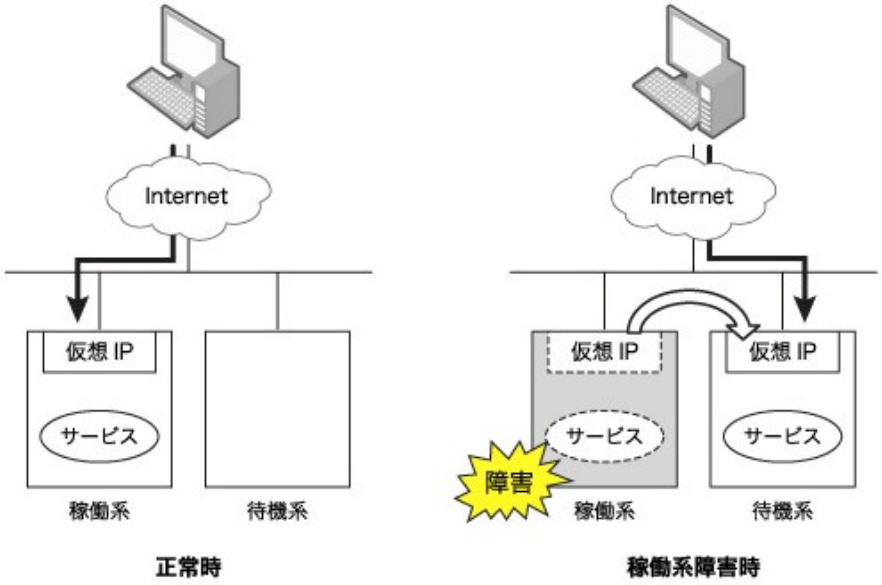
こうした問題に対応するために、最近ではシステムの障害を自動的に検知してサーバを切り替える仕組みが利用されるようになっています。待機しているサーバから稼働しているサーバのサービスの実施状況を監視し、障害時には自動的にシステムを切り替えます。障害時に、稼働系サーバから待機系サーバに切り替える動作を**フェイルオーバ**と呼びます。また、故障した機器の修理などを行い、元の状態に戻す動作を**フェイルバック**と呼びます(図 3-5)。

図 3-5: フェイルオーバとフェイルバック



実際に稼働しているサーバのフェイルオーバやフェイルバックを行ったときに、サービスを行うサーバへクライアントからのリクエストが届くようにするため、一般的にはサービス用の IP アドレスをサーバ間で付け替える処理を行います(図 3-6)。

図 3-6:IP アドレス移動のイメージ



この IP アドレスの切り替え処理には、IP アドレスと MAC アドレスを引き継ぐ方法と、IP アドレスだけを引き継ぐ方法の 2 つの方法があります。

3.2.3　VRRP

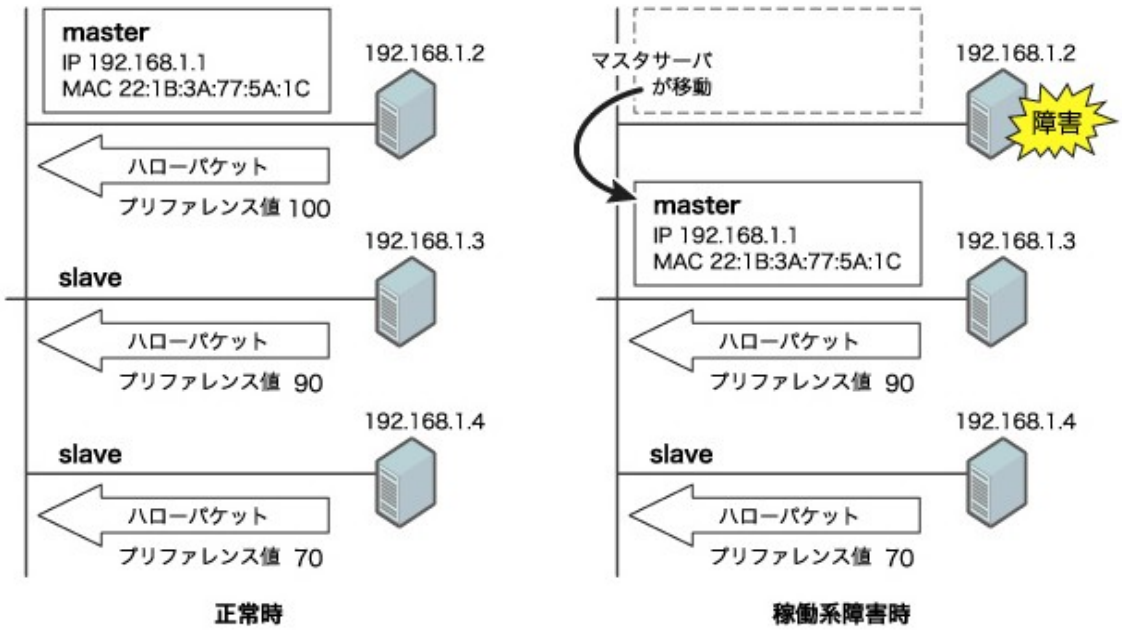
VRRP（Virtual Router Redundancy Protocol）は、RFC2338 で定義されたプロトコルで、ルータやファイアウォールなどのネットワーク機器で冗長性を確保するためによく使われています。Linux でも、VRRP を実現することができます。

VRRP では、いくつかの装置の間で複数個の仮想 IP アドレスを共有することができます。各装置が定期的にハローパケットと呼ばれる確認用パケットを送信することで、稼働状態にあることを他の機器に通知します。サービス中の装置のハローパケットが送信されなくなった場合には、もっとも優先順位の高い（プリファレンス値の大きい）バックアップ装置がサービス用に切り替わります(図 3-7)。

このように VRRP は、監視と IP アドレスの切り替えの両方をサポートしていますが、この監視はアプリケーションレベルのものではなく、装置そのものの稼働を監視することしかできません。これは、VRRP がもともとルータ間で冗長構成を採るために考えられたプロトコルであるためで、ルータやファイアウォールなど比較的単純なサービスで利用されることが多いようです。

VRRP では、各装置のインタフェース固有の MAC アドレスとは別に、サービス用の MAC アドレスを使います。そのため、IP アドレスと MAC アドレスは常に対になっています。サービスを稼働しているサーバでその MAC アドレスのパケットを受け取ることで、サービスを切り替えます。他のネットワーク装置が ARP テーブルに記録した IP アドレスと MAC アドレスの情報は、そのまま継続して利用することができるため、シームレスに装置間で処理を切り替えることができます。

図 3-7: VRRP のイメージ

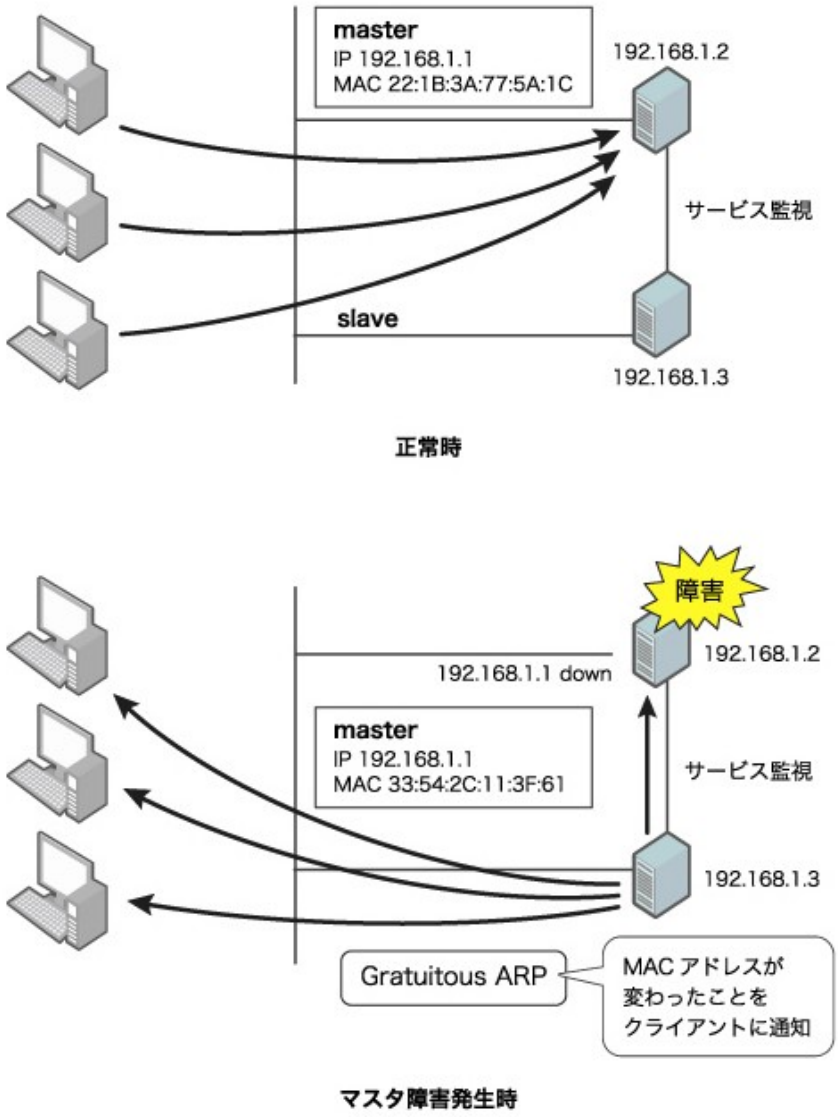


3.2.4　Gratuitous ARP

VRRP とは別に、フェイルオーバやフェイルバックが行われたときに、各サーバに強制的に IP アドレスを付与してしまう方法も使われます。この場合には、各サーバのネットワークインタフェースに付属している MAC アドレスがそのまま使われるため、IP アドレスと MAC アドレスの組み合わせは変更になります。そのため、ネットワーク上の様々な機器が ARP テーブルに記憶している IP アドレスと MAC アドレスの情報を更新しないと、正常に通信を行うことができなくなります。

それを解決するために使われるのが、Gratuitous ARP という特殊なアドレス通知です。Gratuitous ARP を受け取った装置は、自身の ARP テーブルを破棄するか書き換えなければなりません。これによって、IP アドレスを管理するサーバが切り替わったことを、周辺の装置に強制的に学習させます(図 3-8)。

図 3-8:Gratuitous ARP のイメージ



3.2.5　Linux での実装

Linux では VRRP を利用した IP アドレスの切り替えも、Gratuitous ARP を使った IP アドレスの切り替えも利用することができます。

keepalived

VRRP を実現するための代表的なソフトウェアは、keepalived です。

keepalived （ http://www.keepalived.org/ ）は 、Linux Virtual Server Project (http://www.linux-vs.org/) が提供しているソフトウェアです。サーバの稼働監視と VRRP によるフェイルオーバ、フェイルバックをサポートします。

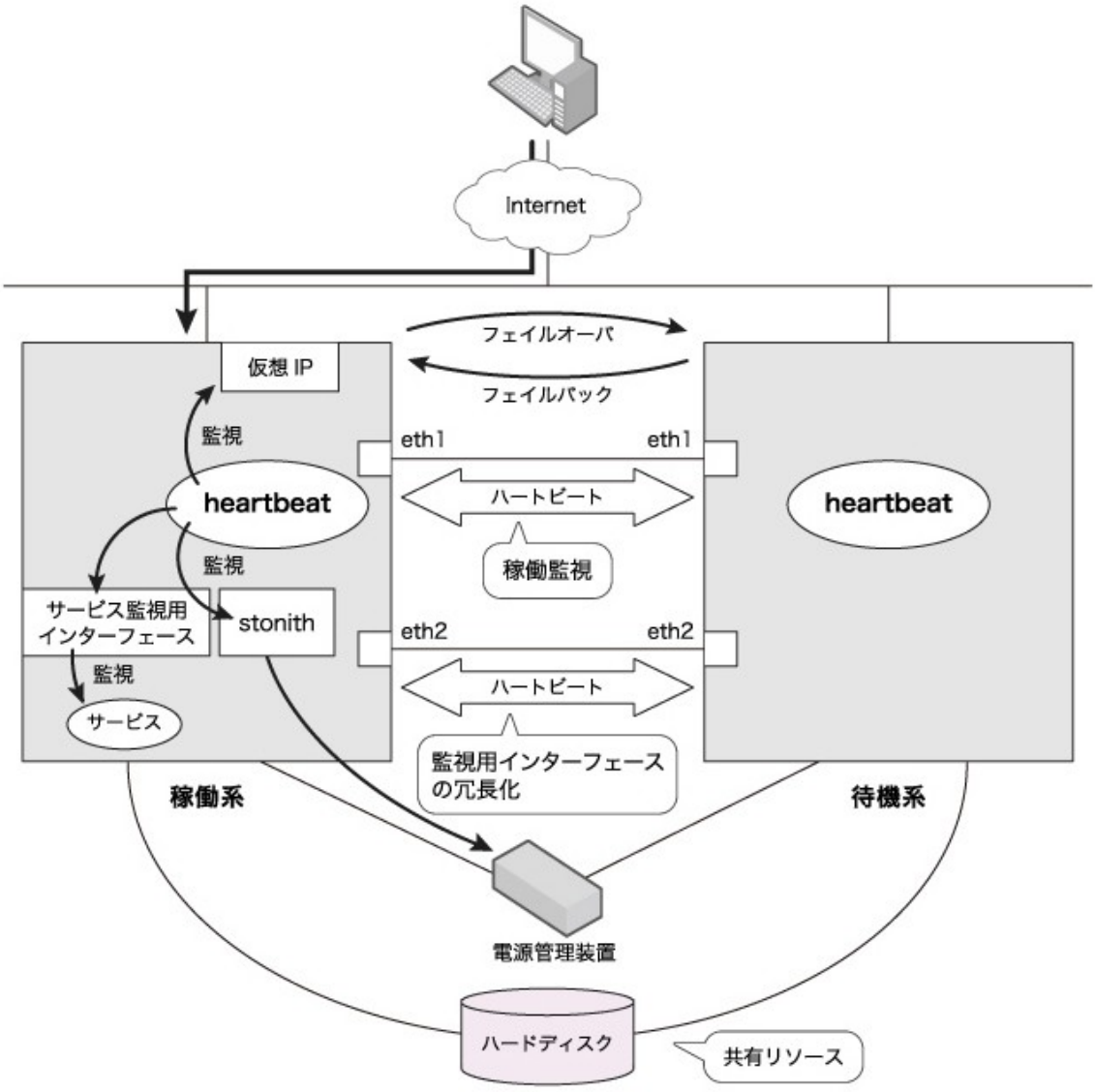
heartbeat

Gratuitous ARP を使ったシステムの切り替えをサポートする仕組みを提供する代表的なソフトウェアは、heartbeat（http://www.linux-ha.org/wiki/Heartbeat）です。heartbeat は The Linux-HA project（http://www.linux-ha.org/）が提供しているクラスタソフトウェアで、IP アドレスの切り替えだけでなく、サーバの稼働監視や様々なサービスの稼働状態を監視することができるインタフェースを提供しています。

3.2.6　heartbeat

heartbeat は、IP アドレスの切り替えをはじめとする、アクティブ・スタンバイクラスタに必要な様々な機能を提供するソフトウェア群です。図 3-9 は、heartbeat の機能のイメージです。

図 3-9:heartbeat の機能



heartbeatでは、次のような機能が提供されています。

- 稼働監視
  稼働系サーバと待機系サーバとの間で、定期的にハートビートと呼ばれるパケットを交換し、お互いの稼働状況をモニタします。それぞれの監視のタイムアウト時間などは必要に応じて設定できます。
- 監視用インタフェースの冗長化
  相手サーバの状況をより詳細に把握するため、複数のネットワークインタフェースを通じてハートビートを交換することができます。ネットワークインタフェースのうちの何個かが故障しても、他のインタフェースを使って正常に切り替えができます。監視用インタフェースとして、シリアルインタフェースも利用できます。
- サービス監視インタフェース
  稼働系サーバで、自サーバ内部のサービスが正しく動作していることを管理するためのインタフェースが用意されています。
- 共有リソース管理
  IPアドレス、ネットワークサービス、ファイルシステムなどを、フェイルオーバ・フェイルバック時に引き継ぐ必要のある**共有リソース**として管理できます。
- 自発的フェイルオーバ
  稼働系サーバが自サーバ内の問題を検知したときには、自らサービスの継続を放棄し、待機系サーバに切り替えることができます。
- 強制的フェイルオーバ
  待機系サーバが稼働系サーバの異常を検知した時には、強制的にサービスを稼働系サーバから待機系サーバに切り替えることができます。
- フェイルバック
  稼働系サーバが復旧し再起動してきたときに、手動または自動でフェイルバックすることができます。
- STONITH
  サービスの切り替えが正常に行えない場合には、相手サーバの電源を切断するなどの方法で、強制的に相手サーバを停止できます。

次は、heartbeatの設定ファイル(/etc/ha.d/ha.cf)ファイルの例です。

▼*/etc/ha.d/ha.cfの設定例*

```
#
# /etc/ha.d/ha.cf
#

# ログ
logfile   /var/log/ha-log

# 監視時間設定
keepalive 2                          ← ハートビートを交換する時間間隔（秒）
deadtime  30                         ← 相手が停止しているとみなす時間（秒）
warntime  10                         ← ログに警告を記録するまでの時間（秒）
```

```
initdead  120                              ← 起動後、監視開始するまでの時間（秒）

# Ethernet デバイス経由のハートビートのパラメータ
udpport 694                                ← ハートビートを交換する UDP のポート番号
ucast   eth1 10.0.0.128                    ← ハートビート用インタフェースと相手の IP アドレス

# シリアルインタフェース経由のハートビートのパラメータ
serial  /dev/ttyS0                         ← ハートビートを交換するシリアルデバイス
baud    19200                              ← シリアルデバイスに設定するボーレート


# 自動フェイルバック
auto_failback off                          ← 自動フェイルバック

# ウォッチドック
watchdog /dev/watchdog                     ← ウォッチドックデバイス

# クラスタノード
node    sv01 sv02                          ← クラスタを構成するノード

# 外部プログラム
respawn root /usr/local/bin/check_active   ← 監視用外部プログラムの起動設定
```

クラスタで管理するリソースは、次の例のように/etc/ha.d/haresources ファイルで設定します。

▼*クラスタのリソース設定（/etc/ha.d/haresources）*

```
sv01 httpd 192.168.1.129/24
```

先頭の sv01 は、稼働系となるサーバ名です。共有リソースとして httpd サービスと代表 IP アドレス（192.168.1.129/24）を共有リソースとして定義しています。heartbeat を構成する 2 つのサーバの設定は、/etc/ha.d/ha.cf ファイルの相手の IP アドレス（ucast）の部分以外は、同じ設定となります。

このように、heartbeat では、比較的簡単な設定を行うだけで共有リソースや代表 IP アドレスの管理ができます。

クラスタの共有リソース

heartbeat には標準で、IP アドレスやファイルシステムなどを共有するためのクラスタ共有リソースの設定が同梱されています。さらに、条件を満たせば/etc/init.d/配下に用意されているシステムのサービス制御スクリプトも、そのままクラスタ共有リソースとして利用することができます。さらに、同様の機能を有するスクリプトを用意すれば、オリジナルの共有リソースを定義することも可能です。

クラスタ共有リソースとして利用することのできるスクリプトは、次のような条件を満たす必要があります。

- 引数に start を指定することで、サービスが開始される。
- 引数に stop を指定することで、サービスを停止できる。
- 引数に status を指定することで、サービスの状態を確認できる。サービス稼働時には「running」という文字列を含んだメッセージを出力して正常終了し、サービス停止時には「stopped」という文字列を含んだメッセージを出力して異常終了する。

OpenSUSE などの Linux ディストリビューションでは、標準的なサービス制御スクリプトがこの仕様を満たしています。そのため、そのままクラスタ共有リソースとして利用することができます。しかしながら、RedHat Enterprise Linux や Fedora の標準的なサービス制御スクリプトは、引数に status を指定して実行しても、「running」や「stopped」のような文字列を出力しません。こうした Linux ディストリビューションでは、標準的なサービス制御スクリプトを利用するのではなく、次の例のように仕様を満たすラッパープログラムを自分で作成する必要があります。

▼ *リソースラッパープログラムの例(/etc/ha.d/resource.d/mysqld)*

```
#! /bin/sh
ORIGINAL=/etc/init.d/mysqld

if [ "$1" == "status" ]
then
    ${ORIGINAL} status
    RET=$?
    if [ $RET -eq 0 ]
    then
        echo "running"
    else
        echo "stopped"
    fi
else
    ${ORIGINAL} $*
    RET=$?
fi
exit $RET
```

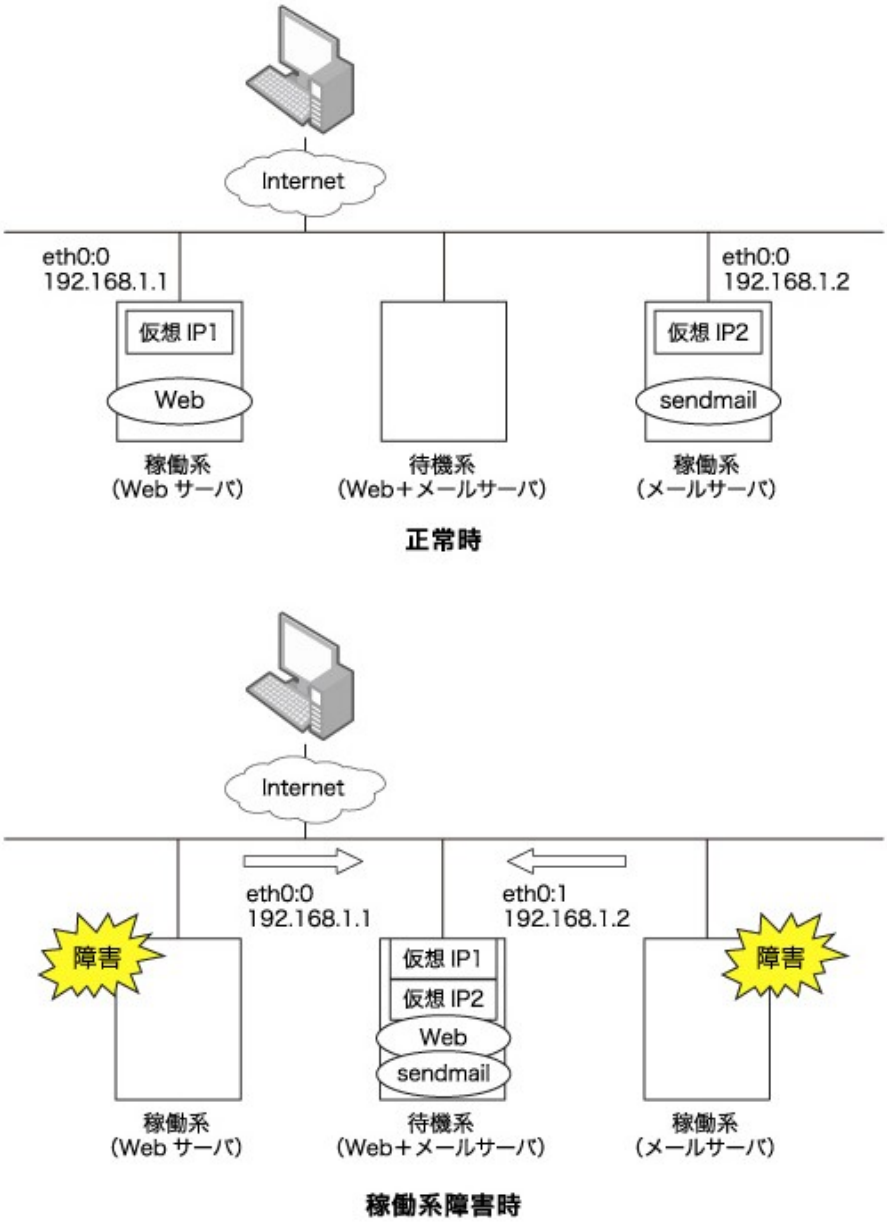
スクリプトを作成したら、/etc/ha.d/resource.d/へ配置することで、共有リソースの制御スクリプトとして利用できるようになります。

3.2.7　マルチサーバ構成のクラスタと Pacemaker

heartbeat では、単純な 2 台のサーバでのアクティブ・スタンバイたけではなく、より多くのサーバが参加するマルチサーバ構成（n-node 構成）のクラスタシステムをサポートしています。例えば、WWW サーバとメールサーバの 2 つのサーバ機能を冗長化しようとすると、アクティブ・スタンバイクラスタでは 4 台のサーバが必要になります。このうちの 2 台は、通常は待機系として監視だけを

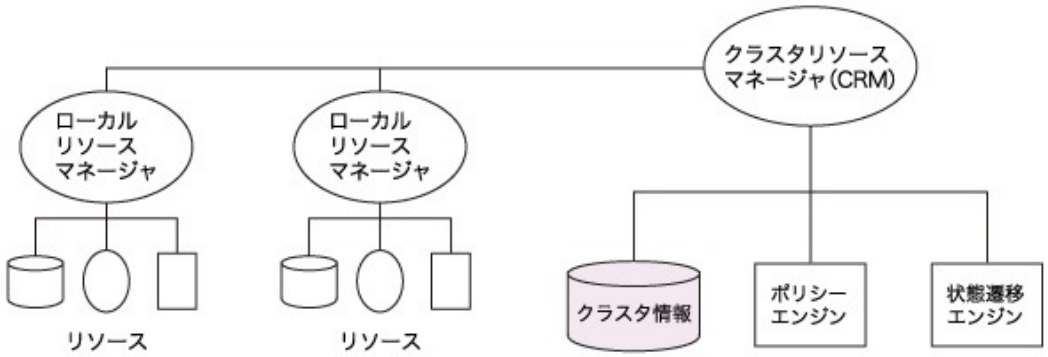
行っています。マルチサーバ構成のクラスタシステムでは、WWW サーバ、メールサーバの稼働系のサーバに加えて、待機系サーバを 1 台だけ作れば冗長構成を実現することができます(図 3-10)。

図 3-10: マルチサーバ構成のクラスタシステム



こうしたマルチサーバ構成のクラスタシステムでは、必要となるリソースと、その依存関係が大変複雑になります。例えば、メールサーバを稼働する場合には、そのデータを保管したディスクのリソースを利用する権利や、メールサービス用の IP アドレスも利用できなければなりません。こうした複雑な構成をサポートするために、heartbeat では CRM（Cluster Resource Manager）という仕組みを用意しています(図 3-11)。

図 3-11:heartbeat の CRM の構成



PaceMaker（http://www.linux-ha.org/wiki/Pacemaker）は、この CRM をより高度化したソフトウェアです。より複雑なリソース管理が必要な場合に利用することができます。

3.3　ロードシェアリング

システム全体のパフォーマンスを考慮しながら、冗長性も確保しなければならない場合には、ロードシェアリングがよく利用されます。ロードシェアリングでは、前述したアクティブ・スタンバイクラスタリングやマルチサーバでのクラスタリングとは違い、ロードシェアリングされたすべてのサーバの役割が同じです。つまり、ロードシェアリングでは、サービスはすべてのサーバで動作します。

3.3.1　ロードシェアリングのシステム構成

一般的なロードシェアリングクラスタでは、図 3-12 のように**ロードバランサ**（負荷分散装置）がクライアントからのリクエストを受け付けて、各サーバへ処理を分配します。実際の処理を行うサーバを**実サーバ**（Real Server）と呼びます。それに対して、実サーバを組み合わせて作成されるサーバシステム全体を、**仮想サーバ**（Virtual Server）とよびます。

図 3-12:ロードシェアリングのイメージ

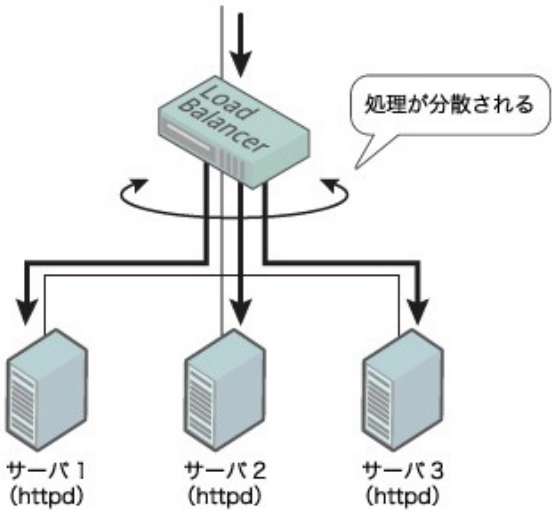
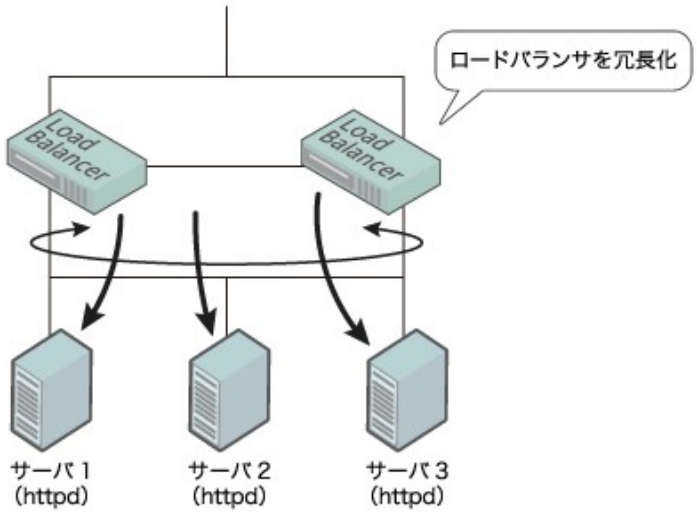


図 3-12 のような構成では、サーバ 1、サーバ 2、サーバ 3 のどのサーバが故障しても、ロードバランサが自動的にそれを検知し、故障サーバを切り離します。そのため、サーバが故障しても、システム全体としては多少パフォーマンスが低下することがあっても、サービスそのものが停止することはありません。

ただし、図 3-12 のような構成では、ロードバランサが単独障害点ですので注意が必要です。フォールトトレランスを実現するには、図 3-13 のようにロードバランサも冗長化する必要があります。

図 3-13:ロードシェアリング（フォルトトレランス）のイメージ



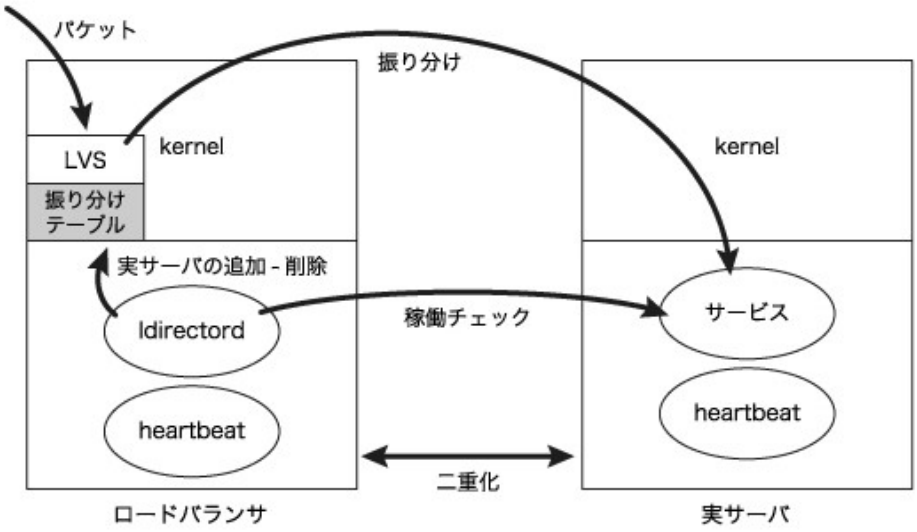
3.3.2　Linux での実装

システム全体として性能を確保し同時に稼働率を向上させるためには、ロードバランサは次のような機能を提供する必要があります。

1. リクエストの振り分け
2. サーバの稼働状況のチェック
3. 実サーバの動的な追加と削除
4. ロードバランサ自身の二重化

ロードバランサは専用のハードウェア製品として販売されていますが、Linux でもロードバランサを作成することができます。Linux では、このうち 1 の機能が LVS という機能名でカーネルに組み込まれています(図 3-14)。また、2～4 の機能は、前述した heartbeat を使って実現することができます。

図 3-14:LVS の構成



heartbeat には、サーバの稼働状況を確認し、LVS へ動的に設定を行うための ldirectord が同梱されています。ldirectord は heartbeat のリソースとして動作するように設計されています。次は、ldirectord の設定ファイルの例です。

▼*ldirectord の設定例（/etc/ha.d/ldirectord.cf）*

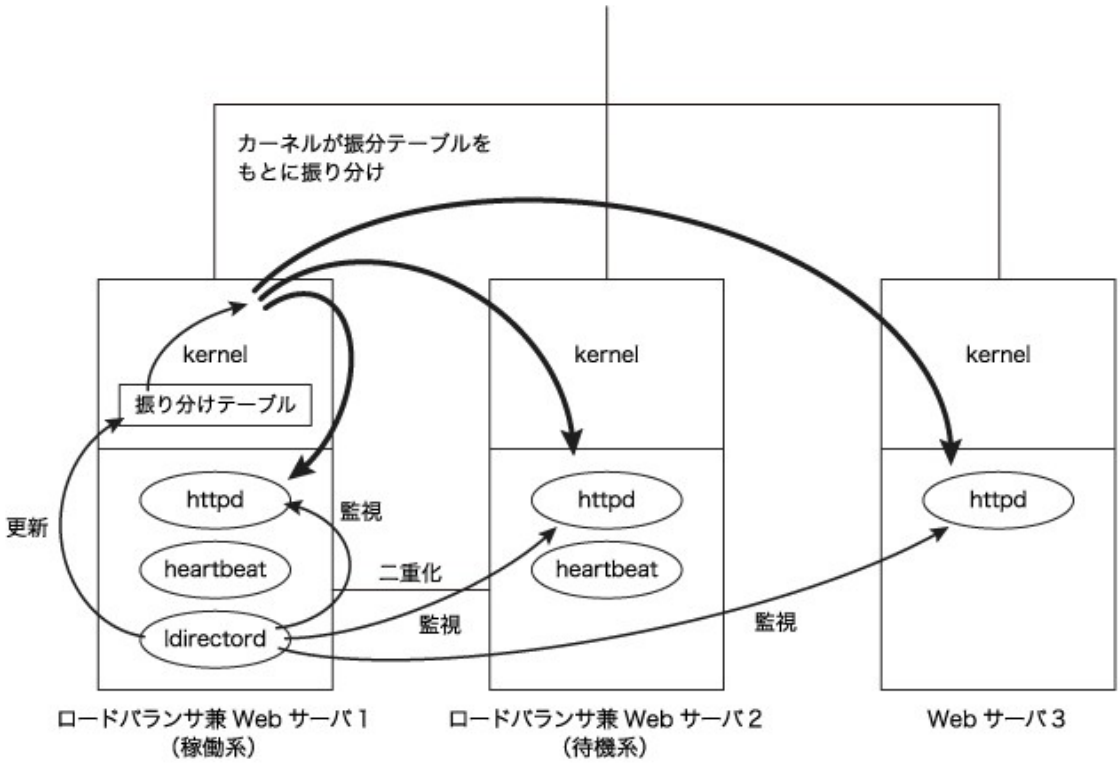
```
# Global Directives
checkinterval=1                          ← 実サーバのチェック間隔
checktimeout=3                           ← 実サーバが動作していないとみなす時間（秒）

# Sample for an http virtual service
virtual=192.168.6.240:80                 ←仮想サービスの IP アドレスとポート番号
        real=192.168.6.2:80 gate         ←実サーバの設定
        real=192.168.6.3:80 gate
        real=192.168.6.6:80 gate
        scheduler=rr                     ←リクエストの配分方法（ラウンドロビン）
        protocol=tcp                     ←実サーバをチェックするプロトコル
        service=http                     ←実サーバをチェックするサービス
        checktype=negotiate              ←実サーバのチェック方法
        checkport=80                     ←実サーバをチェックするポート番号
        virtualhost=www.designet.jp      ←サーバチェックでアクセスする仮想ドメイン名
        request="index.html"             ←サーバチェックでアクセスする URI
        receive="Test Page"              ←サーバチェックのチェック用文字列
```

この例の最後の 3 行では、「http://www.designet.jp/index.html」からホームページを取得し、そのデータに「Test Page」が含まれることまでをチェックする設定を行っています。ldirectord では、このようにサービス毎の稼働チェックを細かく行うことができます。ldirectord は、ftp, smtp, http,

https, pop, imap, nntp, ldap, sip, dns, mysql, pgsql などのサービスを監視することができます。また、これ以外のサービスの場合でも、サーバの TCP ポートが接続可能な状態になっていることを確認することができます。

このように heartbeat の ldirectord の機能を利用することで、Linux サーバでロードバランサを作ることができます。また、図 3-15 のように、ロードバランサとサービスを組み合わせて実現する構成も可能です。

図 3-15:ロードバランサ兼用サーバのイメージ



4章 データの共有

複数のサーバを使ってシステムを冗長化することで、システム全体の稼働率を向上させることができます。しかし、こうしたシステムを実際に作成するときに問題になることが多いのが、データの扱いです。この章では、複数サーバでデータを共有する必要性と、そのための仕組みについて学習します。

4.1 データ共有の必要性

例えばWWWサービスがホームページというデータを扱うように、ネットワークサービスを提供するシステムでは、ほとんどの場合には何らかのデータを扱います。そのため、アクティブ・スタンバイクラスタやロードバランシングなど、複数のサーバを使ってシステムを冗長化する場合には、データの管理をどのように行うのかが非常に重要になります。

管理すべきデータの内容と管理方法はサービスの内容によって異なりますが、主に次のように分類することができます。

- ユーザ管理データ
  システムを利用するユーザやグループと、各ユーザの属性情報のデータです。例えば、ロードシェアリングクラスタのシステムにログインしようとしたときに、どのサーバに接続されたかによってユーザ名とパスワードが異なると、非常に使いにくいシステムになってしまいます。そのため、ユーザ、パスワードなどのユーザ属性データについては、すべてのサーバで同じ情報を参照する必要があります。ユーザデータは、コンピュータへログインできるユーザとしてだけでなく、様々なサービスでも参照されます。それぞれを別に管理することもできますし、一緒に管理することもできます。
  一般に、ユーザ管理データは変更される機会は多くありませんが、参照が多いという特徴があります。そのため、高速に検索できる必要があります。ユーザの登録や変更には多少手間がかかっても、高速にアクセスできる方法で共有する必要があります。

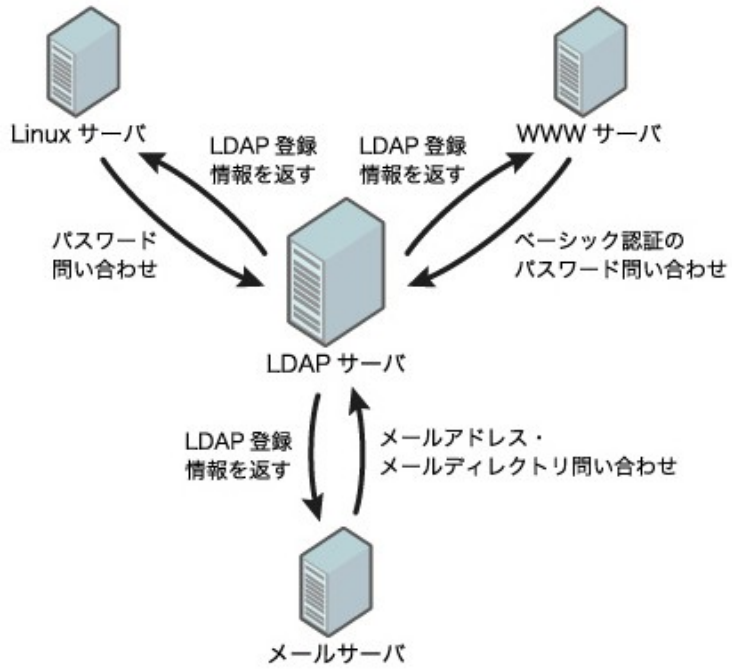
- 参照型データ
  WWWサーバのホームページの情報や、Anonymous FTPサーバのダウンロード用ファイルのように、あまり変更されない静的なデータです。サーバによってデータが違うと混乱の原因となりますので、複数のサーバが同じデータになるように管理する必要があります。しかし、データの一貫性が重要でなければ、データ更新が発生したときに各サーバに変更内容を反映すれば十分です。データの一貫性が重要な場合には、次の更新型データとして扱う必要があります。
  参照型のデータの更新は手動で各サーバに対して行っても構いませんが、扱うファイルの数が多い場合には、自動的に更新する仕組みを導入する場合が多いようです。

- 更新型データ
  ファイルサーバのファイル情報、メールサーバのメールデータ、RDBのデータのように、頻繁に変更されるデータです。こうしたサービスでは、クラスタがフェイルオーバしたときに参照されるデータも最新でなければなりません。また、ロードバランシングなどで各サーバがアクセスするデータも常に最新でなければなりません。そのため、更新されたデータがリアルタイムに各サーバに反映されるように、何らかの仕組みを導入する必要があります。

4.2　ユーザ情報の共有（LDAP）

複数のシステムでユーザ情報を統一して管理する方法としてよく利用されるのが、LDAP（Lightweight Directory Access Protocol）です。図 4-1 は、LDAP によるユーザ情報共有のイメージです。

図 4-1:LDAP によるユーザ情報共有のイメージ

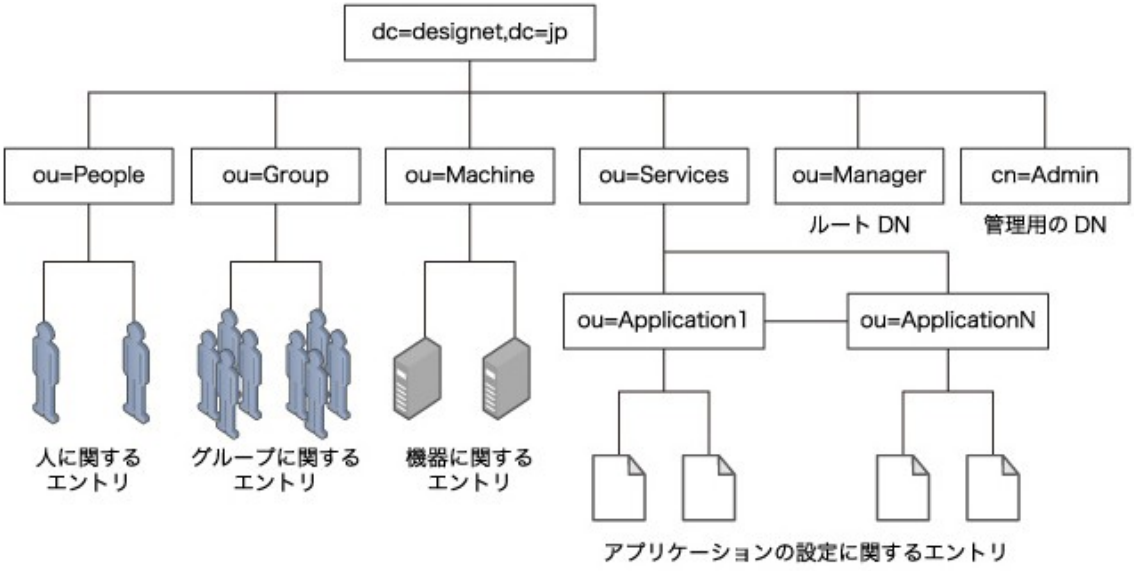


LDAP は、その名のとおりディレクトリアクセスのためのプロトコルです。ここでいうディレクトリは、電話帳や住所録のような意味で、人に関する情報を管理するための構造のことを指しています。つまり、LDAP は人に関する情報を管理するためのデータベースであるといえます。ただし、LDAP はリレーショナルデータベースではありません。

4.2.1　LDAP のデータ形式

図 4-2 は、LDAP での情報の管理を示したものです。

図 4-2:LDAP DITの例



この例では、様々な情報がツリー構造で管理されています。ツリー構造の各項目は**エントリ**とよばれます。そして、トップにある「dc=designet,dc=jp」というエントリを**root エントリ**とよびます。また、このようなツリー構造を**DIT**（Directory Information Tree）とよびます。このようなツリー構造で情報を管理するのがLDAPの特徴です。

LDAPでは、各エントリには**RDN**（Relative Distinguished Name)という識別名を付けて管理します。RDNは階層構造を表すために、上位のRDNを含む形式で表現されます。例えば、図4-2の「人に関するエントリ」を管理するためのエントリは、「ou=People,dc=designet,dc=jp」のように上位のエントリをカンマで区切って表記します。これを**DN**（Distinguished Name)とよびます。DNは、DIT全体の位置を表すのです。これは、表記順が逆ではありますが、/usr/bin/lsのようにファイル名のパスを表すために、上位のディレクトリを/で区切って表記するのに似ています。

また、図4-2の「ou=People,dc=designet,dc=jp」のように、エントリを格納するためだけに用意されたエントリを**コンテナ**と呼ぶこともあります。これは、ファイルシステムで言うディレクトリにあたる概念です(表4-1)。

▼表 4-1:ファイルシステムのツリー構造とLDAP DITの比較

ファイルシステム	LDAP DIT
ファイル	エントリ
ディレクトリ	コンテナ
ファイル名	RDN
絶対パス	DN

ファイルシステムでは、ファイルにはどのようなデータでも記載することができます。これに対して、LDAP のエントリでは厳密にデータの型が決められています。エントリには、**属性（attribute）**を**属性タイプ**と**属性値**のペアで登録します。また、LDAP ではエントリに登録できる属性タイプの種類をあらかじめ決めることができます。この形式のことを**オブジェクトクラス**（objectClass）と呼びます。オブジェクトクラスは、管理するデータの内容によって「人を管理するためのオブジェクトクラス」「グループを管理するためのオブジェクトクラス」というように、用途に合わせて様々な形式を定義することができます。

▼*LDAP エントリの例*

```
#
# コメント
#
dn: uid=admin,ou=People,dc=designet,dc=jp                    ← (1)
objectClass: account                                         ← (2)
objectClass: posixAccount                                    ← (3)
cn: admin user
uid: admin
userPassword: {CRYPT}038Ac9UDYRM5U
uidNumber: 1000
gidNumber: 1000
loginShell: /bin/bash
homeDirectory: /home/admin
description: This user is a administrator on DesigNET domain.
        If you have some problem, you can call to him.
```

これは、LDAP エントリに登録するデータの例です。（1）のように DN も属性の 1 つとしてエントリに登録されます。また、（2）（3）では、このエントリで利用することのできる属性型を決めるオブジェクトクラスが宣言されています。また、このように LDAP のデータをテキストで表した形式を**LDIF**（LDAP Data Interchange Format）と呼びます。

4.2.2　Linux での実装

Linux 上で動作する LDAP サーバとして、もっともよく利用されているのが OpenLDAP です。OpenLDAP は、実際に様々な Linux ディストリビューションに標準的に採用されています。OpenLDAP は、OpenLDAP Foundation が運営する OpenLDAP Project（http://www.openldap.org）が開発を行っている LDAP サーバです。OpenLDAP では、LDAP サーバだけでなく、LDAP のデータを管理するためのユーティリティプログラムや、様々なアプリケーションから使うことのできる API をライブラリとして公開しています。最近のほとんどの Linux では、ユーザ管理で LDAP が利用できるようにするため、OpenLDAP ライブラリが標準的に利用できるように構成されています。

OpenLDAP が提供する LDAP サーバプログラムは slapd ですが、LDAP はデータベースですので単純に slapd を起動するだけではユーザ管理データベースとして利用することができません。最低でも、次のような設定を行う必要があります。

- LDAP サーバの root エントリ（suffix）、管理者 DN（rootdn）とパスワード（rootpw）を設定します。
- LDAP サーバを起動します。
- root エントリと管理者 DN のデータを登録します。
- LDAP で管理するデータに合わせて基本コンテナを登録し、DIT を作成します。
- ユーザやグループのデータを登録します。

次は、slapd の設定ファイルの例です。

▼*/etc/openldap/slapd.conf の例*

```
include         /etc/openldap/schema/core.schema                    ←  (1)
include         /etc/openldap/schema/cosine.schema
include         /etc/openldap/schema/inetorgperson.schema
include         /etc/openldap/schema/nis.schema

allow bind_v2

pidfile         /var/run/openldap/slapd.pid
argsfile        /var/run/openldap/slapd.args

database        bdb
suffix          "dc=designet,dc=jp"                                 ←  (2)
rootdn          "cn=Manager,dc=designet,dc=jp"                      ←  (3)
rootpw          {SSHA}A/9vWhxe6Ek7JWI0iwQJDnr8QgOqKayF              ←  (4)

directory       /var/lib/ldap

index objectClass                       eq,pres                     ←  (5)
index ou,cn,mail,surname,givenname      eq,pres,sub
index uidNumber,gidNumber,loginShell    eq,pres
index uid,memberUid                     eq,pres,sub
index nisMapName,nisMapEntry            eq,pres,sub
```

（1）は、LDAP のオブジェクトクラスを定義したファイル（**スキーマ**）の読み込みです。OpenLDAP には、RFC で規定された様々なスキーマが付属していますが、必要に応じてそれを読み込みます。（2）（3）（4）は、DIT のトップの DN、管理者 DN、管理者パスワードの設定です。管理者パスワードは、slappasswd コマンドで作成することができます。また、（5）は検索で利用するインデックスの定義です。

設定が終わったら、slapd を起動し、root エントリ、管理者 DN、基本コンテナを登録し、最後に ldapadd コマンドでユーザのデータを登録します。次は、これらのエントリの例です。

▼*root エントリと管理者の DN の LDIF の例*

```
dn: dc=designet,dc=jp
```

```
objectClass: organization
objectClass: dcObject
o: DesigNET, INC.
dc: designet

dn: cn=Manager,dc=designet,dc=jp
objectClass: organizationalRole
cn: Manager
```

▼*基本コンテナの LDIF の例*

```
dn: ou=People,dc=designet,dc=jp
objectClass: organizationalUnit
ou: People

dn: ou=Services,dc=designet,dc=jp
objectClass: organizationalUnit
ou: Services
```

▼*ユーザエントリの LDIF の例*

```
dn: uid=admin,ou=People,dc=designet,dc=jp
objectClass: account
objectClass: posixAccount
cn: admin user
uid: admin                                    ← ユーザ名
userPassword: {CRYPT}038Ac9UDYRM5U            ← パスワード
uidNumber: 1000                               ← UID
gidNumber: 1000                               ← GID
loginShell: /bin/bash                         ← シェル
homeDirectory: /home/admin                    ← ホームディレクトリ
```

▼*エントリの登録の例*

```
$ ldapadd -x -D "cn=Manager,dc=designet,dc=jp" -W -f init.ldif
Enter LDAP Password: ******                   ← 管理者 DN のパスワード
adding new entry "dc=designet,dc=jp"

adding new entry "cn=Manager,dc=designet,dc=jp"
```

4.2.3　管理用ソフトウェア

LDAP のデータ管理を行うための管理インタフェースのソフトウェアとして、いくつかのソフトウェアがオープンソースで公開されています。

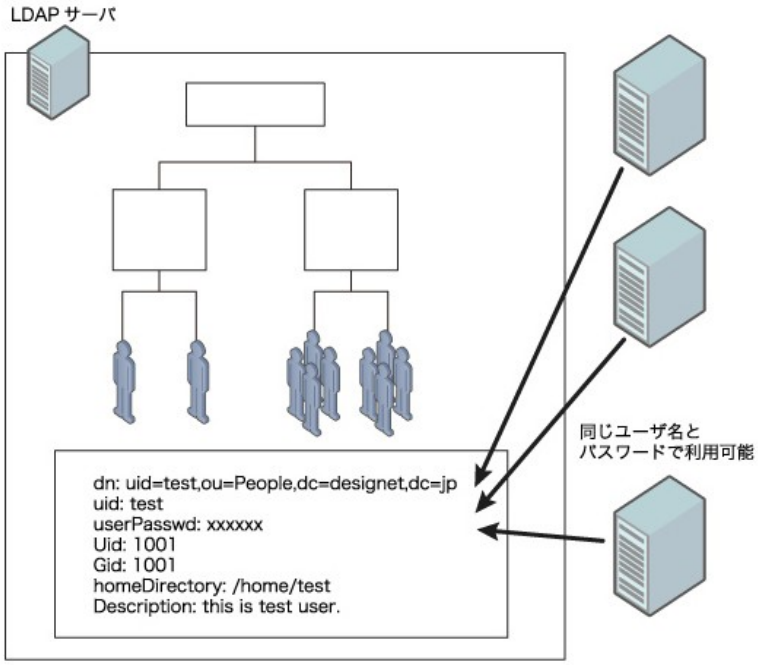
phpLDAPadmin (http://phpldapadmin.sourceforge.net/) は、Web ブラウザを用いて LDAP の管理を行う Web ベースのアプリケーションです。phpLDAPadmin は、LDAP のディレクトリツリーを視覚的に分かりやすく表示することができます。また、日本語にも対応しています(図 4-3)。

図 4-3:phpLDAPadmin

4.2.4　システムユーザとの連携

ほとんどの Linux ディストリビューションは、LDAP によるユーザ管理に対応しています。LDAP でユーザを管理すると、/etc/passwd, /etc/group ファイルに替わって、LDAP に登録されたデータを参照します。この機能を利用すると、ネットワーク内のどのサーバへも同じユーザとパスワードでアクセスすることができるようになります(図 4-4)。

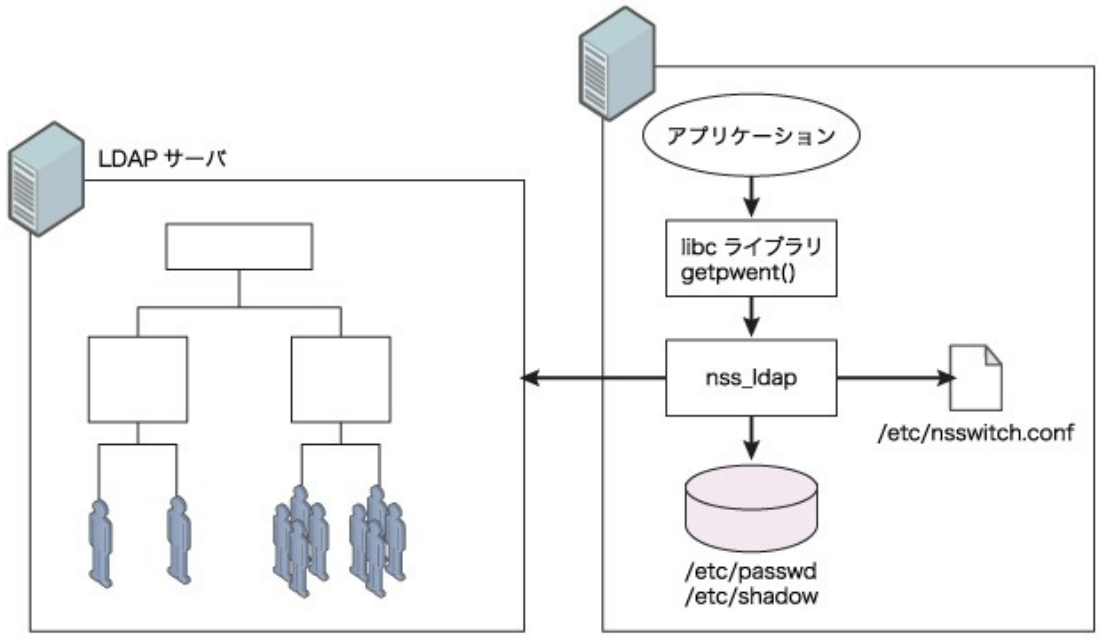
図 4-4:LDAP によるシステムユーザの管理



Linux での実装

ほとんどの Linux や Unix では、ユーザ認証に LDAP を利用するための仕組みとして、nss_ldap を利用することができます。nss_ldap では、パスワードファイルからの情報を取得するライブラリ関数 getpwent(3) に実装された NSS Library 機能が LDAP サーバと連携します。そのため、getpwent()などのパスワードユーティリティ関数を使って、パスワード情報を取得する多くのアプリケーションを一括して LDAP と連携させることができます(図 4-5)。

図 4-5:nss_ldap による LDAP サーバとの連携



nss_ldap を利用するためには、次の設定を行う必要があります。

- システムが利用する LDAP サーバの情報(/etc/ldap.conf)
- システムの認証設定(/etc/nsswitch.conf)

次は、その設定例です。

▼*nss_ldap の設定例 (/etc/ldap.conf)*

```
host                    127.0.0.1
port                    389
ldap_version            3
base                    dc=designet,dc=jp
binddn                  cn=Manager,dc=designet,dc=jp
bindpw                  secret
scope                   sub
crypt                   des
ssl                     no
bind_policy             soft
```

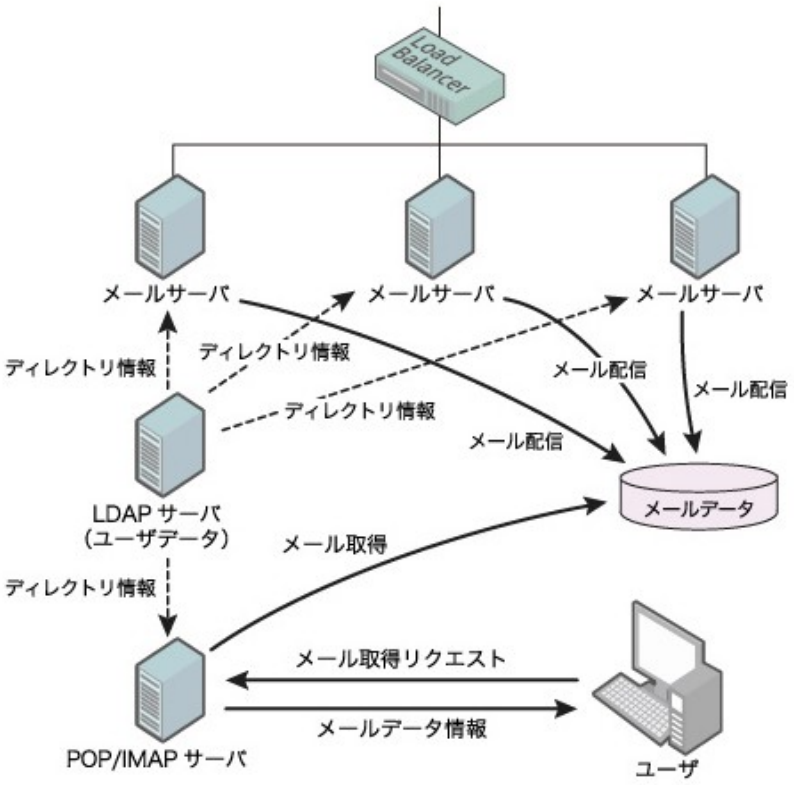
▼*nss_ldap の設定例 (/etc/nsswitch.conf)*

```
passwd:     files ldap
shadow:     files
group:      files ldap
```

4.2.5　メールサーバでの利用

アクティブ・スタンバイやロードシェアリングの技術を使って、メールサーバの稼働率を向上させる場合には、メールデータの共有とともにメールユーザを共有する必要があります。LDAP サーバを使ってユーザを管理することで、システム内のすべてのサーバで同じユーザ情報を利用することができます(図 4-6)。

図 4-6:メールサーバのロードシェアリングと LDAP サーバ



多くの Linux で利用されている MTA である Postfix は、メールユーザを Linux のユーザとは別に管理することができ、これを仮想メールボックスと呼んでいます。仮想メールボックスでは、ユーザ情報を LDAP と連携して管理することができます。仮想メールボックスを利用するためには、次のような設定をする必要があります。

- メールを配送するユーザの情報を LDAP に登録します。
- postfix の設定ファイル（/etc/postfix/main.cf）に仮想メールボックスの設定を行います
- POP/IMAP サーバに仮想メールボックスを参照する設定を行う。
- LDAP ユーザ管理用のツールを用意する

メール配送ユーザのLDAP情報

メールを配送するユーザの情報をLDAPに登録します。最低限必要なのは、どのメールアドレスを誰が受けとるのかという情報です。また、POP/IMAPでアクセスするためにユーザのパスワードも登録する必要があります。次は、そうしたLDAPエントリの例です。mail属性にユーザのメールアドレスが登録しています。

▼*Postfix仮想メールボックス用のLDAPエントリの例*

```
dn: uid=ldapuser,ou=People,dc=designet,dc=jp
objectClass: inetOrgPerson
objectClass: simpleSecurityObject
uid: ldapuser
sn: user
cn: ldap
userPassword: {CRYPT}wdE4h0I3hrpsU
mail: ldapuser@designet.jp
```

Postfixの設定

Postfixには仮想メールボックスの設定を行います。次の例のように、仮想メールボックスを利用するドメインを登録し、メールを管理するアカウントのUIDやGIDを設定します。

▼*postfixの仮想メールボックスの設定例（/etc/postfix/main.cfの一部）*

```
# 仮想メールボックスに配送するドメインの設定
virtual_mailbox_domains = designet.jp

# 仮想メールボックスの配送先の設定
virtual_mailbox_base = /home/vmail
virtual_mailbox_maps = ldap:/etc/postfix/ldap-account.cf

# メール保管アカウントの設定
virtual_uid_maps = static:400
virtual_gid_maps = static:400
```

設定ファイル中のvirtual_mailbox_mapsでは、メールの配送を行うときにldapデータベースを参照するように設定しています。この例では、/etc/postfix/ldap-account.cfにLDAPサーバへの接続情報を設定するようにしています。次は、その設定例です。

▼*postfixの仮想メールボックスの設定例（/etc/postfix/ldap-account.cfの一部）*

```
server_host = 127.0.0.1
server_port = 389
bind = yes
bind_dn = cn=Manager,dc=designet,dc=jp
bind_pw = secret
scope = sub
```

```
search_base = dc=designet,dc=jp
query_filter = (|(mail=%s)(mailAlias=%s))
result_attribute = uid
result_format = %s/Maildir/
```

この例では、メールのデータは、/home/vmail/<uid>/Maildir に配送されます。

POP/IMAP サーバソフトウェアも、ほとんどのソフトウェアが LDAP に対応しています。次は、dovecot の LDAP 連携設定の例です。

▼*dovecot の LDAP 連携設定（/etc/dovecot.conf の認証部分）*

```
auth default {
  mechanisms = plain

  passdb ldap {
     args = /etc/dovecot-ldap.conf
  }

  userdb ldap {
     args = /etc/dovecot-ldap.conf
  }
  user = root
}
```

設定ファイル中の passdb, userdb は、それぞれパスワードデータベースとユーザデータベースの設定です。この例では、ldap を参照し、LDAP サーバへの接続情報は/etc/dovecot-ldap.conf に設定することになっています。次は、その/etc/dovecot-ldap.conf の設定例です。

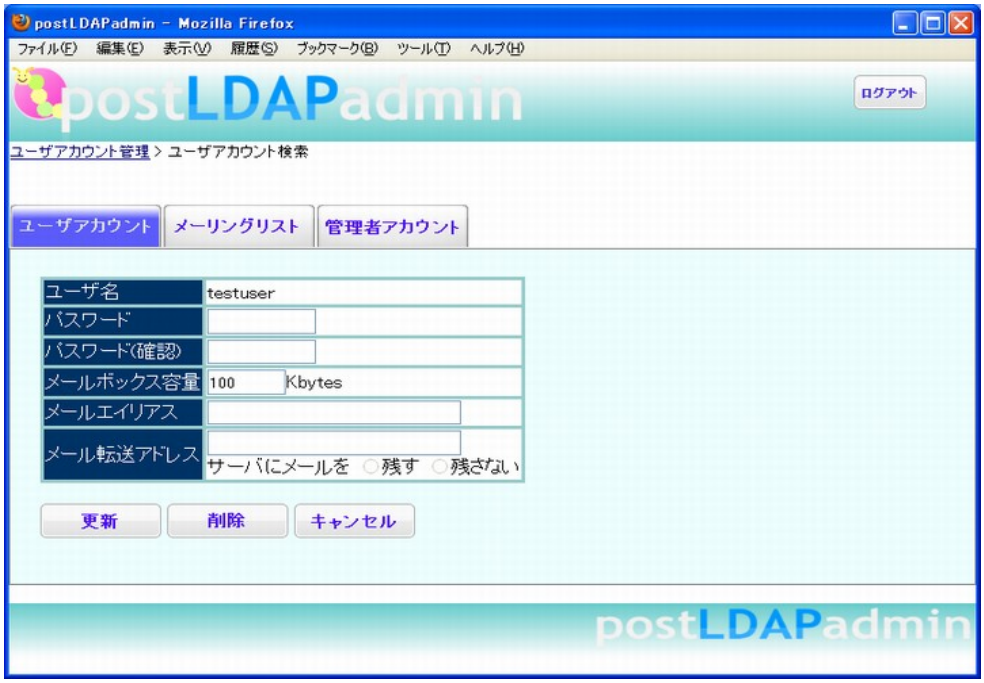
▼*dovecot の LDAP 設定（/etc/dovecot-ldap.conf）*

```
hosts = 127.0.0.1
dn = cn=Admin,dc=designet,dc=jp
dnpass = admin
auth_bind = no
base = ou=People,dc=designet,dc=jp
scope = subtree
pass_attrs = uid=user, userPassword=password
pass_filter = (uid=%u)
```

postLDAPadmin

postLDAPadmin（http://sourceforge.jp/projects/postldapadmin/）は、メールサーバ（Postfix）や POP/IMAP サーバが LDAP と連携して動作するために必要な情報を Web ベースで管理するためのアプリケーションです。メールユーザの管理のために特化しているのが特徴です (図 4-7)。

図 4-7:postLDAPadmin

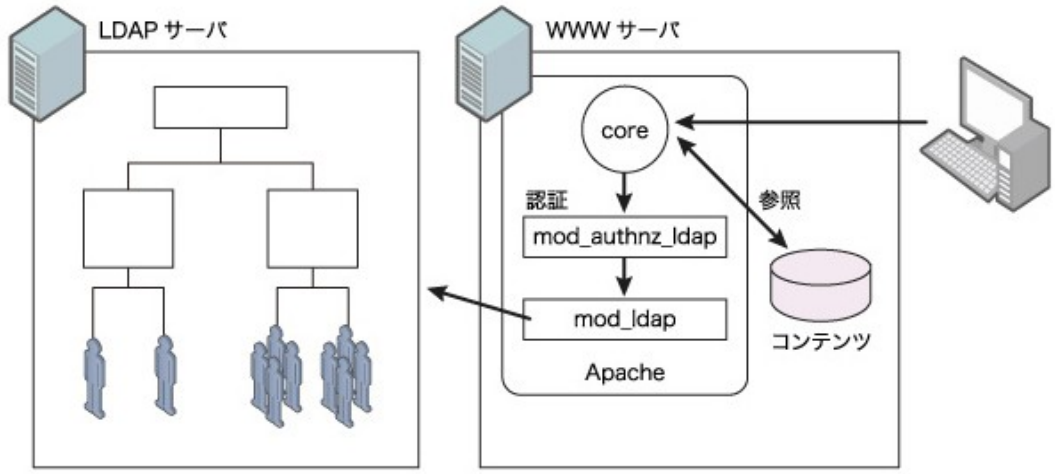


4.2.6　WWW サーバでの利用

ほとんどの WWW サーバでは、ユーザ認証によってコンテンツへのアクセスを制限する機能を持っています。ロードシェアリングを使って WWW サーバを冗長化する場合には、このユーザ認証で利用するユーザやパスワードのデータも LDAP サーバと連携して管理する必要があります。

Linux の WWW サーバとして、ほとんどのディストリビューションが採用している Apache では、mod_authnz_ldap という LDAP 連携用のモジュールが用意されていて、LDAP サーバと連携することができます(図 4-8)。

図 4-8:WWW サーバでの LDAP の利用



Apache と LDAP サーバを連携するためには、次のような設定を行う必要があります。

- LDAP モジュール、LDAP 認証モジュールの読み込み
- LDAP を使ったベーシック認証の設定

次は、Apache の LDAP 連携設定の例です。

▼*Apache の LDAP モジュールの読み込み設定（/etc/httpd/conf/httpd.conf の一部）*

```
LoadModule ldap_module modules/mod_ldap.so
LoadModule authnz_ldap_module modules/mod_authnz_ldap.so
```

▼*Apache の LDAP によるベーシック認証の設定（/etc/httpd/conf.d/auth.conf）*

```
<Directory "/var/www/html/admin">
    Options Indexes FollowSymLinks

    AllowOverride None

    Order allow,deny
    Allow from all

    AuthName "LDAP user authentication"            ← 認証のときに画面に表示する認証名
    AuthType basic                                 ← 認証のタイプ

    AuthBasicProvider ldap                         ← ベーシック認証のデータベース指定
    AuthLDAPBindDN cn=Admin,dc=designet,dc=jp      ← LDAP サーバへの接続で使うバインド DN
    AuthLDAPBindPassword admin                     ← バインド DN のパスワード
```

```
    AuthLDAPURL ldap://127.0.0.1/ou=People,dc=designet,dc=jp  ← LDAP サーバと検索条件を
                                                                示す URL
    require ldap-attribute host=test.designet.jp          ←ページを参照するための許可条件
</Directory>
```

▼*LDAP エントリの例*

```
dn: uid=testuser,ou=People,dc=designet,dc=jp
objectClass: account
objectClass: simpleSecurityObject
uid: testuser
userPassword: {CRYPT}PI8dX0mqSzbZA
host: test.designet.jp
host: fc7.designet.jp
```

4.3　サーバ間のデータの同期（rsync）

参照型のデータを扱うシステムでは、データ更新の頻度は低く、緊急性も高くありません。そのため、サーバの変更が発生したときに、手動でサーバ間のデータの同期を行う仕組みが利用されることが多いようです。

サーバ間でデータのコピーを行う方法としては、rcp, scp を利用することができます。しかし、これらのコマンドでは全データをコピーしますので、ファイルの数が多くなるとデータ更新に時間がかかるようになります。こうしたケースでデータ同期の仕組みとしてよく利用されるのが rsync です。

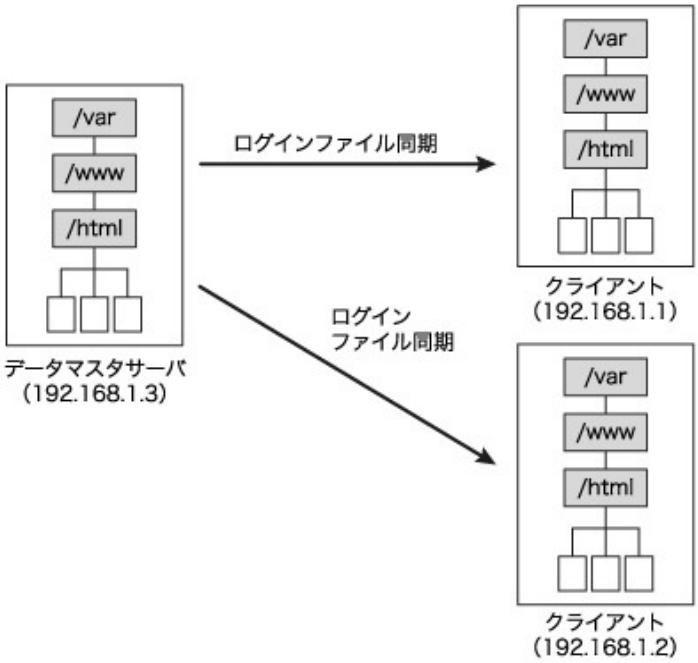
4.3.1　rsync の概要

rsync は、Andrew Tridgell 氏と Paul Mackerras 氏が開発したユーティリティプログラムで、ネットワーク間でファイルの同期を取るために使われます。rcp や scp では無条件にファイルやディレクトリをコピーするのに対して、rsync では更新されたファイルだけを処理することができます。

Rsync では次の 4 つのモデルでデータを同期することができます。

- ログインモデル（push 式）
  データマスタサーバから配布先のクライアントへログインし、任意のファイルの同期を取る方法です。クライアント側のユーザ認証による厳密なアクセスチェックを行うことができます。クライアント上のすべてのファイルが同期可能になります(図 4-9)。

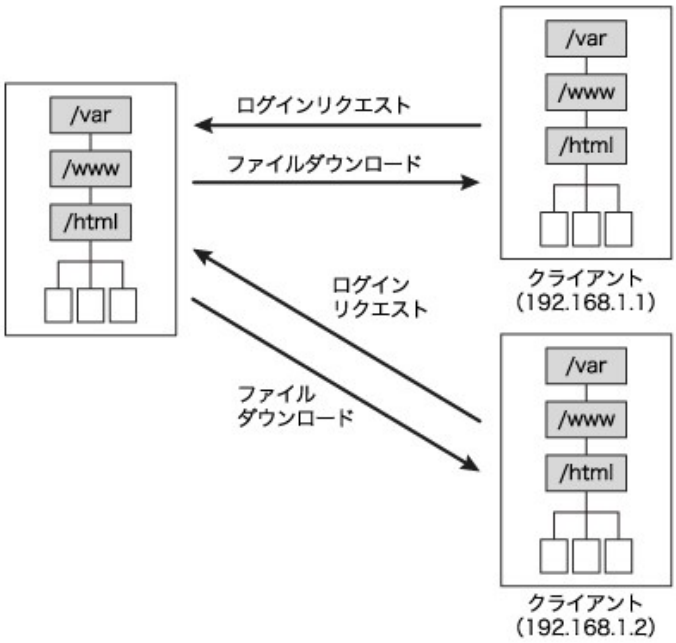
図 4-9: rsync によるログインモデル push 式でのファイル同期



- ログインモデル（pull 式）

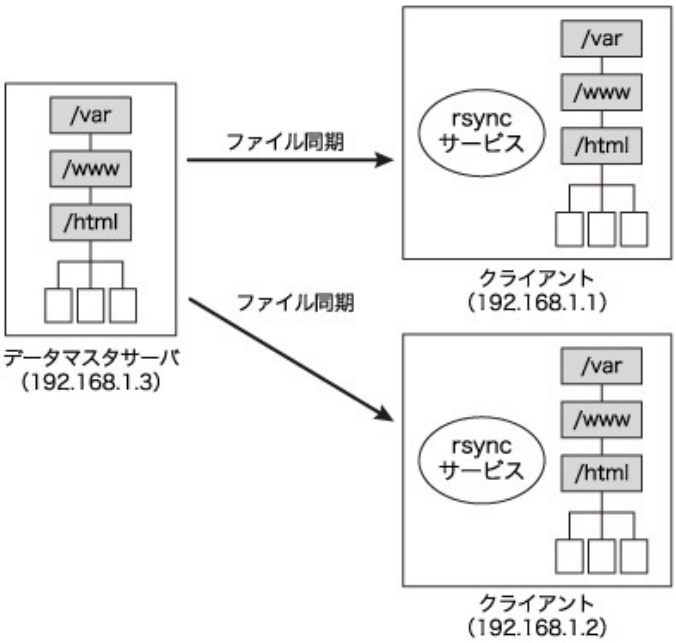
データを管理するデータマスタサーバへログインし、任意のファイルの同期を取る方法です。サーバの IP アドレスに加えて、ユーザ認証による厳密なアクセスチェックを行うことができますが。データマスタサーバ上のすべてのファイルが同期可能になります(図 4-10)。

図 4-10: rsync によるログインモデル pull 式でのファイル同期



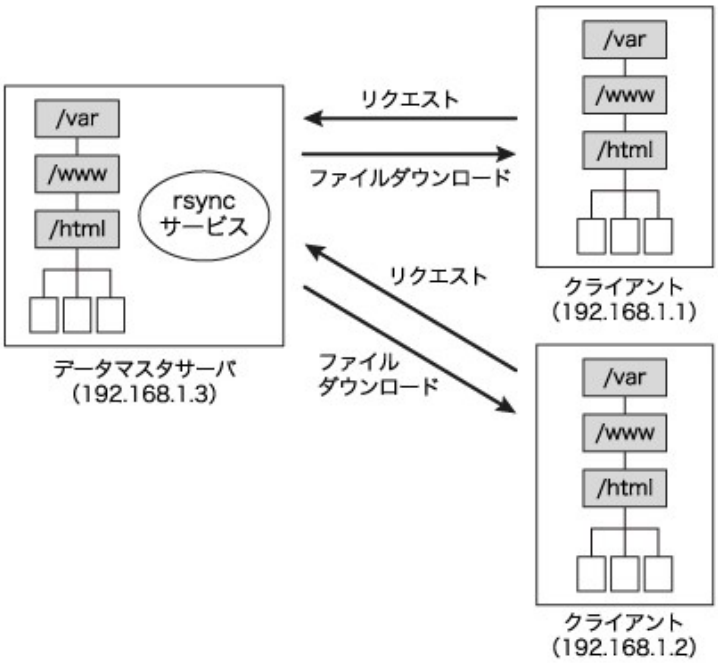
- サーバモデル（push 式）
  クライアントの限られた領域だけを公開し、マスタサーバからデータを更新します。クライアントへのログインアカウントは必要ありません。データの同期は、サーバから強制的に行われます(図 4-11)。

図 4-11：rsync によるサーバモデル push 式でのファイル同期



- サーバモデル（pull 式）
  マスタサーバの限られた領域だけを公開します。サーバの IP アドレスでのアクセス制限を行うことができるため、必要なクライアントだけに情報を公開することができます。データマスタサーバへのログインアカウントは必要ありません。データの同期は、クライアントからのリクエストで行われます(図 4-12)。

図 4-12:rsyncによるサーバモデルpull式でのファイル同期



どの方法を使う場合でも、rsyncはデータマスタサーバとクライアントの両方にインストールされている必要があります。

4.3.2　ログインモデル（push式）

データマスタサーバから、各クライアントへログインしてサーバ上のデータを強制的に配布する方式です。すべてのクライアントに、ログイン用のアカウントが必要です。ファイルの所有者やグループを保持してデータをコピーするためには、データマスタサーバからクライアントへrootでログインする必要があります。そのため、クライアントの数がそれほど多くなく、データマスタサーバが完全に信用できる場合にしか利用されません。次のような条件を整備する必要があります。

- クライアント上のログインユーザ
  同期するデータを管理するユーザを決め、必要であれば作成します。
- データマスタサーバからのリモートログインの設定
  データマスタサーバからリモートログインできる状態を作ります。一般的には、クライアント上でrshサービスまたはsshサービスを起動し、データマスタサーバからのアクセスとログインを許可します。
- ログインユーザの設定
  必要に応じてログインユーザのrshやsshの環境設定を行います。環境設定を適切に行えば、パスワード入力を行わずにコピーを行うこともできます。

次のような書式でファイルの同期を取得することができます。

▼*rsh を利用する場合の書式*

```
rsync [OPTION] SRC... [USER@]HOST:DEST
```

▼*ssh を利用する場合の書式*

```
rsync [OPTION] --rsh=ssh SRC... [USER@]HOST:DEST
```

次は ssh を利用して rsync を行う場合の実行例です。パスワード認証を行った後、ファイルの同期をしています。

▼*ssh を利用した場合の実行例*

```
# rsync -a /var/www/html/ root@192.168.1.1:/var/www/html/
root@192.168.1.1's password:                    ← パスワードを入力
```

4.3.3　ログインモデル（pull 方式）

クライアントからデータマスタサーバへログインし、データをダウンロードするモデルです。サーバへのログインアカウントは、すべてのクライアントで同じにすることもできますし、別々にすることも可能です。また、ログインユーザにファイルを読み込むアクセス権さえあれば、クライアントの root ユーザで rsync コマンドを実行することで、ファイルの所有者やグループを保持してデータをコピーすることができます。ログインユーザが読むことのできるファイルであれば、サーバ上のすべてのデータがコピーできてしまうため注意が必要です。次のような条件を整備する必要があります。

- サーバ上のログインユーザ
  同期するデータを管理するユーザを決め、必要であれば作成します。
- データマスタサーバへのリモートログインの設定
  データマスタサーバへリモートログインできる状態を作ります。一般的には、データマスタサーバ上で rsh サービスまたは ssh サービスを起動し、クライアントからのアクセスとログインを許可します。
- ログインユーザの設定
  必要に応じてログインユーザの rsh や ssh の環境設定を行います。環境設定を適切に行えば、パスワード入力を行わずにコピーを行うこともできます。

次のような書式でファイルの同期を取得することができます。

▼*rsh を利用する場合の書式*

```
rsync [OPTION...] [USER@]HOST:SRC... [DEST]
```

▼*ssh を利用する場合の書式*

```
rsync --rsh=ssh [OPTION...] [USER@]HOST:SRC... [DEST]
```

次は ssh を利用して rsync を行う場合の実行例です。

▼*sshを利用した場合の実行例*

```
# rsync -a root@192.168.1.3:/var/www/html/ /var/www/html/
root@192.168.1.3's password:                    ← パスワードを入力
```

4.3.4　サーバモデル（push方式）

データマスタサーバ側から強制的にデータを配布する方式です。各クライアントでは rsync サービスを起動し、設定されたエリアに対してのみ更新を許可します。該当サービスに接続できる機器からは、自由にデータを更新できてしまいますので、IP アドレスによるアクセス認証は必須です。次のような条件を整える必要があります。

- クライアント上で、rsync サービスを起動しておきます。
- rsync サービスへアクセスできるホストを限定します。
- rsync デーモンがアクセスできる領域を決め、書き込みができるようにアクセス権を設定します。

rsync サービスは、xinetd から起動されるサービスです。次は、xinetd への設定例です。

▼*/etc/xinetd.d/rsyncの設定例*

```
# default: off
# description: The rsync server is a good addition to an ftp server, as it ¥
#       allows crc checksumming etc.
service rsync
{
        disable= no
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/bin/rsync
        server_args     = --daemon
        log_on_failure  += USERID
}
```

アクセス制御は、xinetd の機能を利用して行います。TCP Wrapper が有効になっている機器では、/etc/hosts.deny, /etc/hosts.allow で行うことができます。

▼*/etc/hosts.denyの設定例*

```
rsync: ALL
```

▼*/etc/hosts.allowの設定例*

```
rsync: 192.168.1.3
```

rsync デーモンの設定は、/etc/rsyncd.conf で行います。

▼/etc/rsyncd.conf の設定例

```
#
# Global
#
uid = apache                    ← ファイルを管理するユーザ ID
gid = apache                    ← ファイルを管理するグループ ID

#
# www file modules
#
[wwwfiles]                      ← ファイルを管理する単位（モジュール）
       path = /var/www/html     ← 管理対象のディレクトリ
       use chroot = no
       read only = no           ← 書き込みを許可
```

この例では、「wwwfiles」という名称でファイルを管理する単位を指定しています。rsync では、これをモジュールとよびます。データマスタサーバからクライアントへデータを push するため、この例のようにモジュールへの書き込みを許可しておく必要があります。

次のような書式でファイルの同期を取得することができます。

▼ *マスタサーバから rsync を利用する場合の書式*

```
rsync [OPTION...] SRC... [USER@]HOST::DEST
rsync [OPTION...] SRC... rsync://[USER@]HOST[:PORT]/DEST
```

DEST には、クライアントに設定されているモジュール名を設定します。次は ssh を利用して rsync を行う場合の実行例です。

▼ *ssh を利用した場合の実行例*

```
# rsync -a /var/www/html/ rsync://192.168.1.1/wwwfiles/
```

4.3.5　サーバモデル（pull 方式）

データマスタサーバに配置されているデータを、クライアントから必要に応じてダウンロードするモデルです。データマスタサーバへのアクセスにログイン認証は必要ありません。また、通常はクライアントからデータマスタサーバへの更新は禁止します。IP アドレスによるアクセス制限を実施することもできますが、広く不特定の人にデータを公開することも可能です。

次のような条件を整備する必要があります。

- データマスタサーバ上で、rsync サービスを起動しておきます。
- rsync サービスへアクセスできるホストを限定します。

rsyncデーモンがアクセスできる領域とアクセス権を設定します。rsyncの設定方法については、設定するサーバがデータマスタになったこと以外は、前述したサーバモデル(push方式)の場合と同様です。アクセス制御には、許可するクライアントすべてを設定する必要があります。

▼*/etc/hosts.denyの設定例*

```
rsync: ALL
```

▼*/etc/hosts.allowの設定例*

```
rsync: 192.168.1.1 192.168.1.2
```

rsyncデーモンの設定は、/etc/rsyncd.confで行います。

▼*/etc/rsyncd.confの設定例*

```
#
# Global
#
uid = apache                        ← ファイルを管理するユーザID
gid = apache                        ← ファイルを管理するグループID

#
# www file modules
#
[wwwfiles]                          ← ファイルを管理する単位（モジュール）
       path = /var/www/html         ← 管理対象のディレクトリ
       use chroot = no
       read only = yes              ← 書き込みを禁止
```

次のような書式でファイルの同期を取得することができます。

▼*クライアントからrsyncを利用する場合の書式*

```
rsync [OPTION...] rsync://[USER@]HOST[:PORT]/SRC... [DEST]
rsync [OPTION...] [USER@]HOST::SRC... [DEST]
```

SRCには、クライアントに設定されているモジュール名を設定します。次はsshを利用してrsyncを行う場合の実行例です。

▼*sshを利用した場合の実行例*

```
rsync -a rsync://192.168.1.3/wwwfiles/ /var/www/html/
```

4.4 NAS 共有ストレージ（NFS）

更新型のデータを管理するためによく使われるのが、NAS（Network Attached Storage)です。NAS というのは、実際にはネットワーク経由で利用する記憶装置の総称で、いわゆるファイルサーバのことです。ネットワークが登場した頃からファイルシステムをネットワーク上で共有したいという要望があり、歴史的にも、多くの企業や団体がファイル共有を実現する様々なプロトコルを開発してきました。そのため、現在でも次のような複数のプロトコルが使われています。

- NFS
  NFS（Network File System）は、Linux をはじめとする Unix 系のほとんどの OS でサポートされているプロトコルです。Sun Microsystems 社が 1984 年に公開した NFS version 2 から利用されており、もっとも古くから使われているファイルシステムです。現在は、NFS version 3 と NFS version 4 が使われていて、Internet Engineering Task Force が開発、管理しています。Linux では、NFS の機能は標準的に組み込まれています。NFS は、もともと Unix 上で動作するように作られているため、シンボリックリンクやファイルキャッシュなど、基本的なファイルシステムの仕組みはすべて実現されています。ただし、ファイルロックのサポートは限定的です。また、アクセス認証を IP アドレス単位でしか行えないなど、設計が古い部分もあり、インターネットなどのセキュリティに配慮する必要のあるネットワークでの利用は推奨されていません。社内ネットワークなどの安全なネットワークで利用する必要があります。
- CIFS
  CIFS（Common Internet File System）は、Microsoft が開発した、いわゆる Windows ファイル共有で利用されるプロトコルです。以前は SMB（Server Message Block）と呼んでいました。Linux では Samba を利用して実装することができます。NFS に比べ、IP アドレス単位でのアクセス認証に加え、接続時のユーザ認証も実施します。2006 年に Windows Vista 向けにリリースされた SMB2.0 ではシンボリックリンクをサポートするようになりましたが、もともと Windows 用のプロトコルですので、Unix や Linux のファイルシステムとして必要な要件をすべて満たしてはいません。また、ユーザ認証を行う代わりにユーザ毎に接続を張る必要があるため、サーバなどのマルチユーザのシステムでの利用には向いていません。最近では、GNOME に CIFS の機能が統合されていますので、デスクトップ用途で利用されることが多いようです。
- AFS
  AFS（Apple File Sharing）は、アップルコンピュータが開発した MacOS、MacOSX のためのファイル共有プロトコルです。Linux でも netatalk として実装されていて、利用することができます。NFS と同様に 1984 年に発表されたファイル共有のプロトコルで、最初は AppleTalk 上で動作するように開発されました。現在は、TCP/IP 上で動作するようになっています。最近になって、Unix や Linux と同様のファイルパーミッションに対応しました。主に、Mac が参加するネットワークで、Mac OS に独特のファイル属性などを扱う必要がある場合に利用されます。
- WebDAV
  HTTP プロトコル上で実現されるファイル共有です。HTTP 上で実現するためプラットフォーム依存がないのが特徴です。

ほとんどの NAS では、これらのプロトコルを使うことができますが、CIFS, AFS, WebDAV は更新型データの管理には向いていません。そのため、更新型データ管理には NFS がもっともよく使われます。

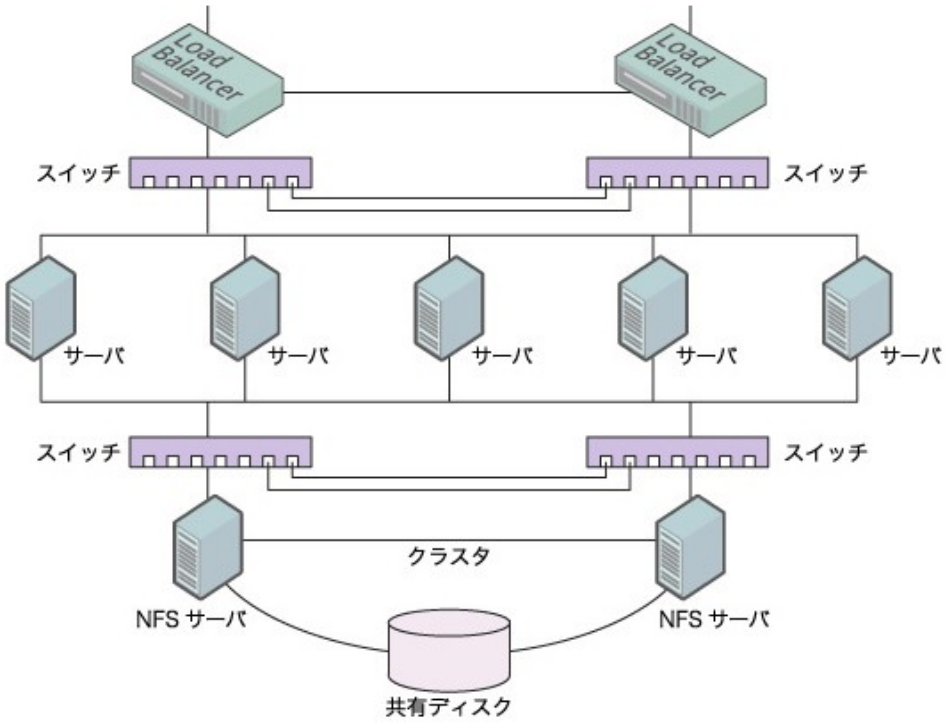
4.4.1　NFS によるデータ管理のモデル

NFS には、次のような特徴があり、ロードシェアリングによってシステムを冗長化する場合によく使われます。

- 汎用性
  Linux では NFS は標準的にサポートされています。また、多くの NAS が NFS に対応しています。
- スケーラビリティ
  サーバの性能が許せば、ファイルを共有することのできるホスト数には上限がありません。そのため、大規模なファイル共有システムにまで適用することができます。
- パフォーマンス
  Linux のバッファキャッシュを有効に利用できるキャッシュメカニズムを持っています。そのため、ファイルの読み込み時には高いパフォーマンスを発揮することができます。
- 管理性
  Linux では、いくつかの用意されたサービスを起動するだけで簡単に利用することができます。また、設定も容易です。

図 4-13 は、NFS サーバを使ったロードバランシングのシステム構成例です。

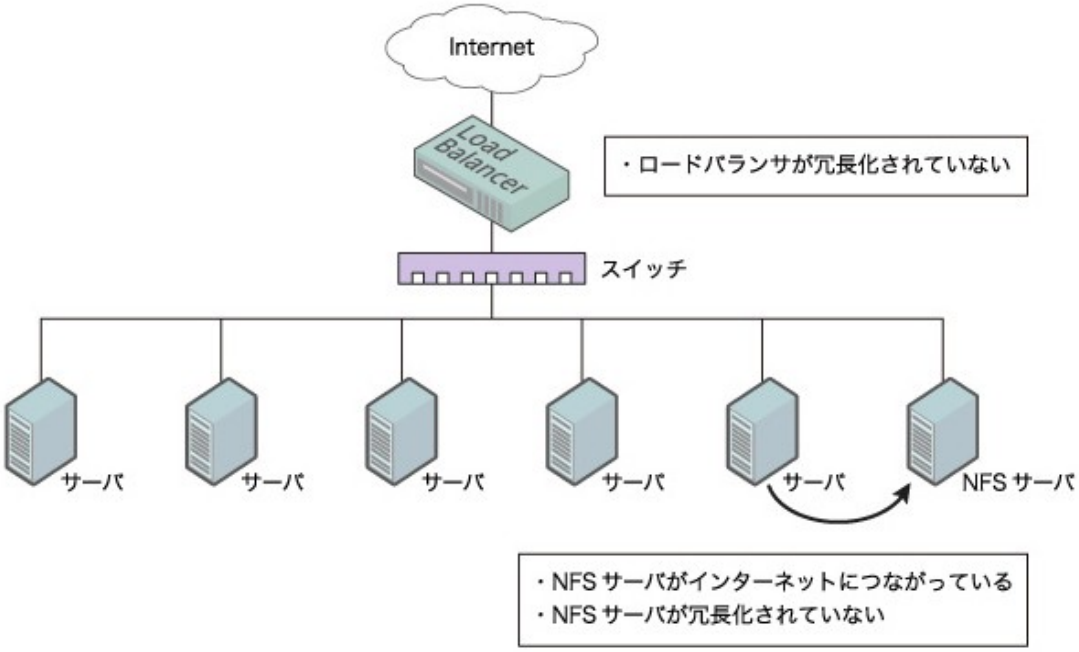
図 4-13:NFS によるファイル共有のイメージ



次のような点に注意してシステム構成を考える必要があります。

- ネットワークのパフォーマンス確保
  NFS サーバとサーバの間でデータ転送を高速に行えるようにするため、ファイル共有専用のネットワークを作り、ファイルを共有するのが一般的です。
- セキュリティ
  NFS サーバでは単純なホストベースのアクセス認証しかサポートしていませんので、インターネットから直接接続できるようなネットワークには配置しません。
- 冗長性
  NFS サーバやスイッチの冗長性にも十分に配慮する必要があります。この例では、アクティブ・スタンバイクラスタとして構成しています。図 4-14 のようにシステムを構成すると、NFS サーバやスイッチングハブが単独障害点となってしまい、フォールトトレランスが実現できません。

図 4-14:NFS によるファイル共有の悪いイメージ



4.4.2　NFS の注意点

NFS は決して万能ではありません。特に次の 2 つの特性については、十分に考慮してシステムを作る必要があります。

- ファイルキャッシュ
  NFS は、優秀なファイルキャッシュのメカニズムを持っていますが、サーバ間でキャッシュの一貫性を保つ機能をサポートしていません。
- ファイルロック
  NFS は、ファイルロックの機能を限定的にしかサポートしていません。そのため、次のような用途での利用は避けるべきです。
  - ファイルのロックを頻繁に利用するアプリケーション（データベースなど）
  - 複数のサーバから、同時に同じファイルを更新する処理

この 2 つの特性が原因で、次のような用途には使用できないことが知られています。

- データベースファイルの配置（共有はしなくても不向き）
- ロックが必要なファイルの共有
- 同時に複数のサーバから同じファイルを更新するシステム
- データの書き込み中に、ファイルを読み出す可能性の高いシステム

なお、この最後の項目は、ファイルの更新方法には十分注意する必要があることを示しています。

次のような手順でファイルを更新することで、この問題を回避することができます。

- 同じ NFS ファイルシステム上に元のファイルのコピーを作成する
- コピーを修正する
- 修正したファイルを適切なファイル名にリネームする

4.4.3　Linux での実装

Linux は、NFS サーバとしても NFS クライアントとしても利用できます。

サーバの実装

Linux を NFS サーバとして構成する場合には、次のような設定を行う必要があります。

- 関連サービスを起動する
- 共有するファイルを設定する
- アクセス制御の設定を行う

NFS のサーバとして動作する場合には、少なくとも nfsd、portmap の 2 つのサービスを起動する必要があります。また、NFS 上でロックを使う場合には、lockd も起動する必要があります。portmap は rpcbind、lockd は nfslock などの名称の場合があります。

▼*NFS 関連サービスの起動*

```
# service nfs start
NFS サービスを起動中:                                    [  OK  ]
NFS クォータを起動中:                                    [  OK  ]
NFS デーモンを起動中:                                    [  OK  ]
NFS mountd を起動中:                                     [  OK  ]
# service nfslock start
NFS statd を起動中:                                      [  OK  ]
# service portmap start
rpcbind を起動中:                                        [  OK  ]
```

NFS で共有するファイルシステムの設定は、/etc/exports で行います。次は、その設定例です。

▼*/etc/exports の設定例*

```
/home/safe          192.168.1.70(sync,rw)
/media/cdrom1        192.168.2.181(sync,rw)
```

公開するディレクトリに対して、アクセスを許可するクライアントと共有の権限を設定します。/etc/exports に設定をしたら、つぎのように exportfs を実行することでカーネルの管理テーブルに反映することができます。

▼*ディレクトリの公開*

```
# exportfs -a
```

また、アクセスするクライアントの許可は、/etc/exports 以外にも、TCP_WRAPPER などのアクセス制御で制限される場合があります。その場合には、次の例のようにクライアントからのアクセスを許可します。

▼*アクセス制御設定（/etc/hosts.deny）*

```
mountd: ALL
```

▼*アクセス制御設定（/etc/hosts.allow）*

```
mountd: 192.168.1.70 192.168.2.181
```

クライアントの実装

Linux を NFS クライアントとして構成する場合には、次のような設定を行う必要があります。

- 関連サービスを起動する
- ファイルシステムのマウント
- netfs サービスの起動

NFS クライアント側でも、portmap サービスを起動する必要があります。また、NFS 上でロックを使う場合には、lockd も起動する必要があります。起動方法は、サーバと同様です。

NFS サーバが公開しているファイルシステムをマウントする設定は、/etc/fstab で行います。次はその設定例です。

▼*NFS 共有の設定（/etc/fstab）*

```
nfsserver:/media/cdrom     /media/cdrom       nfs     defaults    0    0
```

最初の 1 カラム目は NFS サーバ名と公開しているファイルシステムのパスです。2 番目のカラムは、自サーバのマウントを行うディレクトリで、3 つめのカラムは nfs でマウントすることを示しています。4 番目のカラムは、マウント時のオプションの設定です。ここでは、defaults と設定していて、特別な設定を行っていませんが、必要に応じてオプションを指定することができます。設定ができたら、netfs サービスを起動することで、ファイルシステムがマウントされます。

▼*netfs サービスの起動*

```
# service netfs start
その他のファイルシステムをマウント中:                          [  OK  ]
```

4.5　SAN とクラスタファイルシステム

NFS は、更新型データを簡単に管理することができます。しかし、キャッシュやロックなどの機能や利用方法に制限があり、データベースなどのデータを管理することはできません。こうした場合には、**SAN**（Storage Area Network）を利用することができます。

4.5.1　SAN の概要

SAN は、ハードディスクやテープなどの記憶装置を、高速なネットワークで接続してシステム化することで、大容量の記憶装置を作成する技術です。ファイバチャネルと iSCSI の 2 つのネットワーク技術が使われています。

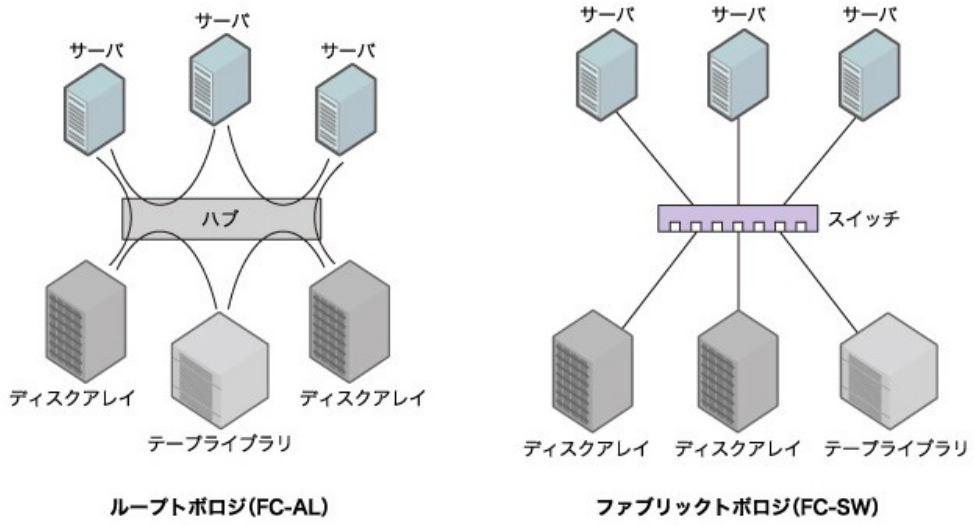
ファイバチャネル

ファイバチャネル（Fibre Channel）は、大容量の記憶容量を実現するために、Hewlett-Packard，IBM、SUN Microsystems などが主導して作った FCSI（Fibre Channel Systems Initiative）という業界団体が規格化したネットワーク技術です。現在は、ANSI（米国規格協会）の X3T11 分科会が標準化を行っています。長距離伝送が可能で、なおかつ最大 8Gbit/s という高速通信を行うことができるのが特徴です。最近では、さらに高速な 16Gbit/s の規格化が進められています。

ファイバチャネルは、Ethernet とはまったく異なる独自のネットワーク規格です。そのため、ネットワークの構成（トポロジ）も独特です。次の 3 つのトポロジをサポートしています(図 4-15)。

- ポイント・ツー・ポイント（P2P）
- ループトポロジ（FC-AL）
- ファブリックトポロジ（FC-SW）

図 4-15:FC-AL, FC-SW のイメージ



どのトポロジを使用する場合でも、ファイバチャネルを利用するにはコンピュータ側に HBA（Host Bus Adapter）と専用のドライバソフトウェアを搭載する必要があります。

iSCSI

iSCSI(Internet Small Computer System Interface)は、ローカルのディスクドライブなどの接続に主に利用される SCSI プロトコルを拡張して、TCP/IP ネットワーク上で利用することができるようにした規格です。そのため、一般的に使われる Ethernet を使って SAN を実現することができます。最近では、Ethernet 技術も高速化が進んでいることから、ファイバチャネルと同様に利用されています。

iSCSI は、TCP/IP 上で利用するため通信速度は残念ながらファイバチャネルに劣ります。しかし、ファイバチャネルでは専用の HBA が必要なのに対して、iSCSI では通常の Ethernet を使いますので、通常の Ethernet 用の LAN ポートを利用することができ、導入コストも低く抑えることができます。

SCSI プロトコルでは、機器に命令を発する装置を**イニシエータ**、命令を受け取る装置を**ターゲット**と呼びます。この呼び方は iSCSI でも同様です。iSCSI に対応したストレージ専用機は、多くのベンダから市販されています。

4.5.2　Linuxでの実装

Linux ではファイバチャネルも iSCSI も利用することができます。ファイバチャネルを利用するためには、HBA に合わせて専用のドライバをインストールする必要があります。また、利用法は HBA によって異なります。

また、Linux は iSCSI のイニシエータとしてもターゲットとしても利用することができます。それを実現するためのドライバやユーティリティが次のようなプロジェクトで開発されています。

- イニシエータ
  - UNH-iSCSI(http://unh-iscsi.sourceforge.net/)
  - Linux-iSCSI（http://linux-iscsi.sourceforge.net/）
  - Open-iSCSI（http://www.open-iscsi.org/）
- ターゲット
  - UNH-iSCSI(http://unh-iscsi.sourceforge.net/)
  - iSCSI Enterprise Target（http://iscsitarget.sourceforge.net/）
  - Linux SCSI target framework（http://stgt.berlios.de/）

iSCSI では、ターゲットやイニシエータに名前を付けて管理します。これを iSCSI 名と呼びます。iSCSI 名は、インターネット上で一意でなければなりません。iSCSI 名の命名規則として、IQN 形式と EUI 形式の 2 つがあります。

- IQN(iSCSI Qualified Name)形式
  各組織が所有しているドメイン名を使う形式
- EUI(IEEE EUI-64 format)形式

EUI-64 フォーマットの識別子を iSCSI 名として使用する形式です。

EUI-64 は IEEE によって主にハードウェアベンダーに割り当てられるものです。そのため、IQN 形式の名称を使うのが一般的です。次は、IQN 形式の名称の例です。

```
iqn.2001-04.com.example:storage:diskarrays-sn-a8675309
↑    ↑       ↑                 ↑
(1)  (2)     (3)               (4)
```

(1) 接頭辞(IQN 形式の場合は「iqn.」固定
(2) ドメインの取得年月
(3) ドメイン名(逆順)
(4) 任意の識別子(ドメイン内で一意にする必要がある)

例えば、Open-iSCSI では、次のような手順で iSCSI ディスクを利用します。

- 該当ホストをイニシエータとして設定します。
- iscsi サービスを起動します。
- iSCSI ターゲットにログインします。

これらの処理が成功すると、iSCSI ディスクを/dev/sda のような通常のディスクデバイスとして利用できるようになります。

▼*Open-iSCSI でのイニシエータの設定例（/etc/iscsi/initiatorname.iscsi）*

```
InitiatorName=iqn.2000-03.jp.designet:initiator.test
```

▼*Open-iSCSI での iSCSI ディスクの利用例*

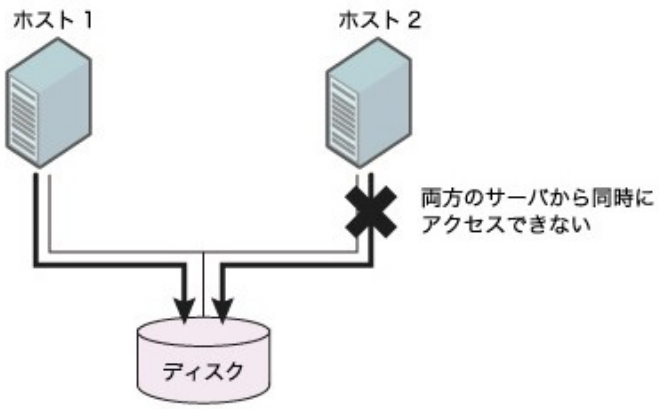
```
# service iscsi start                                          ← iscsi サービスを起動
iscsid dead but pid file exists
Turning off network shutdown. Starting iSCSI daemon:       [  OK  ]
                                                           [  OK  ]
Setting up iSCSI targets: iscsiadm: No records found!
                                                           [  OK  ]
# iscsiadm -m discovery -t sendtargets -p 192.168.2.235:3260      ← ターゲットを検索
192.168.2.235:3260,1 iqn.2000-03.jp.designet:storage.test
# iscsiadm -m node -T iqn.2000-03.jp.designet:storage.test -p 192.168.2.235:3260 -l
                                                                              ←ログイン
Logging in to [iface: default, target: iqn.2000-03.jp.designet:storage.test, portal:
192.168.2.235,3260]
Login to [iface: default, target: iqn.2000-03.jp.designet:storage.test, portal:
192.168.2.235,3260]: successful

# dmesg                                                        ← カーネルメッセージ
       :
scsi8 : iSCSI Initiator over TCP/IP
  Vendor: IET       Model: Controller        Rev: 0001
```

```
  Type:   RAID                                ANSI SCSI revision: 05
scsi 8:0:0:0: Attached scsi generic sg0 type 12
  Vendor: IET       Model: VIRTUAL-DISK      Rev: 0001
  Type:   Direct-Access                       ANSI SCSI revision: 05
SCSI device sda: 2104515 512-byte hdwr sectors (1078 MB)
sda: Write Protect is off
sda: Mode Sense: 79 00 00 08
SCSI device sda: drive cache: write back
SCSI device sda: 2104515 512-byte hdwr sectors (1078 MB)
sda: Write Protect is off
sda: Mode Sense: 79 00 00 08
SCSI device sda: drive cache: write back
 sda: unknown partition table
sd 8:0:0:1: Attached scsi disk sda                          ← /dev/sda として認識した
sd 8:0:0:1: Attached scsi generic sg1 type 0
```

4.5.3　アクティブ・スタンバイクラスタでの構成

ファイバチャネルや iSCSI などを使うことで、複数のサーバから同じディスクを参照することができるようになります。図 4-16 は、こうしたアクティブ・スタンバイのシステム構成の例です。

図 4-16:アクティブ・スタンバイシステムの構成例



ホスト 1 とホスト 2 の両方から、同じディスクを参照しています。しかし、両方のホストから同時にディスクへのアクセスを行うことはできません。

図 4-17 は、Linux の一般的なファイルシステムのデータの扱いを示したものです。Linux のファイルシステムでは、アプリケーションがファイルを読み込むときには、ファイルを一旦メモリに読み込みます。このときに、ファイルのデータだけでなく、ファイル名、更新時間、データの物理的な保存場所などの属性情報も一緒にメモリ上に読み込まれます。そして、一旦読み込んだファイルのデータと情報を、高速化のために可能な限りメモリ上にキャッシュします。

図 4-17: ファイルシステムとキャッシュ

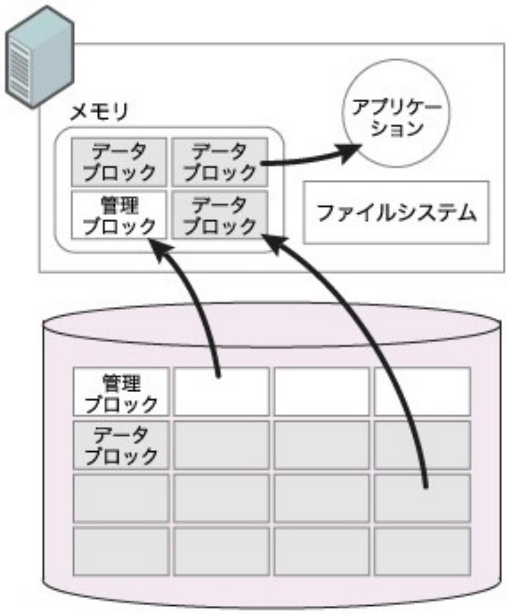
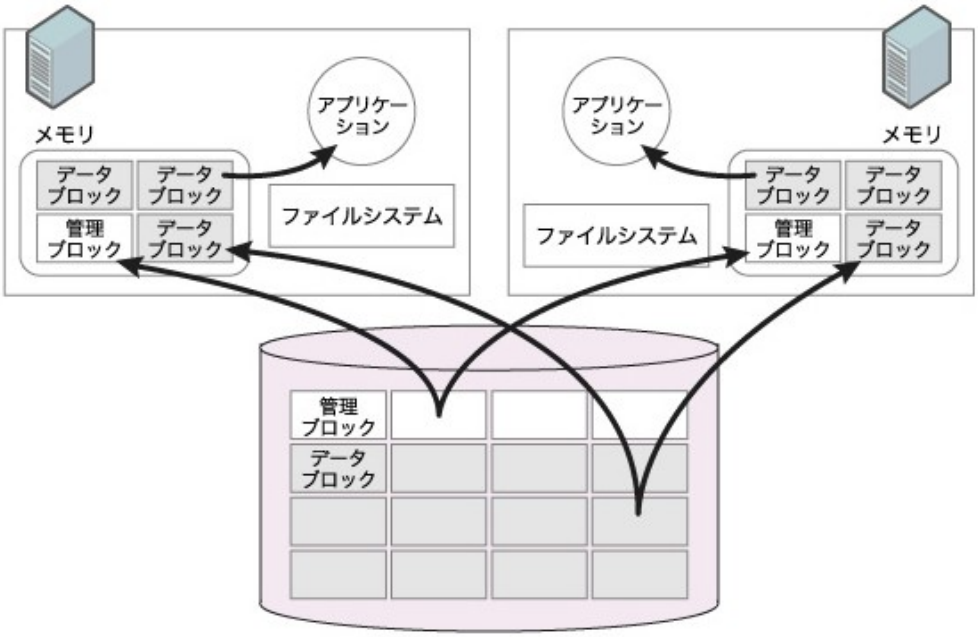


図 4-18 は、このような状態のファイルシステムを複数のホストからマウントしようとした場合のイメージです。

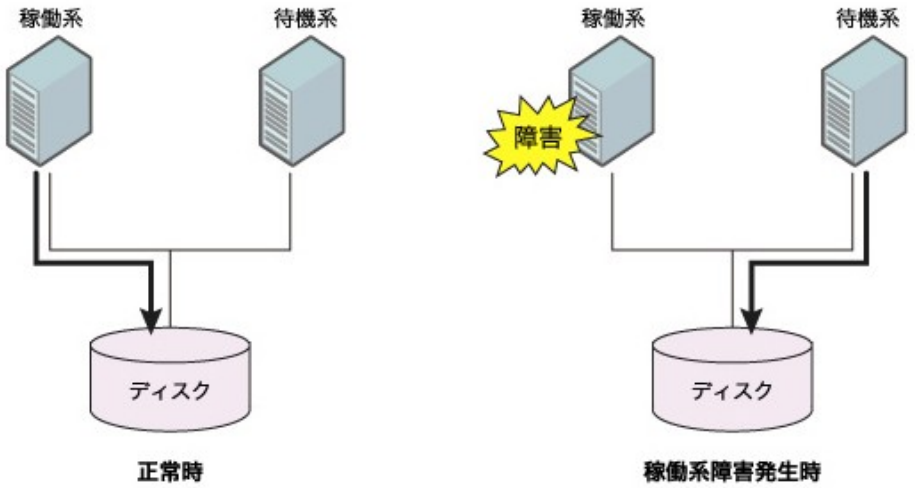
図 4-18:複数のサーバからファイルシステムを同時にマウントする



アプリケーションが参照するデータは、物理的なディスク上ではなく各ホストのメモリ上のデータとなります。そのため、どちらかのマシンでデータを更新しても、もう一方のホストはそれを検知することができません。その結果、ホストによって見る情報が変わってしまうことになります。特に、相手のホストによってファイルの属性情報が更新されると、ハードディスク上の物理的な位置に関する情報が信頼できなくなります。このようなことが両方のホストから行われると、ファイルシステム上のデータはまったく一貫性がなくなり、事実上破壊されたのと同じ状態になってしまいます。したがって、一般的なファイルシステムを使って 2 つのホストからデータを共有することは事実上できません。

このような制約のため、共有ディスクを使ってファイルシステムを共有する場合には、2 つのホストから同時にデータを利用するのではなく、図 4-19 のようにクラスタの稼働系となっているホストでのみデータを利用します。

図 4-19:共有ディスク利用時のファイルシステム共有

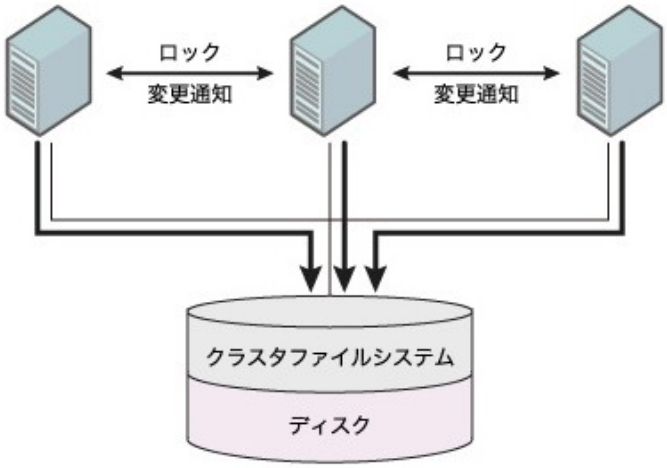


間違って両方のホストからファイルシステムをマウントすると、ファイルシステムが破壊されてしまう恐れがありますので、十分に注意して利用する必要があります。

4.5.4　ロードシェアリングでの構成

複数のホストからファイルシステムをマウントすることができないと、NFS のようにロードシェアリングしているサーバからデータを参照する用途では利用することができません。そのため、ロードシェアリングを行う場合には、**クラスタファイルシステム**を使います(図 4-20)。

図 4-20:クラスタファイルシステムの利用



クラスタファイルシステムは、ネットワークに参加する各ホストが、ファイルシステムのレベルで変更の情報を互いに通知して、ネットワークシステム全体で一貫性が保つことができます。

Linuxでの実装

Linuxでは、次の2つのクラスタファイルシステムを利用することができます。

- GFS(Global File System)
  RedHatによって開発されているクラスタファイルシステムです。Red Hat Cluster Suiteの一部として、レッドハット社による商用サポートの対象となっているのが特徴です。GFSから派生したGFS2は、Linuxカーネルのバージョン2.6.19以降に統合されています。GFSは、Red Hat Clusterというクラスタソフトウェアとともに使う必要があります。

- OCFS(Oracle Cluster File System)
  Oracleによって開発されているクラスタファイルシステムです。元々Oracleのデータベース製品からの利用を主眼に開発されていましたが、バージョン2(OCFS2)ではPOSIXに準拠するなど、汎用的な利用が可能となっています。またOCFS2は、Linuxカーネルのバージョン2.6.16以降に統合されています。OCFSは、クラスタソフトウェアとは切り離されていますので、heartbeatなどとともに利用することができます。

クラスタファイルシステムでは、同じディスクを共有するサーバ同士で、キャッシュやロックの情報を交換します。そのため、どのサーバとディスクを共有するかという設定が必要です。例えば、OCFSを使う場合には、次のような設定を行う必要があります。

- ファイルシステムを利用するホストの設定を行う
- 事前にサーバ間の通信条件を設定する
- キャッシュやロックの情報を管理するサービス(o2cb)を起動する
- どれか1つのサーバから、ocfs2ファイルシステムを作成する

利用ホストの設定

クラスタファイルシステムを利用するホストの設定は、/etc/ocfs2/cluster.confで行います。
次は、その設定例です。各ホスト毎にIPアドレス、ポート番号、ホスト名などを設定します。

▼*OCFSのホスト設定の例（/etc/ocfs2/cluster.conf）*

```
cluster:
        node_count = 2
        name = ocfs2

node:
        ip_port = 7777
        ip_address = 10.0.0.100
```

```
        number = 0
        name = host1
        cluster = ocfs2

node:
        ip_port = 7777
        ip_address = 10.0.0.101
        number = 1
        name = host2
        cluster = ocfs2
```

通信条件の設定

o2cb サービスを起動すると、次のように、最初に通信条件の入力を求められます。[]内にデフォルト値が表示されますので、特に希望する設定がなければ Enter を入力することで、デフォルト値が使われます。

▼*通信条件の設定とサービスの起動*

```
# service 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) [n]: y
Cluster stack backing O2CB [o2cb]:
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]:
Writing O2CB configuration: OK
Loading filesystem "configfs": OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading filesystem "ocfs2_dlmfs": OK
Creating directory '/dlm': OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Starting O2CB cluster ocfs2: OK
```

ファイルシステムの作成

o2cb サービスを開始したら、ファイルシステムを作成することができます。次は、その作成例です。

▼*OCFS ファイルシステムの作成*

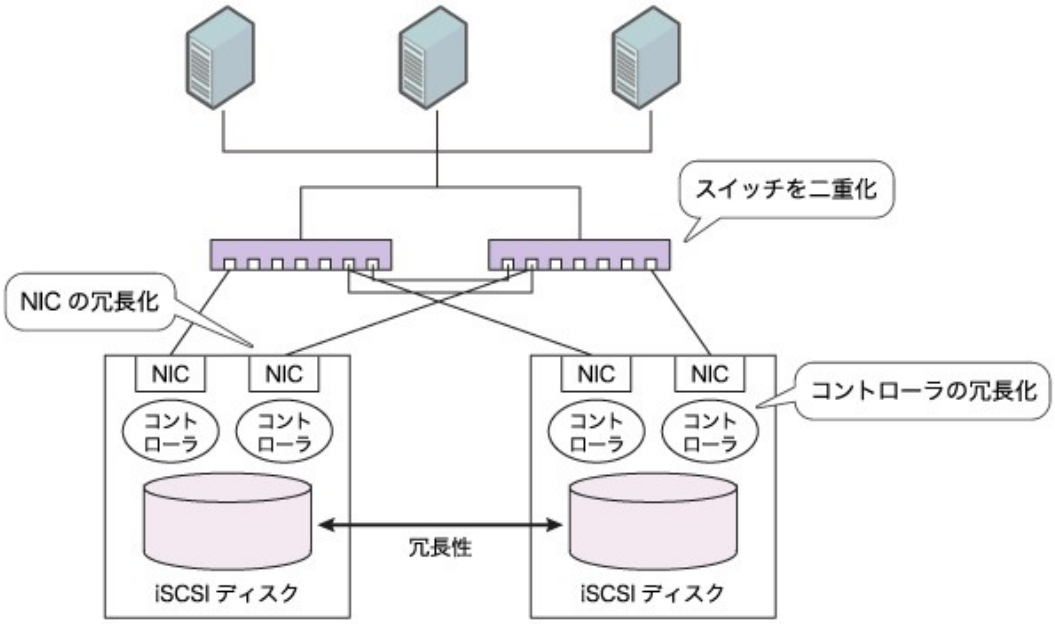
```
host1# mkfs -t ocfs2 -N 2 -L ocfs2_fs0 /dev/sdb
mkfs.ocfs2 1.4.2
Cluster stack: classic o2cb
Filesystem label=ocfs2_fd0
Block size=4096 (bits=12)
Cluster size=4096 (bits=12)
Volume size=1077510144 (263064 clusters) (263064 blocks)
9 cluster groups (tail covers 5016 clusters, rest cover 32256 clusters)
Journal size=67108864
Initial number of node slots: 2
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 1 block(s)
Formatting Journals: done
Formatting slot map: done
Writing lost+found: done
mkfs.ocfs2 successful
```

-N オプションではノード数、-L オプションではファイルシステムラベルを設定しています。

共有ディスク利用の注意点

共有ディスクを利用する場合には、ディスクやスイッチが単独障害点にならないようなシステム構成を考える必要があります(図 4-21)。最近のハードディスクの中には、ディスク装置間で冗長性を取ることができるものが増えています。

図 4-21:共有ディスク利用時の冗長化構成の例



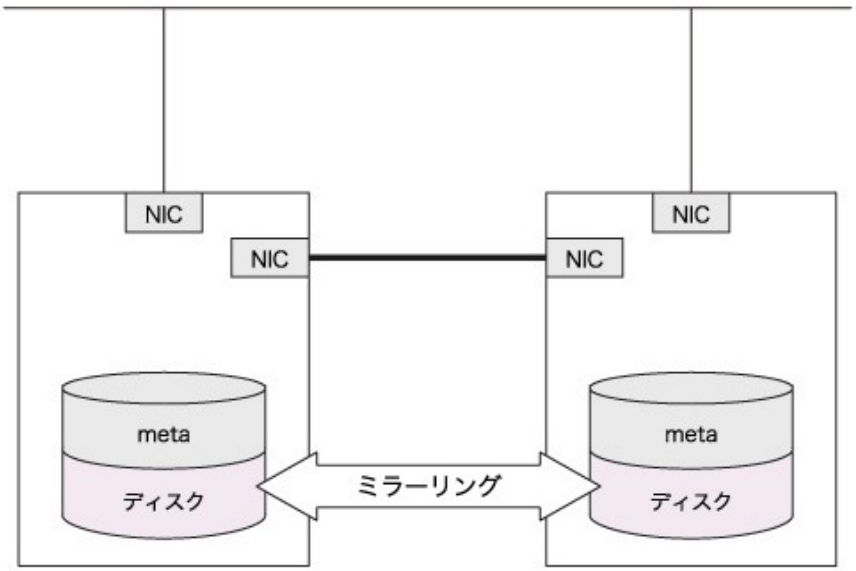
ディスクのデータを保護するためには、RAID などをサポートした製品を利用しますが、それだけでは解決になりません。特に iSCSI やファイバチャネルのディスクは、ネットワークを利用して通信を行う複雑なディスク装置で、故障すると回復にもかなり時間がかかります。そのため、ディスク装置自体の冗長性も必ず考慮しなければなりません。

4.6 ネットワークミラーリング

ファイバチャネルや iSCSI のディスクを使って共有ディスクを作る場合には、ディスク装置の冗長性まで考慮する必要があります。そのため、冗長構成を取ることのできる特殊なディスク装置を使う必要があります。こうしたディスク装置は大変高価ですので、なかなか導入することが難しいのが実状です。

こうした状況を改善するために、ネットワーク上でファイルシステムのデータをミラーリングするネットワークディスクミラーリングの技術が使われています(図 4-22)。

図 4-22:ネットワークミラーリングのイメージ



Linux の多くの実装では、ネットワークミラーリングは特別なデバイスドライバの機能として提供されていて、ファイルシステムとハードディスクドライバの間で動作します。ファイルの書き込みが発生すると、ハードディスクへの書き込みを行うとともに、ネットワークを介してリモートのディスクへの書き込みも行います。

ネットワークミラーリングの特徴

ネットワークミラーリングは、共有ディスクと比較して次のような特徴があります。

- 経済性
  共有ディスクのような特殊な機器が不要で、安価に導入することができる。
- データの冗長性
  共有ディスクではデータが 1 ヶ所にしか保管されないが、ネットワークミラーリングでは 2 つのコンピュータのハードディスクにデータが保管される。

- 処理速度
  ネットワークを介して同期を取るため、通常のディスクアクセスに比べて約 1 割程度速度が低下する。

ネットワークミラーリングを利用する場合には、こうした特性を十分に考慮する必要があります。

4.6.1　Linux での実装

Linux では、DRBD(Distributed Redundant Block Device)というネットワークミラーリングソフトウェアを利用することができます。DRBD は、Philipp Reisner 氏が開発したネットワークミラーリング用のソフトウェアで、GNU Public License に基づき配布されています。いくつかの商用 Linux ディストリビューションで採用されている他、日本でも Linbit 社の代理店により有償サポートを受けることもできます。

DRBD は、ネットワークを経由して 2 台のサーバに接続されたディスクの間でミラーリングを行うことで、リモートサーバへのデータのバックアップや、クラスタシステムでのデータ共有を行うことができます。アクティブ・スタンバイクラスタで利用する場合には、どちらか片方のホストからしかディスクを利用できないように制限することもできます。また、クラスタファイルシステムと組み合わせて使って、同時に 2 つのホストからデータを読み書きすることもできます。DRBD は、ハードディスクドライバとカーネルの間で働くモジュールと、それを管理するための管理コマンドから構成されています。

4.6.2　DRBD の仕組み

DRBD によるネットワークミラーリングは、図 4-23 のように 2 台のサーバで構成します。DRBD でディスクを共有するサーバをピアノードと呼びます。また DRBD は、1 つのカーネルモジュールと 3 つのカーネルスレッドから成り立っています。

図 4-23:DRBD のネットワークミラーリングの仕組み

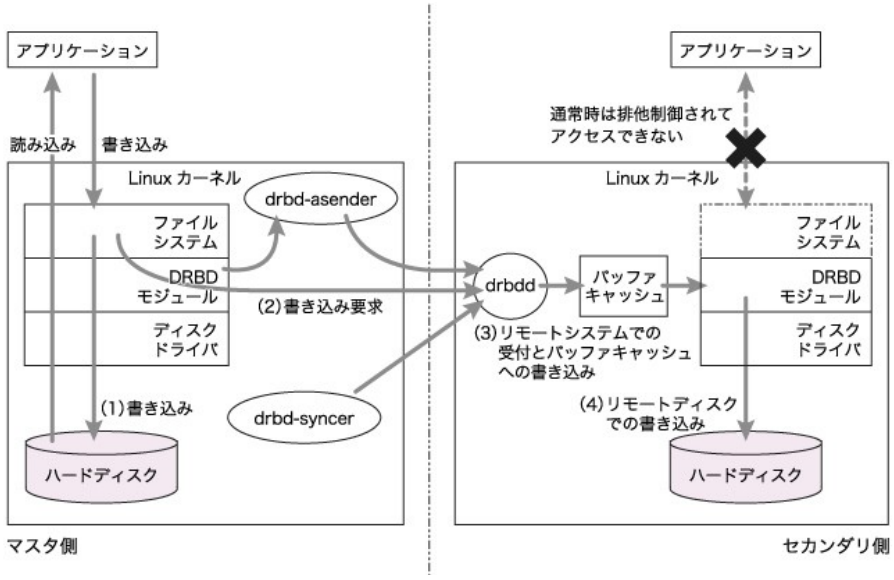


図 4-23 では、マスタ側とセカンダリ側に分けて構成が描かれていますが、この区分けは動作上のモードでしかなく、実際の構成上は完全に対称です。カーネルモジュールとカーネルスレッドは、それぞれ次のような役割を持っています。

- DRBD モジュール
  ブロックデバイスの機能を提供します。読み込み要求があった場合には、ローカルハードディスクからデータの読み出しを行います。書き込み要求があった場合には、ローカルディスクとピアノードへ書き込み要求を出します。
- drbd-asender
  ピアノードと非同期に通信を行うカーネルスレッドです。DRBD モジュールがピアノードに対して大量の書き込み要求を出した場合に、データの送信を管理します。
- drbd-syncer
  ディスクの同期処理を行うためのカーネルスレッドです。
- drbdd
  ピアノードからの書き込み要求を受け付けるカーネルスレッドです。書き込むデータをバッファキャッシュを介して DRBD モジュールに渡します。

4.6.3　ディスクの同期処理

DRBD では、用途に合わせて 3 つのデータ同期モデル（プロトコル）をサポートしています。

- プロトコル A
  ローカルディスクへの書き込み(1)とリモートディスクへの書き込み要求(2)が完了したときに、書き込み完了とします。
- プロトコル B
  ローカルディスクへの書き込み(1)が完了し、リモートディスクへの書き込み要求がバッファキャッシュに書き込まれた時点(3)で書き込み完了とします。
- プロトコル C
  ローカルディスクへの書き込み(1)とリモートディスクへの書き込み(4)が完了した時点で書き込み完了とします。

プロトコル A がもっとも速く動作しますが、ピアノードへリクエストが届くかどうかも保証されていないため、安全性が低くなります。逆にプロトコル C は、リモートディスクの書き込みまでを確認しますので、もっとも安全ですが、速度は遅くなります。プロトコル B はその中間で、実際にピアノードのディスクへの書き込みまでを保証しませんが、要求がピアノードに届いたことを確認するという点で、比較的速度も速く安全な構成です。

4.6.4　ディスクの同期方法

また、DRBD ではディスクの同期を速やかに行うために、**部分同期**と**フル同期**をサポートしています。ピアノードとの最初の接続や、長期間に渡ってピアノードへのデータの書き込みが行えなかった場合には、フル同期が使われます。しかし、一時的な接続断や再起動などで、ピアノードへのディスク書

き込みが行えなかった場合には部分同期が利用され、変更部分だけを同期します。そのため、比較的高速に処理を行うことができます。なお、DRBD では、部分同期を行うための変更情報を**メタデータ領域**と呼ばれるディスク上の管理領域に保管します。メタデータは、ファイルシステムのデータと同じパーティションに置くこともできますし、別に管理することもできます。メタ領域は、最低でも 1 つのリソースあたり 128Mbyte 必要です。管理するデータ領域が大きい場合には、より大きなサイズが必要になります。サイズは、次のように計算します。

$$\text{メタ領域サイズ}\ (Mb) = \frac{\text{データサイズ}\ (Mb)}{32768} + 1$$

4.6.5　Linux での実装

DRBD を利用してファイルシステムを共有する場合には、次のような設定を行う必要があります。

- 共有用のパーティションを用意する（両ノード）
- ピアノードの情報や通信パラメータなどを設定する（両ノード）
- メタデータを作成する（プライマリノード）
- DRBD サービスを起動する（両ノード）
- 最初のデータ同期をとる（プライマリノード）
- ファイルシステムを作成する（プライマリノード）

ピアノードの情報と通信パラメータの設定

DRBD で管理するリソース、ディスクを共有する相手のノード、通信パラメータなどを/etc/drbd.conf に設定します。このファイルは、両方のノードで同じでなければなりません。

▼*DRBD の設定ファイル（/etc/drbd.conf）の例*

```
global {
    usage-count no;
}

common {
    syncer {
        rate 500M;                                          ← ディスク同期処理の速度設定
    }
}

resource r0 {
    protocol C;                                             ← 同期プロトコルの設定
    handlers {
        pri-on-incon-degr "echo '!DRBD! pri on incon-degr' | wall; sleep 60; halt -f";
    }

    startup  {                                              ← 起動時の通信パラメタ
```

```
        wfc-timeout 120;
        degr-wfc-timeout 120;
    }

    disk {                                                  ← ディスク障害への対応方法
        on-io-error detach;
    }

    on host1 {                                              ← host1の定義
        device     /dev/drbd                                ← DRBDデバイス名
        disk       /dev/sdb2;                               ← データ用デバイス
        address    10.0.0.100:7788;                         ← IPアドレスとポート番号
        meta-disk  /dev/sdb1[0];                            ← メタデータのデバイスと配置
    }

    on host2 {                                              ← host2の定義
        device     /dev/drbd0;
        disk       /dev/sdb2;
        address    10.0.0.101:7788
        meta-disk  /dev/sdb1[0];
    }
}
```

メタデータの作成

/etc/drbd.confの設定ができたら、プライマリノードでメタデータの作成を行います。次は、その実行例です。

▼*メタデータの作成（プライマリノード）*

```
host1# drbdadm create-md r0
Writing meta data...
initializing activity log
NOT initialized bitmap
```

サービスの起動（プライマリ）

次に、プライマリノードで drbd サービスを起動します。この時点では、次の例のように、ピアノードの起動待ちとなります。

▼*サービスの起動（プライマリノード）*

```
host1# service drbd start
Starting DRBD resources: [ d(r0) s(r0) n(r0) ].........
***************************************************************
 DRBD's startup script waits for the peer node(s) to appear.
 - In case this node was already a degraded cluster before the
```

```
   reboot the timeout is 120 seconds. [degr-wfc-timeout]
 - If the peer was available before the reboot the timeout will
   expire after 120 seconds. [wfc-timeout]       ← degr-wfc-timeout の値
   (These values are for resource 'r0'; 0 sec -> wait forever)
 To abort waiting enter 'yes' [ 119]:                          ←ピアノードの起動待ち
```

サービスの起動（セカンダリ）

プライマリノードの次にセカンダリノードで drbd サービスを起動します。次は、その実行例です。

▼*サービスの起動と同期（セカンダリノード）*

```
host2# service drbd start
Starting DRBD resources:    [ d(r0) s(r0) n(r0) ].

```

各ノードでサービスを起動した段階では、両方のノードともセカンダリとなります。またこの時点ではデータを同期していませんので、データの一貫性が取れていない状態になります。

データの強制同期

プライマリノードの状態をセカンダリから、プライマリに昇格します。この時、プライマリノード側のデータを正としてセカンダリノード側に上書きすることで、強制的にデータの一貫性を取ります。次は、データを強制同期し、プライマリノードを昇格するコマンドの実行例です。

▼*データの強制同期と同期状態の確認（プライマリノード）*

```
host1# drbdadm -- --overwrite-data-of-peer primary all
host1# service drbd status
drbd driver loaded OK; device status:
version: 8.3.2 (api:88/proto:86-90)
GIT-hash:    dd7985327f146f33b86d4bff5ca8c94234ce840e    build    by    mockbuild@v20z-x86-
64.home.local, 2009-08-29 14:02:24
m:res  cs          ro                 ds                     p  mounted  fstype
0:r0   SyncSource  Primary/Secondary  UpToDate/Inconsistent  C       ← 接続状態が
                                                                     SyncSource になる
...    sync'ed:    11.0%              (699316/779156)K               ← 同期の進捗状況
```

ファイルシステムの作成

データの同期が取れたら、ファイルシステムを作成することができます。次の例のように、/dev/drbd0 のようなデバイス名を指定して、ファイルシステムを作成することで、ネットワークミラーリングで共有可能なファイルシステムが作成されます。

▼*ファイルシステムの作成（プライマリノード）*

```
host1# mke2fs -j /dev/drbd0                ←ext3 ファイルシステムを作成
mke2fs 1.39 (29-May-2006)
       :
       :
host1# mkdir /data
host1# mount /dev/drbd0 /data              ←マウント
```

5章 データベースの冗長化

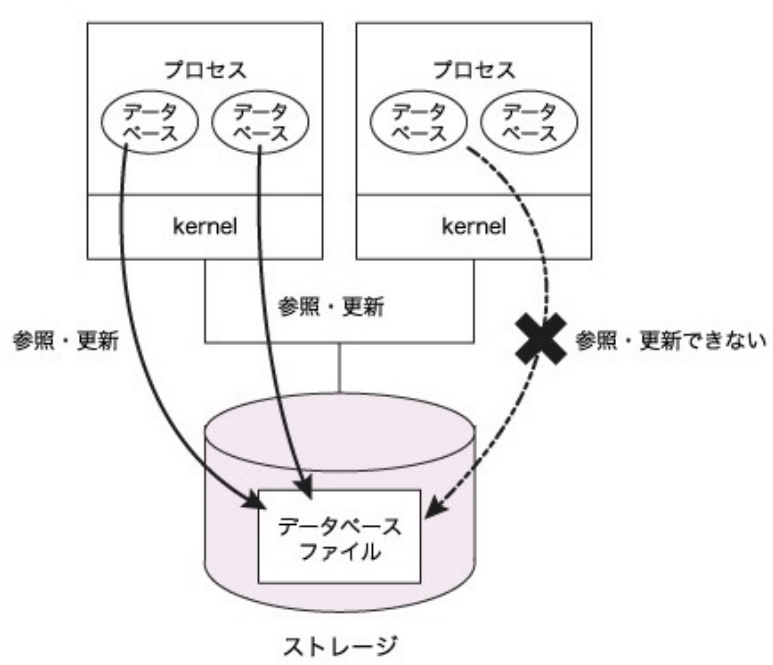
データベースでは、一般的に重要なデータを管理します。システムの障害や停止によってデータベースが利用できなくなると、多くのサービスが継続できなくなります。しかし、データベースが扱う情報は更新型の情報ですので、稼働率を向上するための仕組みを作ることは簡単ではありません。本章では、データベースを冗長化するための方法について学習します。

5.1 データベース冗長化の概要

データベースではデータの参照と更新の両方をサポートします。通常のデータベースでは、データは頻繁に更新されます。また、RDB ではデータベースの各テーブルや項目には相関関係があり、それを適切に維持するために、テーブルのロックやトランザクションなどの制御をサポートしています。

リクエストの受け付けや演算を高速に行うため、データベースは複数のプロセスで処理を行いますが、実際のデータファイルへの更新はこの各プロセスが協調して実施する必要があります。そのため、データベースは、共有メモリ、セマフォ、スレッド、ロックなどの Linux の機能をフル活用して実現されています。このようなプロセス間で高速に情報を共有するための機能は、今のところ 1 つのコンピュータ内でしか利用することができません。したがって、データベースの処理プロセスを複数のコンピュータに分散させ、1 つのデータベースファイルを共有して同時に参照や更新をするというモデルは容易に実現することができません(図 5-1)。

図 5-1:複数のサーバからデータベースを同時に参照・更新することはできない



こうした問題があるため、データベースの冗長化には今のところ決定的なよい方法はありません。次のような方法を用途によって使い分ける必要があります。

- アクティブ・スタンバイモデル
  2 台のサーバでデータベースシステムを構成しますが、通常は稼働系サーバでのみ処理を行

い、障害が発生したときに待機系サーバで処理を継続します。データは、共有ディスクやネットワークミラーリングなどのシステムレベルの仕組みを使って共有しますが、常に 1 つのサーバからしか参照されません。

- レプリケーションモデル
  データベースの機能として実現される冗長化の方法です。**レプリケーション**とは、複数のサーバや記憶装置のデータが同じ状態になるように、データベースに行われた演算や処理を複製することです。複数の記憶装置に同じデータを記録する**データレプリケーション**、同じ演算を異なる複数のサーバで行う**空間レプリケーション**などがありますが、システムの冗長化を行う場合には**空間レプリケーション**を利用します。
  また、レプリケーションには次の 2 つの方法があります。
  - 動的レプリケーション
    データベースに対して行われる更新要求をすべての複製に対して行います。
  - 静的レプリケーション
    更新処理をどれか 1 つのサーバで行い、処理結果だけを複製に伝えます。最初に更新処理を行うサーバを**マスタサーバ**と呼びます。マスタサーバが全システムに 1 つの構成を**シングルマスタモデル**あるいは**マスタスレーブモデル**と呼びます。シングルマスタモデルでは、更新処理は常にマスタサーバに対して行い、スレーブサーバでは参照処理のみを行うことができます。これに対して、システム内に複数のマスタサーバが存在する構成を**マルチマスタモデル**と呼びます。マルチマスタモデルでは、更新処理はどのマスタサーバに対しても行うことができます。

これらのどの仕組みを使っても冗長化を実現することができますが、表 5-1 のようにそれぞれ長所と短所がありますので、用途によって使い分ける必要があります。

▼*表 5-1：データベース冗長化方式の長所と短所*

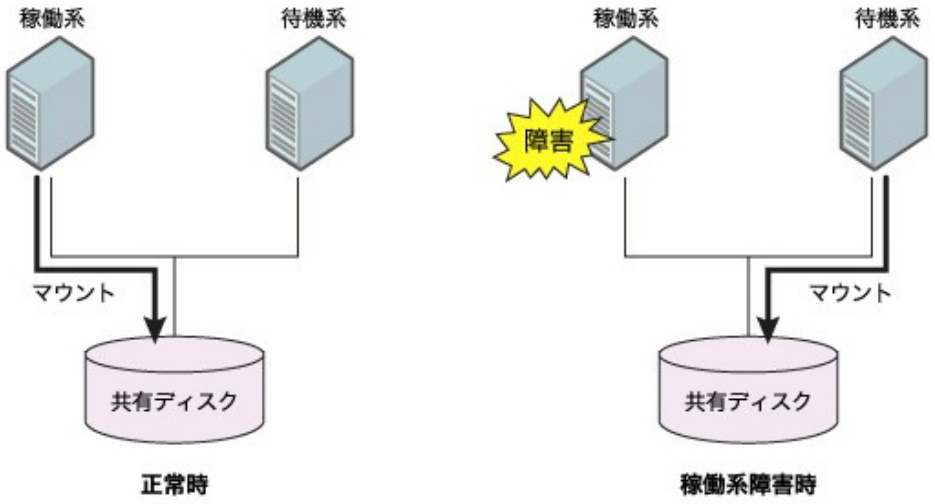
冗長化モデル	長所	短所
アクティブ・スタンバイ（共有ディスク）	• データを複製しないため、更新性能も参照性能も劣化しない • サーバの切り替えが自動的に行われる • アプリケーションは、冗長化を意識する必要がない	• 参照性能は 1 台の処理能力以上にはならない • 共有ディスクも冗長化する必要があり、導入コストが高い
アクティブ・スタンバイ（ネットワークミラー）	• システムでデータの複製を行うため、複製処理による性能劣化が少ない • 参照性能は劣化しない • 更新性能の劣化が最小である • サーバの切り替えが自動的に行われる • アプリケーションは、冗長化を意識する必要がない	• 参照性能は 1 台の処理能力以上にはならない

動的レプリケーション	• 1台の参照処理性能の劣化は最小である • 参照処理の負荷分散ができる • ロードバランシングの仕組みを一緒に実現することができる • システム全体で参照性能を向上することができる • アプリケーション側で更新処理と参照処理を区別して行う必要がない	• データ複製による処理の待ち合わせのため更新処理性能が大きく劣化する • アプリケーションは、動的更新に不向きな処理を行わないように設計しなければならない
シングルマスタ レプリケーション	• 更新処理性能の劣化が少ない • 参照処理を複数台で行うことができる • システム全体で参照性能を向上することができる	• アプリケーション側は、更新処理と参照処理を区別して行うよう設計しなければならない • 参照サーバへの複製に時間がかかりデータがリアルタイムに反映されない • マスタサーバの停止で更新ができなくなり、冗長性が低い
マルチマスタ レプリケーション	• 更新処理、参照処理を複数台で行うことができる • アプリケーション側で更新処理と参照処理を区別して行う必要がない	• 更新処理がすべての複製に反映されるまでに時間がかかる • 更新処理の伝達に時間がかかることを考慮したシステム設計が必要である • 処理が複雑なため、システムが不安定になりやすい

5.2　アクティブ・スタンバイ（共有ディスク）による冗長化

データベースの処理性能が 1 台のサーバの処理性能で十分な場合には、アクティブ・スタンバイによる冗長化を利用します。冗長化を考慮してシステム設定を行う必要がありますが、データベースソフトウェアもアプリケーションソフトウェアも冗長性を意識する必要がないという点では、広範な用途で利用することができます。特殊なディスク装置を用意する必要があるため、コストがかかるデメリットはありますが、処理性能の劣化がまったくないという長所があります。既にデータベースが 1 台で稼働しているシステムの冗長性を高める用途としては最適です(図 5-2)。

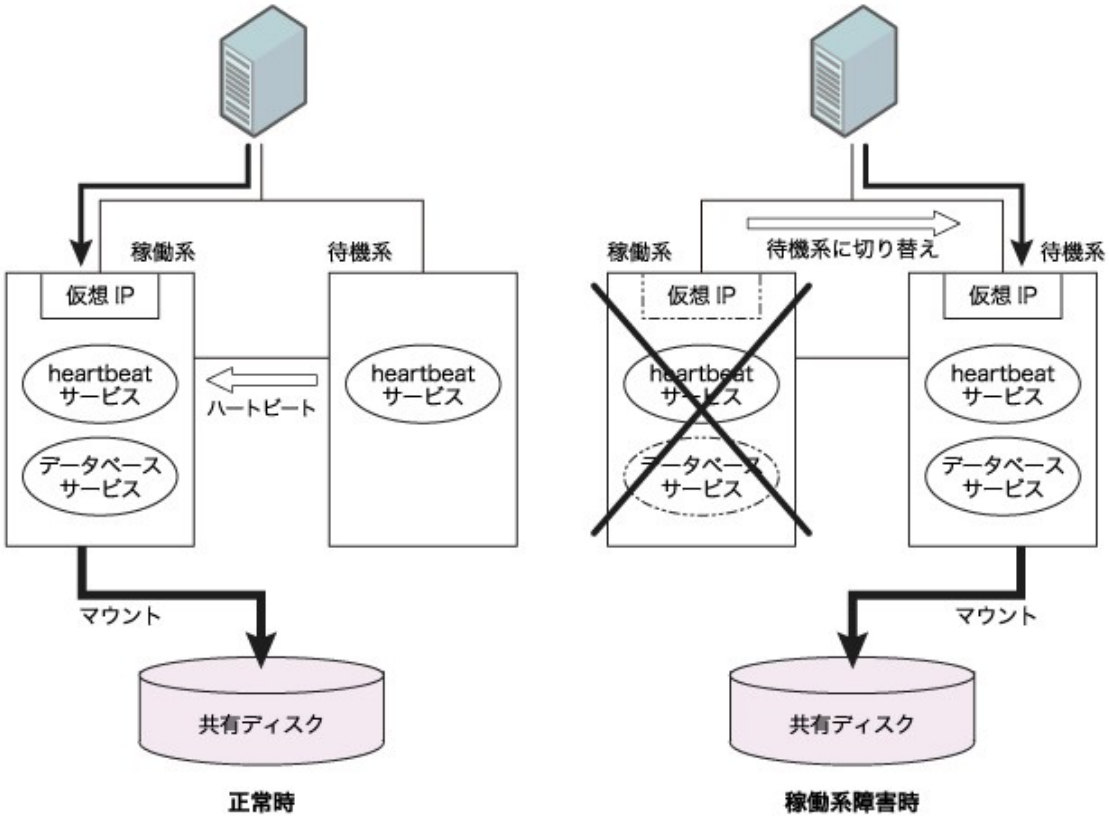
図 5-2: アクティブ・スタンバイによる冗長化のシステム構成



5.2.1　Linux での実装

Linux では、共有ディスクとして iSCSI かファイバチャネルのディスクを利用することで、このモデルを実現できます。アクティブ・スタンバイのための仕組みとしては、heartbeat を使うことができます。データベースファイルの保存場所を共有ディスク上にすることができれば、どのデータベースソフトウェアでも利用することができます(図 5-3)。そのため、PostgreSQL, MySQL, LDAP など、様々なデータベースで使うことができます。

図 5-3:heartbeat による冗長化のソフトウェア構成



heartbeat を使って冗長化を行う場合には、次のような設定を行います。

- データベースの設定ファイルを共有ディスクに移動する
  （本来の設定ファイルの場所には、シンボリックリンクを作成しておく）
- データベースサーバと共有ディスクをリソースに登録する
- 共有ディスク、データベースサーバの順に起動されるようにする

なお、具体的な事例については、8 章および 9 章で詳しく解説します。

5.2.2　PostgreSQL の冗長化

PostgreSQL を冗長化する場合には、データベースファイルを共有ディスク上に配置します。また、共有リソースとしては、共有ディスクと PostgreSQL サービスを登録します。

共有リソースの定義

次は、PostgreSQL の標準的なサービススクリプトをそのまま heartbeat の共有リソースとして使うことができる場合の設定例です。

▼*heartbeat リソースの設定例（/etc/ha.d/haresources）*

```
host01 Filesystem::/dev/sdb1::/data::ext3 postgresql 192.168.1.129/24
```

PostgreSQL の標準的なサービススクリプトが、引数に status を指定した場合に「running」「stopped」を含むメッセージを出力しない場合には、3.2.6 項で解説しましたようにラッパープログラムが必要です。

Filesystem の引数の/dev/sdb1 は、共有ディスク上のデバイスファイルで、この例では/data にマウントします。

データベースディレクトリの変更

PostgreSQL のデータベースディレクトリの設定は、起動時に指定することができます。Fedora や CentOS などのディストリビューションでは、/etc/sysconfig/postgresql に次のように設定することで、データベースディレクトリを変更できます。

▼*PostgreSQL のデータベースディレクトリの変更（/etc/sysconfig/postgresql）*

```
PGDATA=/data/pgsql/data
```

このディレクトリには、あらかじめ initdb コマンドを使ってデータベースディレクトリを作成しておきます。PostgreSQL では、設定ファイルもこのデータベースディレクトリに保管されますので、これ以外のシンボリックリンクなどの設定は必要ありません。

5.2.3　MySQL の冗長化

MySQL を冗長化する場合にも、同様にデータベースファイルを共有ディスク上に配置します。また、MySQL の設定ファイル（/etc/my.cnf）も、共有ディスク上に配置します。

シンボリックリンクとディレクとの準備

MySQL は、設定ファイルとして/etc/my.cnf を参照します。このパスで、共有ディスク上のファイルを参照するようにシンボリックリンクを設定します。

▼*設定ファイルの移動*

```
host1# mount /dev/sdb1 /data                          ←共有ディスクをマウント
host1# mkdir -p /data/mysql/etc                       ←設定の保管場所を作成
host1# mkdir -p /data/mysql/data                      ←データの保管場所を作成
host1# mv /etc/my.cnf /data/mysql                     ←共有ディスクへ移動
host1# ln -s /data/mysql/etc/my.cnf /etc              ←シンボリックリンク
```

```
host1# umount /data                                    ←マウントを解除
```

待機系のサーバでは、設定ファイルディレクトリをバックアップして、シンボリックリンクを作成します。

▼*待機系サーバのシンボリックリンクの例*

```
host2# mv /etc/my.cnf /etc/my.cnf.org
host2# ln -s /data/mysql/etc/my.cnf /etc
```

共有リソースの定義

/etc/ha.d/haresource に共有リソースを定義します。次は、MySQL の標準的なサービススクリプト（/etc/init.d/mysqld）をそのまま heartbeat の共有リソースとして使うことができる場合の設定例です。

▼*heartbeat リソースの設定例（/etc/ha.d/haresources）*

```
host01 Filesystem::/dev/sdb1::/data::ext3 mysqld 192.168.1.129/24
```

MySQL の標準的なサービススクリプトが、引数に status を指定した場合に「running」「stopped」を含むメッセージを出力しない場合には、3.2.6 項で解説しましたようにラッパープログラムが必要です。

データベース配置ディレクトリの調整

データベースの配置ディレクトリは、/etc/my.cnf で定義されていますので、次のように変更します。

▼*MySQL のデータベースディレクトリの変更例（/data/mysql/etc/my.cnf）*

```
datadir=/data/mysql/data
```

5.2.4　OpenLDAP の冗長化

OpenLDAP を冗長化する場合にも、同様にデータベースファイルを共有ディスク上に配置します。

シンボリックリンクの作成

通常は/etc/openldap/に配置されている設定ファイルを共有ディスク上に配置し、シンボリックリンクを作成します。

▼*設定ファイルの移動*

```
host1# mount /dev/sdb1 /data                              ←共有ディスクをマウント
host1# mkdir -p /data/openldap/data                       ←データの保管場所を作成
host1# mv /etc/openldap /data/openldap/etc                ←共有ディスクへ移動
host1# ln -s /data/openldap/etc /etc/openldap             ←シンボリックリンク
host1# umount /data                                       ←マウントを解除
```

待機系のサーバでは、設定ファイルディレクトリをバックアップして、シンボリックリンクを作成します。

▼*待機系サーバのシンボリックリンク*

```
host2# mv /etc/openldap /etc/openldap.org
host2# ln -s /data/openldap/etc /etc/openldap
```

共有リソースの定義

/etc/ha.d/haresource に共有リソースを定義します。次は、slapd の標準的なサービススクリプト（/etc/init.d/ldap）をそのまま heartbeat の共有リソースとして使うことができる場合の設定例です。

▼*heartbeat リソースの設定例（/etc/ha.d/haresources）*

```
host01 Filesystem::/dev/sdb1::/data::ext3 ldap 192.168.1.129/24
```

OpenLDAP の標準的なサービススクリプトが、引数に status を指定した場合に「running」

「stopped」を含むメッセージを出力しない場合には、3.2.6 項で解説しましたようにラッパープログラムが必要です。

データベース配置ディレクトリの調整

OpenLDAP のデータベースの配置ディレクトリは、/etc/openldap/slapd.conf で定義されていますので、次のように変更します。

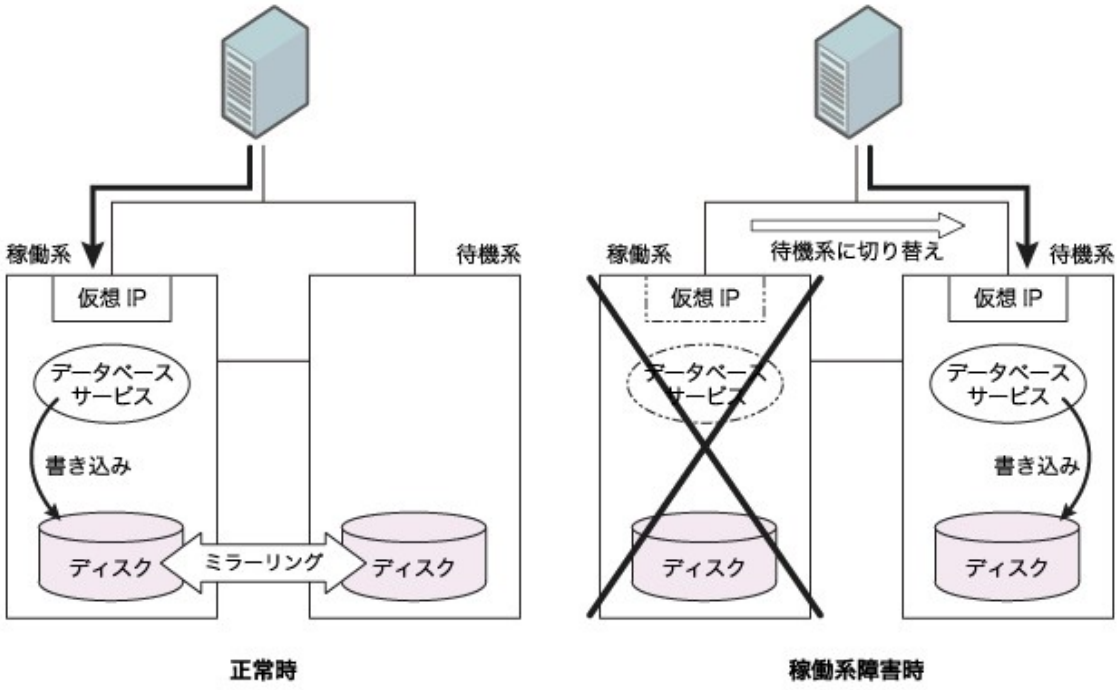
▼*slapd のデータベースディレクトリの変更（/data/openldap/etc/slapd.conf）*

```
include         /etc/openldap/schema/core.schema
include         /etc/openldap/schema/cosine.schema
include         /etc/openldap/schema/inetorgperson.schema
include         /etc/openldap/schema/nis.schema

allow bind_v2

pidfile         /var/run/openldap/slapd.pid
argsfile        /var/run/openldap/slapd.args

database        bdb
suffix          "dc=designet,dc=jp"
rootdn          "cn=Manager,dc=designet,dc=jp"
rootpw          {SSHA}A/9vWhxe6Ek7JWI0iwQJDnr8QgOqKayF

directory       /data/openldap/data                                    ←　変更

index objectClass                       eq,pres
index ou,cn,mail,surname,givenname      eq,pres,sub
index uidNumber,gidNumber,loginShell    eq,pres
index uid,memberUid                     eq,pres,sub
index nisMapName,nisMapEntry            eq,pres,sub
```

5.3　アクティブ・スタンバイ（ネットワークミラー）による冗長化

共有ディスクの代わりに、ネットワークミラーの機能を使ってアクティブ・スタンバイのデータベースシステムを構成することができます(図 5-4)。共有ディスクで構成するのと同じように、データベースソフトウェアもアプリケーションソフトウェアも冗長性を意識する必要がなく、広範な用途で利用することができます。共有ディスクとは異なり、特殊なディスク装置を用意する必要がないため低コストで実現することができますが、処理性能は単独のデータベースサーバに比べるとやや劣化します。しかし、演算処理の処理結果としてディスクに対する変更部分だけが同期されますので、レプリケーション型のデータベースと比べると性能の劣化は最小限です。既にデータベースが 1 台で稼働していて性能に余裕のあるシステムの冗長性を高める用途としては最適です。

図 5-4:アクティブ・スタンバイ（ネットワークミラーリング）による冗長化のシステム構成



5.3.1　Linux での実装

Linux では、DRBD を使ってネットワークミラーリングを実現できます。アクティブ・スタンバイのための仕組みとしては、heartbeat を使うことができます。データベースファイルの保存場所を共有ディスク上にすることができれば、どのデータベースソフトウェアでも利用することができます。そのため、PostgreSQL、MySQL、LDAP など、様々なデータベースで使うことができます。

heartbeat を使って冗長化を行う場合には、次のような設定を行います。

- DRBD のネットワークミラーリングの設定を行い、共有ディスクを作成します。

- データベースの設定ファイルを共有ディスクに移動する
  （本来の設定ファイルの場所には、シンボリックリンクを作成しておく）
- データベースサーバと共有ディスクをリソースに登録する
- DRBD、共有ディスク、データベースサーバの順に起動されるようにする

DRBD のネットワークミラーリングの設定が必要なことを除き、実際の設定は共有ディスクを使う場合とほぼ同じです。ただし、heartbeat のリソースファイルの書き方だけが異なります。

▼*PostgreSQL の場合の heartbeat リソースの設定例（/etc/ha.d/haresources）*

```
host01 drbddisk Filesystem::/dev/drbd0::/data::ext3 postgresql 192.168.1.129/24
```

この例のように Filesystem リソースの前に、drbddisk リソースを開始する必要があります。

5.4　動的レプリケーションによる冗長化

アクティブ・スタンバイクラスタを使ったデータベースの冗長化では、データベースのデータは実質的には共有ディスク上の 1 ヶ所だけに保管されていました。これに対して、動的レプリケーションでは、データベースへの処理要求を複製（空間レプリケーション）することで、複数のデータベースサーバに同じデータが保管されているようにします。処理要求を複製するため、各データベースサーバでは同じ演算が行われ、同じデータが保管されます。一般的に、この処理は、ネットワークミラーリングで処理結果だけを同期するのに比べると、データベースサーバに掛ける負荷が高くなります。しかし、ネットワークミラーリングは 2 台のサーバでしか行うことができませんが、動的レプリケーションを使えば 3 台以上のサーバで同じデータを持つことができます。

5.4.1　Linux での実装（pgpool）

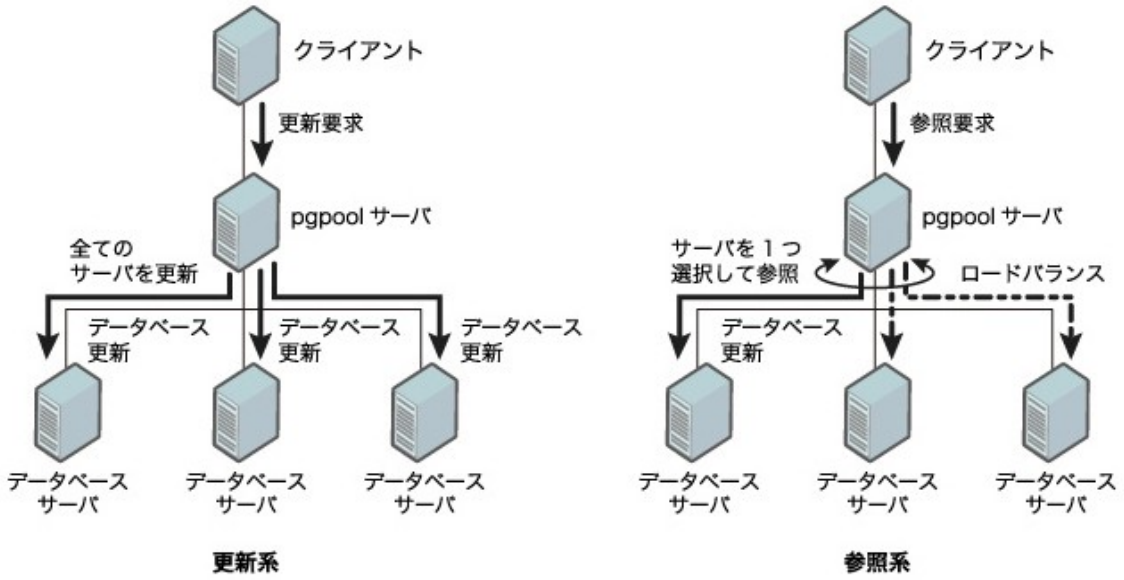
動的レプリケーションによる冗長化を行うソフトウェアとして注目されているのが pgpool です。pgpool は、PostgreSQL に対して動的レプリケーションを行うソフトウェアで、現在は Pgpool Global Development Group が開発・管理しています。pgpool はもともと石井達夫氏が開発しましたが、その後 SRA 社が IPA（独立行政法人　情報処理推進機構）の援助を受けて開発した pgpool Ⅱ へと開発が引き継がれています。現在は、SRA 社が商用サポートも行っています。

pgpool-Ⅱ では、次のような機能を提供しています。

- コネクションプーリング
  pgpool と PostgreSQL サーバ間の接続を切断せず、常時通信ができる状態にしておきます。これによって、PostgreSQL の接続処理による負荷を pgpool が代行し、PostgreSQL サーバの負荷を抑えることができます。
- レプリケーション
  データベースへの更新要求を複数の PostgreSQL サーバへリアルタイムに複製します。この機能を使って、動的レプリケーションを実現することができます。
- 負荷分散
  データベースへの参照要求のロードバランシングを実現します。
- 接続数の制限
  データベースへの接続数を制限し処理要求をキューイングすることで、データベースサーバへの負荷を一定に保ちます。
  パラレルクエリ
  複数のサーバに同時に検索を行うことで、検索時間を短縮します。

pgpool は、PostgreSQL クライアントと PostgreSQL サーバの間でプロキシ（中継サーバ）として動作します。pgpool では管理するサーバ群を**サーバプール**と呼びます。pgpool では、クライアントから送られてきた処理要求（SQL 文）を解析し、更新系の要求か、参照系の要求かを区別して処理することができます。更新系の要求の場合には、サーバプール内のすべてのサーバに同じ更新処理を送ります。したがって、すべてのサーバが同じデータの状態に保たれます。また、参照系の要求の場合には 1 つのサーバを選んでリクエストを送信します(図 5-5)。そのため、参照系の要求に対してはロードバランサとして動作します。つまり、システム全体で動的レプリケーションとロードバランシングの両方の機能を実現することができます。

図 5-5:pgpool の更新系処理と参照系処理



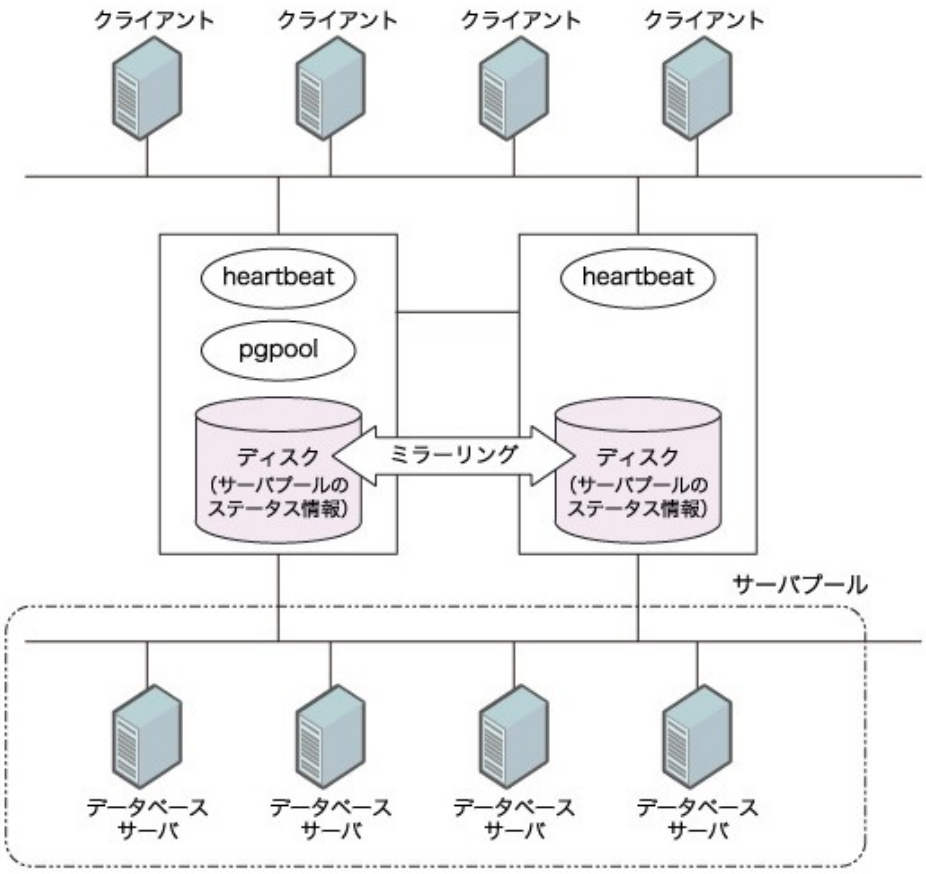
また、pgpool には PostgreSQL サーバの動作と状態を管理する機能があります。pgpool の更新処理では、クライアントからのリクエストは pgpool によって複製され、サーバプール内の PostgreSQL サーバに送られます。pgpool は、この処理結果を比較して、違う結果を返却したサーバを異常なサーバとして検知し、サーバプールから外します。また、異常な処理結果を返したサーバのデータベースファイルを rsync コマンドなどで強制的に同期させ、他のサーバと同じ状態にする**オンラインリカバリ**という機能も提供しています。

こうした更新の処理には比較的多くのコストが掛かりますので、システム全体の更新処理性能は 1 台のデータベースと比べて大きく劣化します。そのため、更新処理の多いシステムにはこの方法は向いていません。また、乱数やトランザクション ID など、同じ要求に対して異なる処理結果を返すような処理を行うことはできません。そのため、PostgreSQL クラインアントは、pgpool を利用することを前提に設計されたものでなければなりません。

pgpool のシステム構成

pgpool のレプリケーション機能を使ってシステムを冗長化する場合には、pgpool が単独障害点にならないよう、pgpool の冗長性を考慮する必要があります。図 5-6 は、そのシステム構成例です。

図 5-6:pgpool のシステム構成例

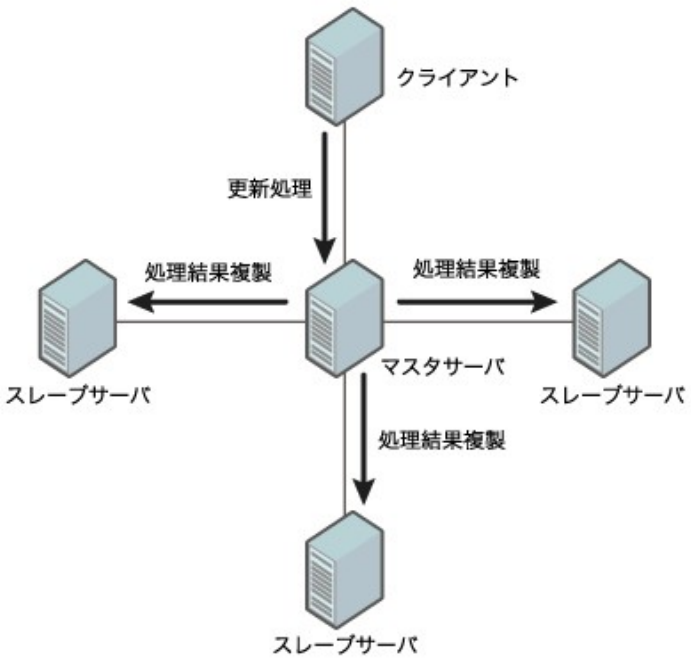


pgpool の冗長化では、pgpool が管理しているサーバのステータス情報が、フェイルオーバ時にも待機系に引き継がれるようにする必要があります。そのため、ステータス情報は、ネットワークミラーリングなどで実現した共有ディスク上に配置する必要があります。

5.5 シングルマスタレプリケーションによる冗長化

動的レプリケーションでは、更新要求を複製することで、すべてのデータベースサーバへ更新処理を行います。これに対して、シングルマスタレプリケーションでは、更新処理は 1 つのマスタサーバで行います。マスタサーバは、更新処理を行った結果を他のサーバへ複製します。マスタサーバから複製を受けるサーバを、スレーブサーバと呼びます。スレーブサーバは、データベースへの参照処理だけを受け付けます(図 5-7)。

図 5-7:シングルマスタレプリケーションの処理イメージ



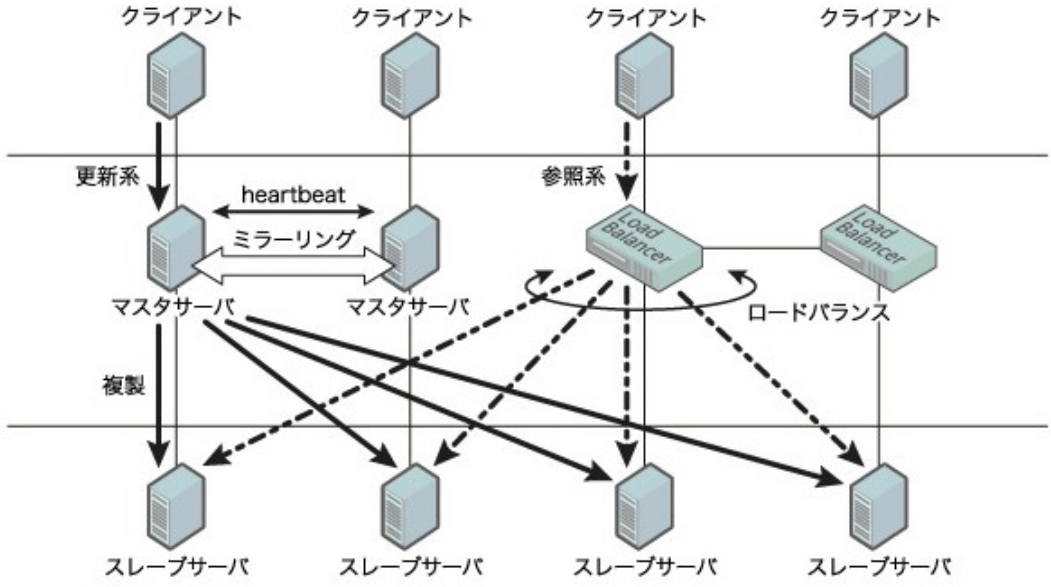
このシステム構成では、各クライアントが更新処理と参照処理を区別して行う必要があります。実際のシステム構成は、次の点を考慮して設計する必要があります。

- マスタサーバの冗長性
  マスタサーバが停止すると、システムへの更新ができなくなります。そのため、マスタサーバが単独障害点にならないように、マスタサーバをアクティブ・スタンバイクラスタなどで冗長化します(図 5-8)。一般的に、シングルマスタレプリケーションでは、マスタサーバへ更新を行ってからスレーブサーバにデータが反映されるまでに、タイムラグがあります。そのため、シングルマスタレプリケーションの仕組みを使ってアクティブ・スタンバイ構成を作成すると、タイミングによっては切り替え直前の更新が反映されない場合があります。つまり、マスタサーバを冗長化する場合には、レプリケーションの仕組みを使うのではなく、共有ディスクやネットワークミ

ラーリングの技術を使う必要があるのです。

- スレーブサーバのロードバランシング
  クライアントからの参照処理がスレーブサーバ間で分散されるように、ロードバランサなどを使います。ロードバランサも冗長化する必要があります。

図 5-8: シングルマスタレプリケーションでのシステム構成例



5.5.1　Linuxでの実装

Linuxでは、PostgreSQL、MySQL、OpenLDAPなどで、シングルマスタレプリケーションを行うことができます。どのソフトウェアでも、スレーブサーバのロードバランスや、障害時にスレーブサーバを自動的にマスタサーバへ変更する機能などは提供されていません。そのため、アクティブ・スタンバイとして構成する場合にも、参照系のロードバランシングで利用する場合にも、heartbeatやロードバランサなどの切り替え用の仕組みとの併用が必須です。

5.5.2　PostgreSQLでの構成

PostgreSQLでは、ストリーミングレプリケーションと呼ばれるレプリケーションを9.0版から利用できるようになりました。このレプリケーションでは、マスタサーバへ更新が行われトランザクションがコミットされた時に、スレーブサーバへ非同期で複製を行います。そのため、スレーブサーバへの変更の反映にはややタイムラグが発生します。

PostgreSQLでストリーミングレプリケーションを行うためには、マスターサーバとスレーブサーバの接続設定を行う必要があります。

マスタサーバ側の設定

マスタサーバでは、スレーブサーバからのレプリケーションのための接続に認証を実施することができます。

▼*PostgreSQL　マスタサーバのレプリケーション用の認証設定（pg_hba.conf）*

```
# TYPE  DATABASE        USER            CIDR-ADDRESS            METHOD
host1   replication     repuser         192.168.1.100/32        md5
```

スレーブサーバからの設定

スレーブサーバには、マスタサーバへ接続するための設定を行います。

▼*PostgreSQL スレーブサーバの設定例（recovery.conf）*

```
primary_conninfo = 'host=192.168.1.50 port=5432 user=repuser
password=replica'
```

5.5.3　MySQL での構成

MySQL は、クエリベースレプリケーションと行ベースレプリケーションをサポートしています。クエリベースレプリケーションでは、SQL のすべてのステートメントを複製します。ログファイルがコンパクトで事後の監査が可能ですが、動作してみなければ値がわからないようなランダム関数や日付を使った更新処理の場合には、正確にレプリケーションができない場合があります。一方、行ベースレプリケーションは処理結果を複製しますので、すべての SQL ステートメントに対して正確に複製を行うことができ、性能的にもメリットがあります。ただし、ログファイルが大きくなるなどのデメリットがあります。

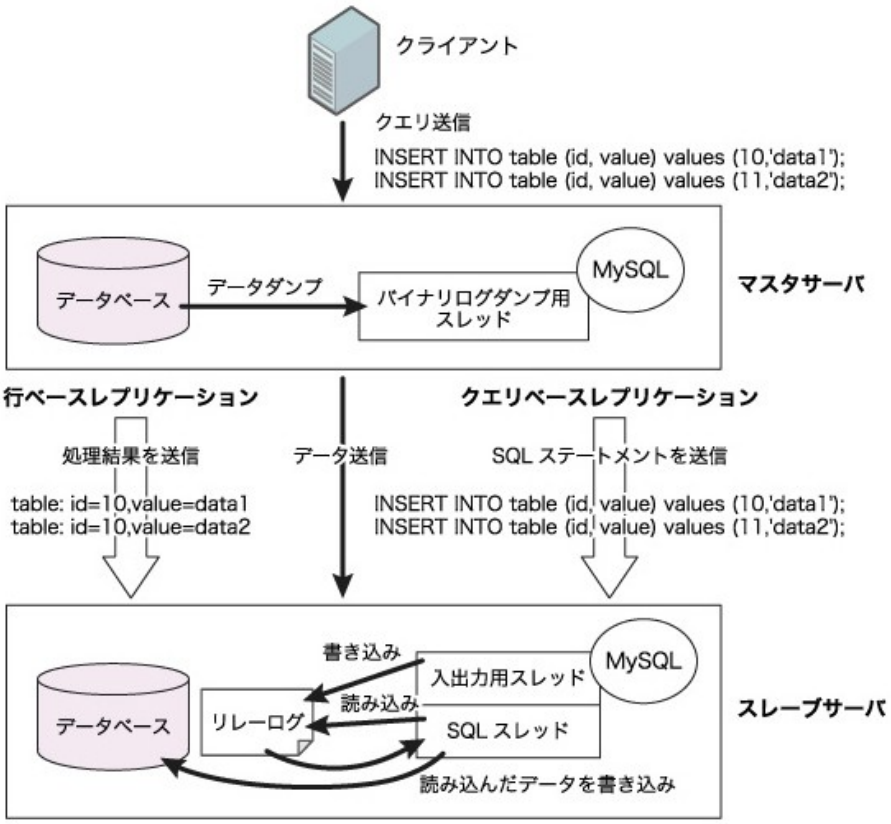
また、MySQL 5.1.8 以降では、ミックスベースレプリケーションがサポートされています。ミックスベースレプリケーションでは、通常はクエリベースレプリケーションで動作し、必要な場合に行ベースレプリケーションに自動的に切り替えることができます。

MySQL のレプリケーションは、図 5-9 のような手順で行われます。

- スレーブサーバで START SLAVE ステートメントを実行する
- スレーブサーバに入出力用のスレッドが起動され、マスタサーバに接続する
- マスタサーバではバイナリログダンプ用のスレッドが起動される
- バイナリログがスレーブサーバへ送られる
- ログを受信したスレーブサーバはリレーログに書き込む

- SQL スレッドがリレーログを読み、データベースに反映する

図 5-9:MySQL レプリケーションのシステム構成



MySQL のレプリケーションは、既存データの有無や設定パラメータによってかなり柔軟に行えるように設計されています。ここでは、まったくデータのない新規のサーバで設定を行う場合の手順を例にとって解説します。

マスタサーバの設定

マスタサーバにはサーバ ID を設定する必要があります。設定は、/etc/my.cnf で行います。次は、その設定例です。

▼*MySQL マスタサーバの設定（/etc/my.cnf）*

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

また、次のようにレプリケーション用のユーザを作成し、マスタ情報を取得しておきます。

▼*MySQL マスタサーバへのレプリケーションユーザの作成とマスタ情報の取得例*

```
$ mysql
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'repl'@'slave.designet.jp' IDENTIFIED BY 'replica';
mysql> FLUSH TABLES WITH READ LOCK;                    ←　テーブルのロック
mysql> SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+---------------+----------+--------------+------------------+
mysql> UNLOCK TABLES;                                  ←　ロックの解除
```

スレーブサーバの設定

スレーブサーバにも、サーバ ID を設定する必要があります。設定は、マスタサーバと同様に/etc/my.cnfで行います。

▼*MySQL スレーブサーバの設定（/etc/my.cnf）*

```
[mysqld]
server-id=2
```

スレーブサーバには、さらにマスタサーバへの接続の設定を行います。次は、その接続設定の例です。

▼*スレーブサーバへマスタサーバを追加するオペレーション例*

```
$ mysql
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master.designet.jp',
    ->     MASTER_USER='repl',
    ->     MASTER_PASSWORD='replica',
    ->     MASTER_LOG_FILE='mysql-bin.003', ←サーバから取得したログファイル名
    ->     MASTER_LOG_POS=73;               ←サーバから取得したログポジション
mysql> START SLAVE;                         ←レプリケーションの開始
```

5.5.4　OpenLDAP での構成

OpenLDAP では、Version2.2 から同期レプリケーションと呼ばれるレプリケーション方式をサポートしています。この機能を使って、シングルマスタレプリケーションを行うことができます。OpenLDAP の同期レプリケーションでは、マスタサーバを**プロバイダ**、スレーブサーバを**コンシューマ**と呼びます。

OpenLDAP の同期レプリケーションでは、コンシューマ側からプロバイダに接続し同期を取ります。定期的にプロバイダに接続して同期を取る refreshOnly と、常に接続を保ち必要に応じて同期を

取る refreshAndPersist の二つの同期タイプをサポートしています。またプロバイダでは、コンシューマがどのデータまでを同期したかという情報（Cookie）と同期していない情報（セッションログ）を管理しています。通常は、最後の同期の後に行われた更新の差分のみを同期しますが、同期していない情報が多くなるとすべての情報を再取得するフル同期を行います。

プロバイダの設定

OpenLDAP のプロバイダ側では、同期レプリケーション用のオーバレイモジュール（syncprov）にセッションログの保存数を設定します。次は、その設定例です。

▼*OpenLDAP のプロバイダの設定例（/etc/openldap/slapd.conf への追加分）*

```
overlay syncprov

syncprov-sessionlog 100
```

コンシューマの設定

コンシューマには、プロバイダへの接続方法の設定と、同期条件の設定を行います。次は、その設定例です。

▼*OpenLDAP のコンシューマの設定例（/etc/openldap/slapd.conf への追加分）*

```
syncrepl rid=123
        provider=ldap://ldap1.mydomain.jp:389/          ← アドレスとポートの設定
        bindmethod=simple                               ← 認証方式
        binddn="cn=Admin,dc=mydomain,dc=jp"             ← 接続認証用 DN
        credentials=admin                               ← 接続認証用パスワード
        type=refreshAndPersist                          ← 同期方法
        interval=00:00:05:00                            ← 同期間隔（5 分）
        searchbase="dc=mydomain,dc=jp"                  ← 同期を行う DN
        filter=(|(objectClass=organization)             ← 同期の条件
                 (objectClass=organizationalRole)
                 (objectClass=organizationalUnit)
                 (objectClass=posixAccount)
                 (objectClass=person))
        scope=sub                                       ← 同期の範囲
```

5.6　マルチマスタレプリケーションによる冗長化

シングルマスタレプリケーションでは、更新系の処理を行うマスタサーバが 1 台しか作成できないため、マスタサーバが単独障害点となってしまう問題があります。そのため、別の冗長化手法と組み合わせなければ、システム全体の稼働率を向上することができません。これに対して、マルチマスタレプリケーションでは、システム内にマスタサーバを複数個用意することができます。

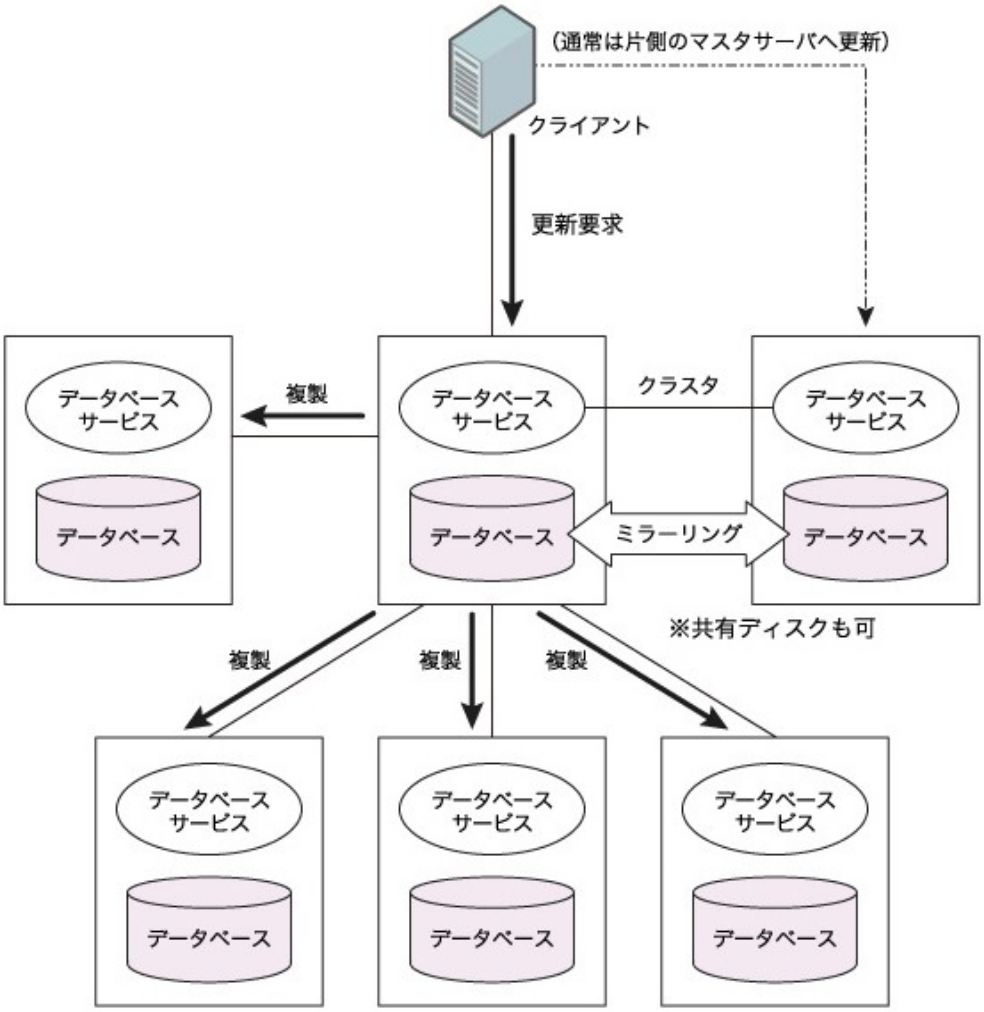
シングルマスタレプリケーションでは、一般に非同期のレプリケーションが使われています。非同期のレプリケーションでは、マスタサーバへの更新が完了した後でスレーブサーバへの複製が行われます。しかし、マルチマスタレプリケーションでは同期レプリケーションが使われます。同期レプリケーションでは、マスタサーバへ更新を行うと、その更新トランザクションの中で別のマスタサーバへの複製までを行います。そのため、各サーバの状態は常にまったく同じとなります。

5.6.1　Linux での実装

Linux では、MySQL、OpenLDAP などでマルチマスタレプリケーションを行うことができます。また、PostgreSQL をマルチマスタ構成で稼働できるように改良した PGCluster というソフトウェアがあります。

ただし、これらの実装では、複数のマスタサーバに同時に多くの更新を行った場合に、完全に整合性が取れることまでを保証しているわけではありません。そのため、通常はどれか 1 つのマスタサーバを使い、障害時には他のマスタサーバを使うというように、複数のマスタサーバを同時に利用しないのが一般的です(図 5-10)。また、結局のところすべてのマスタサーバに更新処理を行うため、更新処理のスピードは単一のサーバの場合よりも劣化します。そのため、性能的には前項で解説したように、マスタサーバを共有ディスクやネットワークミラーリングでアクティブ・スタンバイの構成としておき、スレーブサーバへの参照要求だけをロードバランシングするシステム構成の方が高速になることが多く、システム的にも安定すると考えられます。

図 5-10:シングルマスタレプリケーションの処理イメージ



6章 クラスタシステムの監視

アクティブ・スタンバイクラスタやロードバランシングによる冗長化システムでは、MTTR を最小に抑えるため、システムの障害を少しでも早く検知し復旧を行う必要があります。こうした障害の検知の方法としては、障害の程度に応じて様々な手法が使われています。本章では、障害の検知方法について学習します。

6.1　ハードウェア障害

発生する可能性の高いハードウェアの障害については、システムを構築するときにあらかじめ検知する方法を考えておく必要があります。ハードウェア障害は、その影響度の大きさから次のように分類することができます。

- 軽度
  一部のプロセスの処理だけに問題が発生する。その他の処理は正常に行える。システムへのログインなどは正常に行える。
- 中度
  ほとんどのプロセスに影響が及び、プロセスが処理待ちのままとなることがある。カーネルは動作しているため、ping やコネクション開設要求などには応答するが、システムへのログインも正常に行えず、サービスも正常に動作しない。そのため、一旦この状態に陥ると外部からシステムをコントロールすることはできなくなるが、カーネルの機能は生き残っているため、あらかじめ定義した動作は実施できる。
  この状態になると、あいまいな状態に陥ることが多く、検知が難しい。それにも関わらずネットワークの機能は中途半端に動作しているため、IP アドレスの切り替えが失敗したり、共有ディスクへの変更が継続したりする可能性があり、さらに問題を引き起こす可能性が高い。
- 重度
  カーネルが PANIC したり、まったく動作することができず、システムが完全に停止する。
  ping やコネクション開設要求にも応答しなくなる。

ハードウェアの種類によって、障害の発生による影響度も異なります。

- ハードディスク
  ハードディスク障害が発生した場合には、様々な現象が起こります。障害の影響が低い順に、発生する可能性のある問題をリストアップすると次のようになります。
  - ファイルの書き込みや読み込みが失敗し、ファイルシステムからエラーが返却される。（軽度）
  - ファイルの書き込みや読み込みがブロックし、ハードディスクへアクセスするすべてのプロセスが処理待ちのままとなる。（中度）
  - システムがメモリをスワップしようとしてエラーになると、カーネルが PANIC し、システムが完全に停止する。（重度）
- ネットワークインタフェース
  ネットワークインタフェースの障害の影響は、通信機能に対してのみ発生する。そのため、影響は限定的でシステム自体は動作している。（軽度）
- メモリ、CPU
  メモリや CPU の障害が発生すると、システムは動作し続けることができない。（重度）

6.2　サービス障害

ハードウェアが正常に動作していればすべての機能が順調に動いている、とは限りません。機能が正常に動作しない原因としては、次のようなものが考えられます。

- ハードウェア障害を起因とするアプリケーションの動作不良
- カーネルのバグ
- システム設定の間違い
- アプリケーションプログラムのバグによるプロセスの停止や誤動作
- システムリソース（メモリ、処理能力、ディスク容量）の不足

こうした問題のためにサービスが動作しない場合は、ハードウェア障害で説明した障害の程度に当てはめると、軽度の障害ということになります。

6.3 障害の検知と復旧

ハードウェア障害やサービス障害は、適切な監視を行うことで検知することができます。軽度～中度の障害の場合には、自システムが適正に動作しているかを確認することで検知できます。軽度の障害の検知は、次のようにサービスが正しく動作していることを確認する**サービス監視**を定期的に実施することで行うのが一般的です。

- サービスの動作に必要なプロセスが起動されていることを確認する。
- 実際にサービスを利用する手順をシミュレートする。
- アプリケーションのログなどに、警告メッセージなどの異常な情報が出力されていないかを確認する。

こうした方法でサービスを確認し停止や異常を検知したら、サービスの停止、フェイルオーバー、シャットダウンなど、あらかじめ用意した冗長化の仕組みを使ってシステムを切り替えます。

6.3.1 サービスの監視

Linux にはサービスが正常に動作していることを確認するための様々な仕組みが用意されています。また、コマンドラインから利用できる様々なユーティリティプログラムをうまく使えば、自由にサービス監視を実装することができます。

次は、wget という Web サーバからホームページをダウンロードするユーティリティプログラムを使って、自サーバの Web サービスを監視するプログラムの例です。

▼*Web サービスの監視プログラムの例*

```
#! /bin/sh
RETRY=3
TIMEOUT=3
OKSTRING="${OKSTRING:=OK}"

TMPFILE=/tmp/.webcheck.tmp.$$
LOGFILE=/tmp/.webcheck.log.$$

wget --output-document=$TMPFILE --tries=$RETRY --timeout=$TIMEOUT $1 > $LOGFILE 2>&1
if [ $? -ne 0 ]
then
       cat $LOGFILE

       rm -f $TMPFILE $LOGFILE
        exit 1
fi

CONTENTS=`cat $TMPFILE`
```

```
if [ "$CONTENTS" != "$OKSTRING" ]
then
       echo "Invalid contents."
        echo ""
       cat $TMPFILE

       rm -f $TMPFILE $LOGFILE
        exit 1
fi

rm -f $TMPFILE $LOGFILE
exit 0
```

6.3.2　ログの監視

ログの検査には swatch というプログラムが使われます。swatch は、ログファイルに出力されるメッセージを常時チェックし、特定の文字列を検出された際にメールを送信したり、コマンドを実行したりといったアクションを設定することができます。次は、/var/log/app.log というログファイルに Critical という文字列が出力されたらシステムを停止するという swatch の設定ファイルとコマンドの実行例です。

▼*swatch の設定ファイル（/etc/swatch/critical.cfg）*

```
watchfor /Critical/
    exec /usr/bin/halt
```

▼*swatch の実行例*

```
$ /usr/bin/swatch --use-cpan-file-tail  --config-file /etc/swatch/critical.cfg  --tail-file=/var/log/app.log
```

6.3.3　軽度障害の対策

軽度な障害の場合、システムはまだ自律的に動作することができます。したがって、定期的に動作エラーが発生していないかを検査し、障害を検知した場合には、サービスの停止、フェイルオーバー、シャットダウンなどを実施するようにします。

heartbeat では、こうした検査プログラムを自動的に実行するように設定を行うことができます。

▼*heartbeat　検査プログラムの実行の設定（/etc/ha.d/ha.cf）*

```
# 外部プログラム
respawn root /usr/local/bin/check_active
```

6.3.4　中度障害の対策

中程度の障害は、カーネルの機能は動作していても、アプリケーションのプロセスは動作しないあいまいな障害状態です。外部サーバとの通信は、場合によっては継続している可能性もあり、それでもシステムは正常に動作していません。そのため、外部からは状況を確認するのが極めて難しい状況になります。また、障害を検知しても正常にサービスを停止したり、フェイルオーバやシャットダウンができなくなったりする場合があります。さらに、検査プログラムが正常に動作しなくなり、障害を検知することができなくなったりする場合もあります。

Linux では、こうした状況を想定し、**watchdog デバイス**を用意しています。watchdog デバイスは、該当のデバイスに定期的にアクセスする監視アプリケーションとともに利用します。例えば、30 秒間に 1 度アクセスしてくるはずの監視アプリケーションが、2 分間もアクセスしてこないような状況が発生した場合には、watchdog デバイスはシステムのプロセスが正しく動作していない状況が発生したと判断し、わざとシステムダウンを引き起こします。それによりシステムが完全に停止するため、IP アドレスの切り替えが失敗したり、共有ディスクのデータを 2 つのサーバから更新してしまうなどの問題を引き起こす可能性がなくなります。

heartbeat では、次のように設定しておくことで、watchdog デバイスに対する監視アプリケーションとしても動作します。

▼*heartbeat　検査プログラムの実行の設定（/etc/ha.d/ha.cf）*

```
# ウォッチドック
watchdog /dev/watchdog
```

また、heartbeat では待機系サーバが稼働系サーバの障害を検出した場合に、相手サーバを完全に停止させるための機能として **STONITH** をサポートしています。STONITH を使えば、電源管理サーバなどと連携して、強制的に相手サーバの電源を停止することができます。次の例は、APC の電源管理装置（Rack PDU）を連携して強制的に電源を切るための heartbeat の設定です。

▼*heartbeat　STONITH 設定（/etc/ha.d/ha.cf）*

```
# STONITH
stonith_host sv01 external/rackpdu pw private 2
```

heartbeat は表 6-1 のような多くのデバイスと連携して動作することができます。

▼*表 6-1:heartbeat が連携できる STONITH デバイス*

名称	デバイス名
apcmaster	APC MasterSwitch
apcsmart	APC Smart UPS
baytech	Baytech RPC
cyclades	Cyclades AlterPath PM

ibmrsa	IBM RSA
ibmrsa-telnet	IBM RSA(telnet 経由)
ipmi	IPMI
rackpdu	APC Switched Rack PDU
riloe	COMPAQ RILOE
ssh	ssh
vmware	VMware
ibmhmc	IBM HMC
meatware	ミートウェア
nw_rpc100s	Night/Ware RPC1005
rcd_serial	RC Delayed Serial
rps100	RPS-10M
suicide	仮想デバイス
wti_nps	Western Telematic Network Power Switch

6.3.5　重度障害の対策

重度障害に陥ると、システムはほとんど動作していません。そのため、外部のサーバからネットワークを通じて ping やポート監視を行うことで十分に検出することができます。heartbeat などのクラスタソフトウェアでは、相手サーバとの情報交換が行えなくなるため容易に障害を検出することができます。

7章 システム監視

オープンソースを使えば、様々なソフトウェアを自由に集めてシステムを作成することができます。さらに、クラスタリングやロードシェアリングの技術をうまく使えば、稼働率の高いシステムを作ることも可能です。しかし、どのような稼働率を高める仕組みを導入しても、システムの障害を完全に取り除くことはできません。そのため、システムがどのような状態にあるのかを常に管理しておく必要があります。本章では、システム監視の目的と方法について学びます。

7.1 システム監視の目的

システム監視というと、多くの人は障害を発見するために行うものだと考えています。もちろんシステム全体の稼働率を高めるためには、障害をできるだけ早く発見し、すぐに対策を実施することが望ましいのは言うまでもありません。しかし、監視の最大の目的は、障害の発見ではなく障害の予防です。特に、クラスタリングやロードシェアリングなどの方法で冗長性が確保されているシステムでは、ハードウェアの故障などでシステム全体が停止する危険性はそれほど高くありません。ロードバランサや待機系のサーバから行われるサービス監視の仕組みにより、サービスがフェイルオーバしたときにその通知が行われるような仕組みを導入しておけば、障害を知ることも難しくありません。実は、障害そのものよりも根本的なリソースの不足などの事象の方がはるかに厄介で、回復が難しいことが多いのです。

例えば、データベースサーバでどんどん新しいデータが投入されていくようなシステムの場合には、データベースのデータ容量が増加し、ディスクに保管しきれないような状態が発生する可能性があります。このような問題が発生すると、物理的にハードディスクを増設するなどしなければ障害から回復できない可能性があり、場合によってはディスクを調達する間、長期のシステム停止を余儀なくされるかもしれません。もし共有ディスクが満杯になれば、どのような優秀なクラスタリングの仕組みがあっても、障害から復旧することはできないのです。

監視の目的は、次のように考えることができます。

- システムの定常状態を観察する。
- システム障害の兆候を発見する。
- システムの拡張の時期を知る。
- システムの障害をいち早く発見する。

多くの Linux ディストリビューションでは、用途に合わせてインストールするソフトウェアを自由に選択することができます。また、自分でソースコードをダウンロードして、コンパイル・インストールすることも可能です。このように、自由にインストールソフトウェアをカスタマイズし、オリジナルのシステムを作り出せることは、オープンソースソフトウェアを利用する大きなメリットでもあります。しかし、各ソフトウェアのマニュアルや公式サイトを見ても、ソフトウェアのインストール方法は説明されていても、運用上の注意や監視すべき項目が書かれていることはほとんどありません。これは、自由に組み合わせることができる反面、設定の内容や一緒に使うアプリケーションによって、大きく運用の仕方や注意点が変化してしまうからです。つまり、実際のシステムの特性は、管理者自身がシステムをよく観察して調べる必要があるのです。

7.2　システムの状態を記録する

システムの正確な特性を知るためには、まず管理すべきシステムのことをよく知ることから始める必要があります。そのためには、そのシステムの動作に関する各種の指標を定期的に観察し、その変化を知る必要があります。

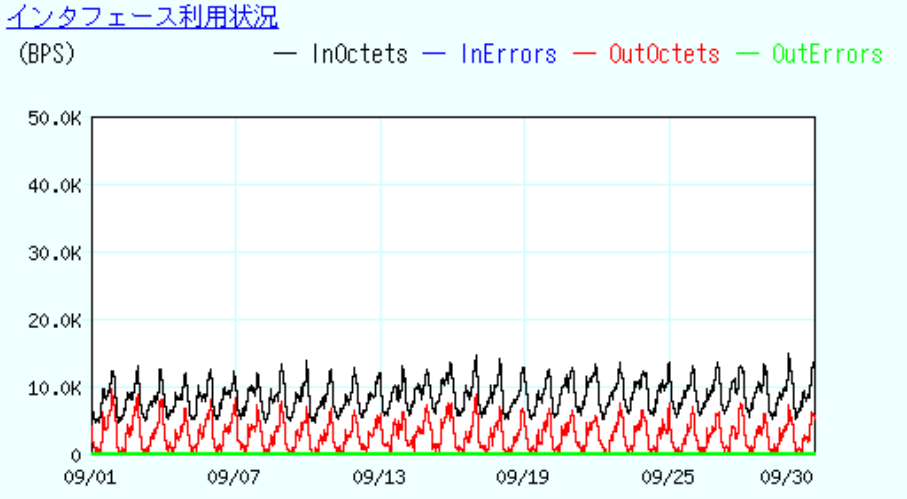
7.2.1　状態管理の概要

システムの状態を把握するために観察する必要のある項目は、実に多種に渡ります。

- システムの利用状況に関するするもの
  - CPU 利用率
  - サーバの負荷状況（処理待ちプロセス数）
  - システム全体のプロセス数
  - システムのログメッセージ
- リソースの利用状況に関するもの
  - ファイルシステムのデータ量
  - ネットワークインタフェースの通信量
  - メモリの使用量
  - SWAP の使用量
  - リソース関連のログメッセージ
- ソフトウェアの利用状況に関するもの
  - 利用量（利用回数）
  - ソフトウェアのプロセス数
  - ソフトウェアのログメッセージ

図 7-1 は、Web サーバのネットワークシステムを通過するデータの量を定期的に記録してグラフにしたものです。

図 7-1:インタフェース利用状況



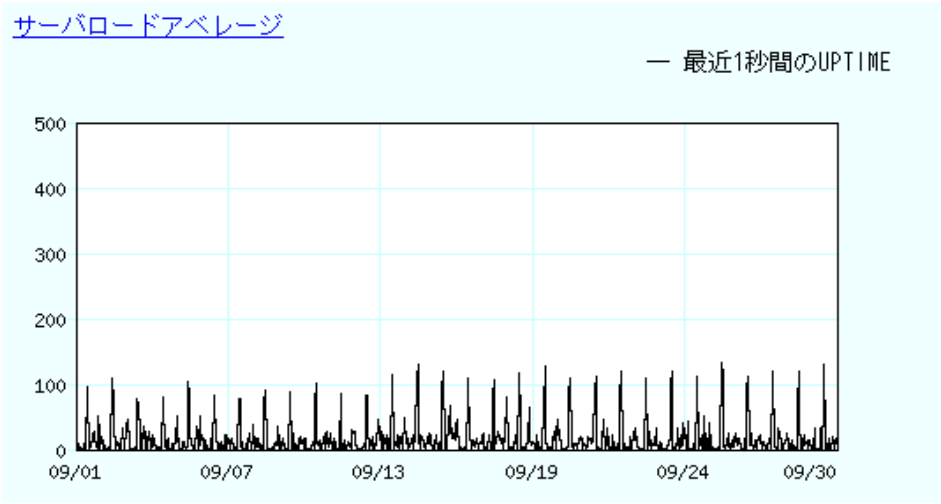
このサーバでは、通信量が定期的に増減を繰り返していることが分かります。Web サーバの閲覧者が多いときには通信量が増加し、Web サーバの閲覧者が少ないときには通信量が減少すると予想することができます。最近のサーバのネットワークインタフェースはギガビット対応ですので、この指標だけを見れば、増減はあるもののシステムには随分と余裕があるように思えます。

一方、図 7-2 は、このサーバの 1 分間のロードアベレージの平均（uptime）を定期的に記録してグラフにしたものです。ロードアベレージは、サーバの処理待ちプロセス数の平均を示す値で、サーバの負荷状況を観察するのに使います。ロードアベレージが 100 ということは、常に平均 100 個のプロセスが処理待ちになっていることを示しています。通信量のグラフと見比べると次のようなことが分かります。

- このサーバは通信量の割にシステムの負荷が高い。
- 通信量が多いときには、サーバの負荷が高い。
- サーバへのアクセス量が周期的に変化している。
- アクセスのピーク時間には、サーバの負荷状況も高い状態になる。

例えば同じ Web ページの応答が遅いという現象でも、それがアクセスのピーク時間に起きているのか、あるいは、ほとんどアクセスがない時間に起きているのかで、重大性がまったく異なるのです。このように、サーバの状態や特性をきちんと知るためには、1 つの指標だけを観察するのではなく、複数の指標を観察する必要があります。

図 7-2:サーバロードアベレージ



7.2.2　状態管理の観点

システム状態に関するデータを収集した上で、それぞれのデータが示す情報を読み取り、適切な対応をする必要があります。ここでは、それぞれのデータを扱う観点について解説します。

ログの管理

ログには、様々な情報が出力されます。これらの情報を適切に利用する必要があります。

異常時に出力されるログは、内容によって重要度がまったく異なります。ほとんど気に留める必要のないものもあれば、絶対に出力されてはならないものまで、程度には大きく差があります。

正常時に出力されるログは、不要と思われがちですが、実は「正常」であることを示すデータとしてとても重要です。したがって、「異常を示すログが出ていないか」、ということと同時に、「正常を示すログが出ているか」もチェックする必要があります。

また、障害が発生した場合には、その発生時期や原因を調べるためにログが必要になります。そのため、過去のログが必要になる場合も多いですので、ログを一定期間は保管しておく必要があります。

限界値

ファイルシステムの利用率や CPU の利用率が 100%になると、システムはそれ以上の処理ができなくなります。システムのプロセス数にも上限があり、プロセスが増えつづけると、新たなプロセスが生成できなくなります。こうした限界値に向かってデータが増加している場合には、定期的なファイルの削除や、プロセスの停止、リソースの追加などが必要であることを示しています。

図 7-3:ディスク利用率の推移

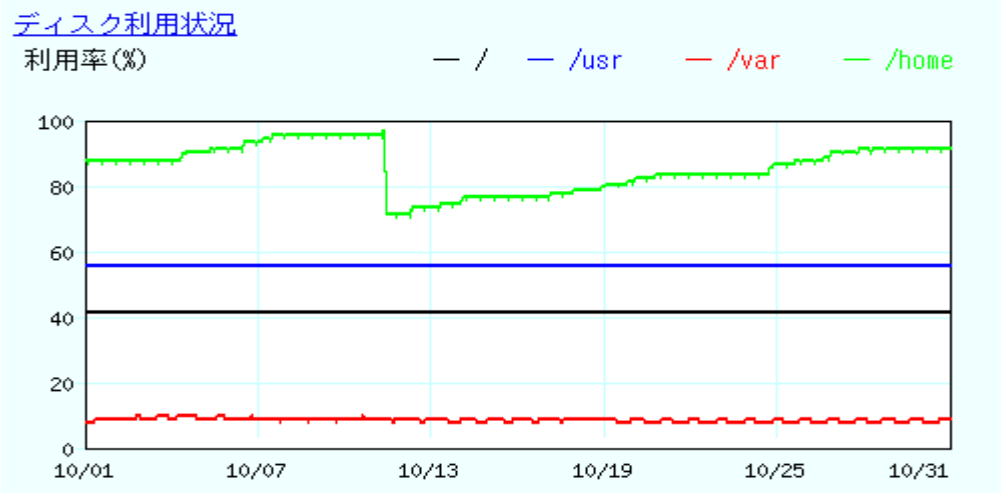


図 7-3 は、あるサーバのディスク利用率の推移を表しています。/home の利用率が、少しずつ増減を繰り返しながら 100%近くまで増加し、10/12 くらいに減少しています。そして、その後また増加が始まっています。一旦は、ファイルの削除でデータが減少しましたが、このまま放置しておけばファイルシステムがいっぱいになってしまう危険性があります。

状態管理で収集したデータをきちんと分析していれば、こうしたファイルシステムフルなどの問題は事前に解決することができます。

異常な増加

ほぼ一定の値を示しているデータが、突然増加しているような場合には、何らかの問題が発生した可能性があります。例えば、/var ファイルシステムの利用率が突然増加した場合には、ログが大量に出力されているのかもしれません。あるいは、プロセス数が突然増加した場合には、何らかのサービスの異常や、ハードウェアの異常を示している可能性もあります。

したがって、定常的なデータの突発的な増加には、何らかの対処が必要になります。

図 7-4:インタフェース利用状況

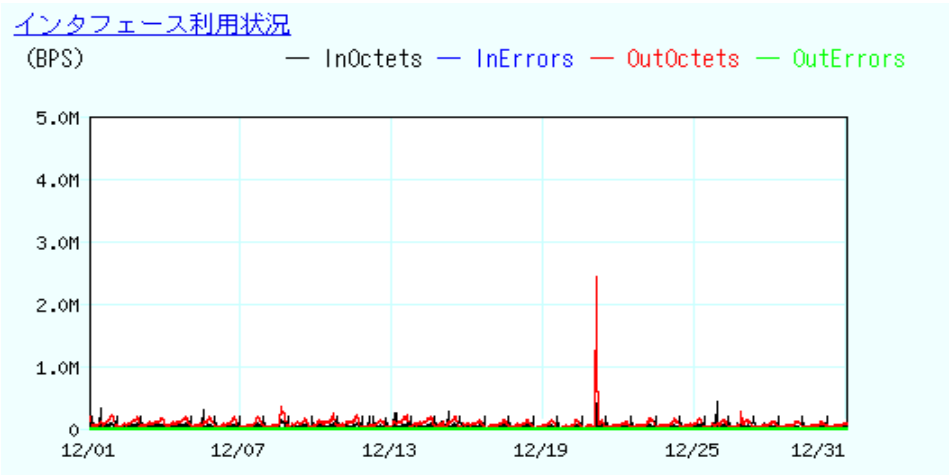


図 7-4 は、あるサーバの通信量のグラフです。12 月 21 日付近で、突発的に通信量が増えています。こうした現象は、外部からの攻撃を示す場合もあれば、ネットワークの異常を示す場合もありますので、ログなどを元に詳細に調査する必要があります。

異常な減少

反対に、定常的なデータが突然減少した場合にも、何らかの障害が起きている可能性があります。

図 7-5:空きメモリ（SWAP メモリ）の状況

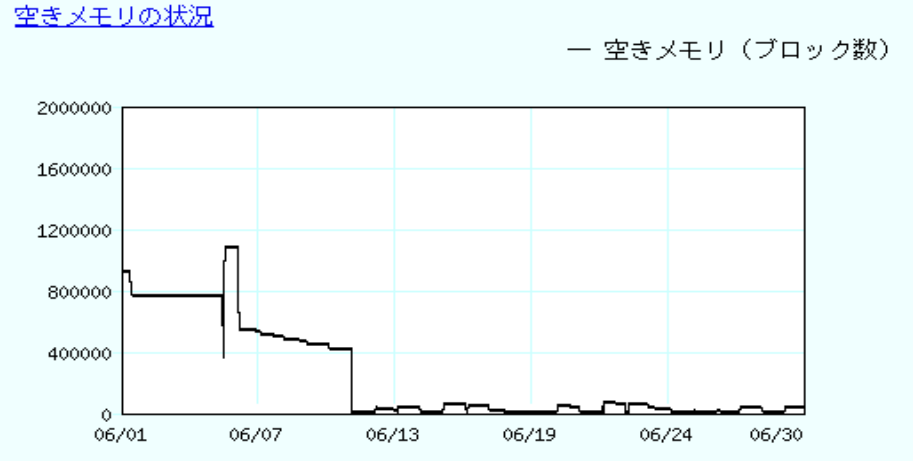


図 7-5 は、システムの SWAP メモリの空き状況を示したグラフです。

6 月 7 日から現象が始まり、6 月 12 日には激減しています。このような場合には、何らかの異常でメモリが余分に使われている可能性がありますので、サーバの状況を深く調べる必要があります。

明らかな傾向の変化

増加や減少だけでなく、データの傾向が明らかに変化することがあります。

図 7-6:インタフェース利用状況の明らかな変化

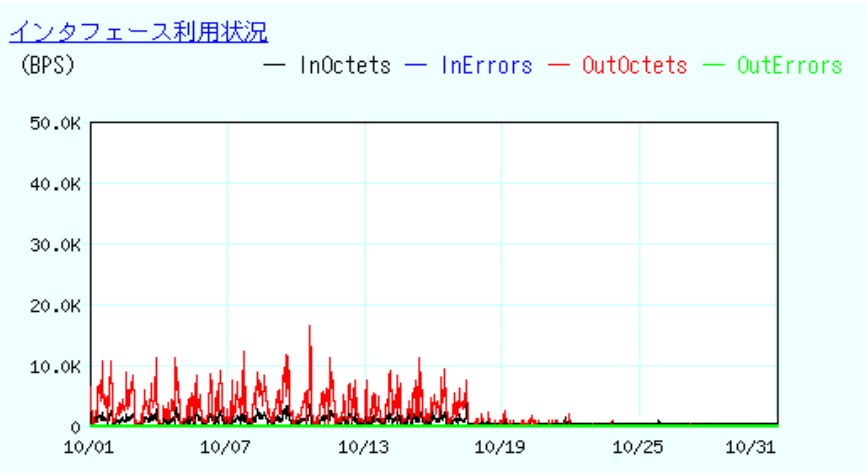


図 7-6 では、10 月中旬までは定期的な増減を繰り替えしているのに、10 月 18 日くらいからはほ

とんど通信がなくなってしまっています。こうした場合には、何らかのトラブルがあり、サーバへの通信が正常にできなくなっている可能性があります。

異常状態への対応

異常な状態に気がついたら、その一面的な状態だけではなく、ログやその他のデータを確認し、システム全体の状態をきちんと調査する必要があります。ファイルシステムの利用率が 100%になったという状態が、不正アクセスが原因で起こっているという場合もありますので、一面的なデータだけではなく、システム全体として状況を把握する必要があるのです。

何らかの原因が特定できる場合には、その原因を取り除いて、再発防止を行う必要があります。例えば、ログファイルが増加し続けていることに気がついた場合には、定期的にログのバックアップを取って、古いログは削除するような処理を組み込むことで、再発を予防することができます。このように、異常な状態を検出したら、きちんと再発を防止しておくことが非常に重要です。

7.2.3　Linuxでの実装

Linuxでは、カーネルの状態や様々なプロセスの状態を、/procにマウントされたprocfsを通じて取得することができます。次は/proc/meminfoを通じてシステムのメモリの状況を取得した例です。

▼*/proc/meminfoの取得例*

```
$ cat /proc/meminfo
MemTotal:      8169160 kB
MemFree:        491992 kB
Buffers:       1291636 kB
Cached:        5255336 kB
SWAPCached:          0 kB
Active:        4636120 kB
Inactive:      2175024 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:      8169160 kB
LowFree:        491992 kB
SWAPTotal:     8385920 kB
SWAPFree:      8385772 kB
Dirty:             524 kB
Writeback:           0 kB
AnonPages:      263804 kB
Mapped:          57920 kB
Slab:           796868 kB
PageTables:      23472 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:  12470500 kB
Committed_AS:  1817328 kB
```

```
VmallocTotal: 34359738367 kB
VmallocUsed:      5320 kB
VmallocChunk: 34359732855 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     2048 kB
```

7.3　sarコマンド

/proc/meminfo 以外の様々なファイルからも、システムの情報を取得することができます。しかし、こうした生の情報をすべて解釈するのは難しいため、Linuxでは用途に合わせて状態を取得するコマンドやユーティリティが用意されています。

sarコマンドは、代表的なシステム情報取得コマンドです。CPU、メモリ、ディスク、ネットワークなどの情報を定期的に記録し、出力します。また、10分毎に情報を記録しておき、それを表示することもできます。

7.3.1　CPUの利用状況

次は、sarコマンドでCPUの利用状況に関する統計を出力した例です。

▼*sarによるCPU情報の取得*

```
$ sar -u
Linux 2.6.18-194.3.1.el5 (alice01)  11/21/10

00:00:01          CPU     %user     %nice   %system   %iowait    %steal     %idle
00:10:01          all      0.37      0.00      0.19      0.61      0.00     98.83
00:20:01          all      0.38      0.00      0.18      0.71      0.00     98.73
00:30:01          all      0.30      0.00      0.17      0.61      0.00     98.93
00:40:01          all      0.36      0.00      0.20      0.65      0.00     98.79
00:50:01          all      0.31      0.00      0.18      0.59      0.00     98.92
01:00:01          all      0.32      0.00      0.18      0.59      0.00     98.91
01:10:01          all      0.32      0.00      0.19      0.54      0.00     98.95
   :
```

次のような指標を出力しています。

- %user　　通常のユーザプロセスがCPUを使っている時間の割合
- %nice　　優先度付きで実行されたユーザプロセスがCPUを使っている時間の割合
- %system　　カーネルやシステムプロセスがCPUを使っている時間の割合
- %iowait　　I/Oの待ち時間の割合
- %steal　　他のシステムのためにCPUが使えなかった時間の割合（仮想サーバで有効）
- %idle　　CPUの空き時間の割合

例えば、%iowaitが増加している場合には、システムはI/Oの性能がボトルネックになってきていることが分かります。

7.3.2　ディスクI/Oの情報

次は、sarでディスクI/Oに関する統計を出力した例です。

▼*sar によるディスク I/O 情報の取得*

```
$ sar -b
Linux 2.6.18-194.3.1.el5 (alice01)  11/21/10

00:00:01          tps      rtps      wtps   bread/s   bwrtn/s
00:10:01        18.25      0.17     18.08      6.00    279.10
00:20:01        17.06      0.01     17.06      0.04    273.25
00:30:01        15.10      0.01     15.09      0.08    236.04
00:40:01        23.44      0.01     23.43      0.08    396.13
00:50:01        15.79      0.00     15.78      0.04    253.45
01:00:01        16.45      0.00     16.45      0.04    245.72
01:10:01        17.29      0.00     17.29      0.00    262.41
   :
```

次のような指標を出力しています。

- tps　　1 秒あたりの合計転送量（Transfer per second）
- rtps　　1 秒あたりの読み込み側の転送量
- wtps　　1 秒あたりの書き込み側の転送量
- bread/s　　1 秒あたりのブロックデバイスからの読み込み量
- bwrtn/s　　1 秒あたりのブロックデバイスへの書き込み量

%iowait とディスク I/O の量を観察することで、I/O がシステムのボトルネックになっていないかを調べることができます。

7.3.3　メモリに関する情報

次は、メモリと SWAP に関する統計を出力した例です。

▼*sar によるメモリと SWAP に関する情報の取得*

```
$ sar -r
Linux 2.6.18-194.3.1.el5 (alice01)  11/21/10

00:00:01       kbmemfree  kbmemused    %memused  kbbuffers    kbcached  kbswpfree  kbswpused
%swpused  kbswpcad
00:10:01          367736    7801424       95.50    1299832    5363184    8385772          148
0.00         0
00:20:01          367444    7801716       95.50    1299936    5363256    8385772          148
0.00         0
00:30:01          368888    7800272       95.48    1300008    5363444    8385772          148
0.00         0
00:40:01          356864    7812296       95.63    1300128    5363672    8385772          148
0.00         0
00:50:01          363004    7806156       95.56    1300188    5363864    8385772          148
0.00         0
```

```
01:00:01          362880    7806280       95.56    1300224    5362520    8385772          148
0.00          0
01:10:01          367004    7802156       95.51    1300284    5359236    8385772          148
0.00          0
   :
```

次のような指標を出力しています。

- kbmemfree　　空きメモリ量(kb)
- kbmemused　　使用中のメモリ量(kb)
- %memused　　使用中のメモリの割合
- kbbuffers　　カーネルバッファとして利用されているメモリの量(kb)
- kbcached　　カーネルがデータキャッシュのために使用しているメモリ量(kb)
- kbswpfree　　SWAPの空き容量(kb)
- kbswpused　　SWAPの利用量(kb)
- %swpused　　SWAPの利用率
- kbSWAPcad　　SWAP中にキャッシュされているデータ量(kb)

Linuxは、システムの高速化のためにできるだけメモリを有効活用しようとします。空いているメモリは、ファイルのキャッシュなどに利用するのです。そのため、空きメモリ量を観察しても、システムのメモリが十分にあるかを判断することはできません。システムの空きメモリがなくなると、徐々にSWAPが利用されるようになるため、SWAPの利用率を観察するのが合理的です。

7.3.4　プロセススケジューリングに関する情報

次は、プロセスのスケジューリングに関する統計を出力した例です。

▼*sarによるプロセスのスケジューリングに関する情報の取得*

```
$ sar -q
Linux 2.6.18-194.3.1.el5 (alice01)  11/21/10

00:00:01      runq-sz  plist-sz   ldavg-1   ldavg-5  ldavg-15
00:10:01            6       318      0.00      0.00      0.00
00:20:01            7       318      0.06      0.07      0.01
00:30:01           11       323      0.00      0.00      0.00
00:40:01            7       327      0.00      0.00      0.00
00:50:01            6       315      0.00      0.01      0.00
01:00:01           11       318      0.09      0.04      0.01
   :
```

次のような指標を出力しています。

- runq-sz　　動作待ち状態になっているプロセスの数
- plist-sz　　プロセスとスレッドの総数
- ldavg-1　　最近1分間のロードアベレージ

- ldavg-5　　最近 5 分間のロードアベレージ
- ldavg-15　　最近 15 分間のロードアベレージ

動作待ち状態になっているプロセスの数が多くなっていれば、システムの処理が追いついていないことを表しています。ただ、この数値はその時間の瞬間的なものです。1 分間、5 分間、15 分間のロードアベレージから、システム全体の負荷を知ることができます。sar は、その他にも様々なシステムの統計情報を取得していて、表示することができます。

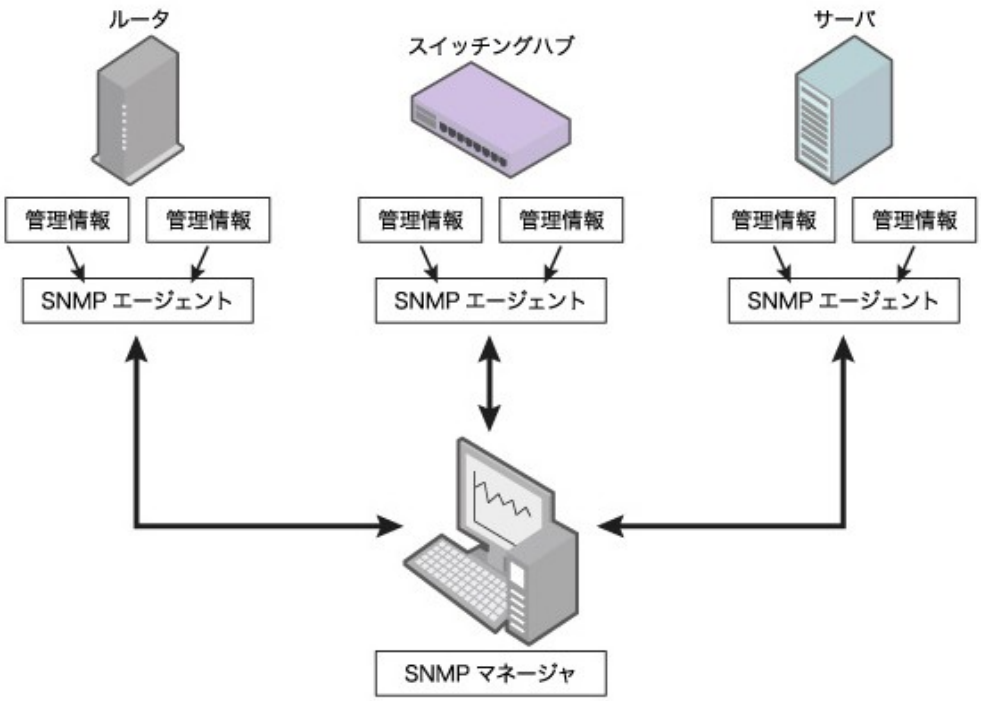
7.4 SNMP

sar は自システムの状況を定期的に記録し、必要に応じてそれを表示することができます。しかし、各サーバで個別にシステムの状況を管理するのではなく、ネットワーク内の様々なサーバを集中管理することができると非常に便利です。そのために使われるのが、SNMP（Simple Network Management Protocol）です。

7.4.1 SNMP の概要

SNMP には、複数のバージョンがあります。SNMP version 1 は、もっとも普及しているバージョンですが、32 ビットの数値までしか扱えないことや、認証が不十分だという問題を抱えています。これを解決するために、SNMP version 2 が提案されましたが、様々なネットワーク機器にすべての機能を実装することが難しく、標準化が途中で断念されました。ただ、当初提案されていた Community Base SNMP（v2c）、Party Base SNMP(v2p)、User Base SNMP(v2u)のうちの v2c の部分は、64 ビットの数値を扱えることなどから、比較的多くのネットワーク機器で採用されています。そして、その後、SNMP version 3 が考案されました。SNMP v3 では User-based Security Model というモデルを採用していますが、まだ広く使われるという段階には至っていません。そのため、SNMP v1, v2c がもっともよく使われています。

図 7-7:SNMP の構成



SNMP では、情報を提供する側を **SNMP エージェント**、情報を取得し管理する側を **SNMP マネー**

ジャと呼びます(図 7-7)。

MIB-II

SNMP で扱うことができる情報が機器によって違うと不便ですので、プロトコルとは切り離して MIB（Management Information Base）という規格で標準化が進められています。現在は、MIB-II（MIB Version 2）と呼ばれる規格が標準的に使われています。MIB-II では、管理する情報に OID と呼ばれる管理番号が付与されていてます。システム管理で利用する主な情報としては表 7-1 のようなものが定義されています。

▼*表 7-1:MIB-II のトップレベルカテゴリ*

カテゴリ	概要
system	ホストやルータの名称などのシステム情報
interfaces	個々のネットワークインタフェースに関する情報
at	ARP などのアドレス変換情報
ip	IP に関する情報
icmp	ICMP に関する情報
tcp	TCP に関する情報
udp	UDP に関する情報
egp	EGP（Exterior Gateway Protocol）に関する情報
snmp	SNMP に関する情報
private	製品特有の情報

MIB-II の OID は、実際には iso から始まる階層で管理されています。

```
+--iso(1)
   +--org(3)
      +--dod(6)
         +--internet(1)
            +--directory(1)
            +--mgmt(2)
            |  +--mib-2(1)
            |     +--system(1)
            :     :
            |     |  +--sysORTable(9)
            |     |     +--sysOREntry(1)
            |     |        |  Index: sysORIndex
            |     |        +-- ---- INTEGER   sysORIndex(1)
            |     |        |        Range: 1..2147483647
            |     |        +-- -R-- ObjID     sysORID(2)
            |     |        +-- -R-- String    sysORDescr(3)
```

また、各 OID で表現される対象（**オブジェクト**）を **MIB 変数**と呼びます。MIB 変数には、表 7-2 のように使用されるデータの形式に合わせて型が決められています。

▼*表 7-2:主な MIB 変数の型*

変数型	内容
COUNTER32	32 ビットの整数の値で、（回数のように）増加していく数値
COUNTER64	64 ビットの整数の値で、（回数のように）増加していく数値
GAUGE32	32 ビットの整数の値で、（温度のように）状態を表す数値
GAUGE64	64 ビットの整数の値で、（温度のように）状態を表す数値
INTEGER	整数の値（32 ビット）
IPADDRESS	IP アドレス
STRING	文字列
TIMETICKS	時間

システム管理で利用する OID

システムの管理には、表 7-3 のような OID がよく使われます。

▼*表 7-3:システム管理に利用される OID の例*

サブツリー	MIB 変数名	型	OID	説明
system	sysDescr	STRING	.1.3.6.1.2.1.1.1	該当機器のシス

				テムに関する説明
	sysObjectID	OID	.1.3.6.1.2.1.1.2	該当機器特有の OID
	sysUpTime	TIMETICKS	.1.3.6.1.2.1.1.3	起動してからの時間
	sysContact	STRING	.1.3.6.1.2.1.1.4	管理者のメールアドレス
	sysName	STRING	.1.3.6.1.2.1.1.5	システムの名称（FQDN）
	sysLocation	STRING	.1.3.6.1.2.1.1.6	システムの設置場所
interfaces	ifNumber	INTEGER	.1.3.6.1.2.1.2.1	インタフェースの数
	ifTable.ifEntry.ifDescr	STRING	.1.3.6.1.2.1.2.2.1.1.x	インデックスｘのインタフェース名（例: eth0）
	ifTable.ifEntry.ifType	INTEGER	.1.3.6.1.2.1.2.2.1.3.x	インデックスｘのインタフェースの種類（Ethernet は 6）
	ifTable.ifEntry.ifSpeed	GAUGE32	.1.3.6.1.2.1.2.2.1.5.x	インデックスｘのインタフェースの回線速度
	ifTable.ifEntry.ifAdminStatus	INTEGER	.1.3.6.1.2.1.2.2.1.7.x	インデックスｘのインタフェースの設定状態。アップリンク（1），ダウンリンク（2），テスト中（3）
	ifTable.ifEntry.ifLastChange	TIMETICKS	.1.3.6.1.2.1.2.2.1.9.x	インデックスｘのインタフェースが現在の状態になった時間
	ifTable.ifEntry.ifInOctets	COUNTER32	.1.3.6.1.2.1.2.2.1.10.x	インデックスｘのインタフェースがこれまでに受け取ったデータの総バイト数
	ifTable.ifEntry.ifOutOctets	COUNTER32	.1.3.6.1.2.1.2.2.1.16.x	インデックスｘのインタフェースでこれまでに送信したデータ

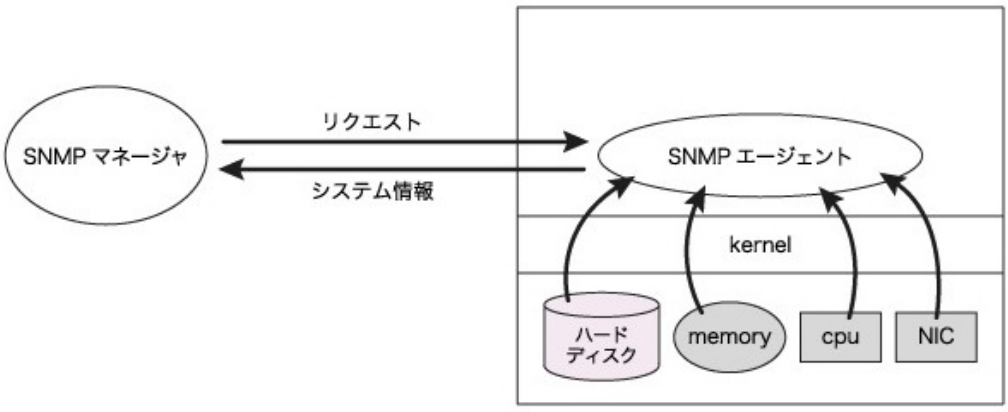
				の総バイト数
	ifTable.ifEntry.ifOutErrors	COUNTER32	.1.3.6.1.2.1.2.2.1.14.x	インデックスｘのインタフェースでこれまでに出力エラーになったパケットの総数
ucdavis	prTable.prEntry.prIndex	INTEGER	.1.3.6.1.4.1.2021.2.1.1	プロセス監視テーブルのインデックス（以下のｐ）
	prTable.prEntry.prNames	STRING	.1.3.6.1.4.1.2021.2.1.2.p	監視対象プロセスの名称
	prTable.prEntry.prCount	STRING	.1.3.6.1.4.1.2021.2.1.5.p	監視対象プロセスの数
	prTable.prEntry.prErrMessage	STRING	.3.6.1.4.1.2021.2.1.101.p	監視対象プロセスの個数が異常の場合のエラーメッセージ
	dskTable.dskEntry.dskIndex	INTEGER	.1.3.6.1.4.1.2021.9.1.1	ディスク監視テーブルのインデックス（以下のｄ）
	dskTable.dskEntry.dskPath	STRING	.1.3.6.1.4.1.2021.9.1.2.d	監視対象ディスクｄのパス
	dskTable.dskEntry.dskDevice	STRING	.1.3.6.1.4.1.2021.9.1.3.d	監視対象ディスクｄのデバイス名
	dskTable.dskEntry.dskTotal	INTEGER	.1.3.6.1.4.1.2021.9.1.6.d	監視対象ディスクｄのトータルバイト数
	dskTable.dskEntry.dskAvail	INTEGER	.1.3.6.1.4.1.2021.9.1.7.d	監視対象ディスクｄの空きバイト数
	dskTable.dskEntry.dskUsed	INTEGER	.1.3.6.1.4.1.2021.9.1.8.d	監視対象ディスクｄの使用バイト数
	dskTable.dskEntry.dskPercent	INTEGER	.1.3.6.1.4.1.2021.9.1.9.d	監視対象ディスクｄの利用率（%）
	dskTable.dskEntry.dskErrorFlag	INTEGER	.1.3.6.1.4.1.2021.9.1.100.d	監視対象ディスクｄに指定した残り容量があるか （0: 正常 1:異

				常）
	dskTable.dskEntry.dskErrorMsg	STRING	.1.3.6.1.4.1.2021.9.1.101.d	監視対象ディスクｄの残り容量が異常の場合のエラーメッセージ
	laTable.laIndex	INTEGER	.1.3.6.1.4.1.2021.10.1.1.1	システムのロードアベレージ情報のインデックス（以下のｘ）
	laTable.laLoad	INTEGER	.1.3.6.1.4.1.2021.10.1.2.2.x	システムのロードアベレージ情報ｘの値
	laTable.laErrorFlag	INTEGER	.1.3.6.1.4.1.2021.10.1.100.x	システムのロードアベレージが指定値内か（0: 正常 1:異常）
	laTable.laErrMessage	STRING	.1.3.6.1.4.1.2021.10.1.101.x	システムのロードアベレージが異常の場合のエラーメッセージ
	memory.memTotalSWAP	INTEGER	.1.3.6.1.4.1.2021.4.3	システムの SWAP メモリサイズ
	memory.memAvailSWAP	INTEGER	.1.3.6.1.4.1.2021.4.4	システムの空 SWAP メモリサイズ
	memory.memTotalReal	INTEGER	.1.3.6.1.4.1.2021.4.5	システムの実メモリサイズ
	memory.memAvailReal	INTEGER	.1.3.6.1.4.1.2021.4.6	システムの空メモリサイズ

7.4.2 SNMP エージェント

SNMP により情報を取得するためには、対象の機器やサーバに SNMP エージェントが導入されている必要があります。SNMP エージェントは、SNMP マネージャからシステム情報に関するリクエストを受け取ると、適切なシステム情報を収集しそれを返します(図 7-8)。

図 7-8:SNMPエージェントの動作イメージ



Linuxでの実装

Linuxでの SNMPエージェントの実装としては、NET-SNMPが使われています。NET-SNMPを利用できるようにするためには、次のような設定を行う必要があります。

- 情報を取得するSNMPマネージャのアクセス情報を設定します。
- 自サーバの情報のうち、管理者などのユーザが定義すべき情報を設定します。
- ファイルシステム、プロセスなどの拡張管理対象を設定します。

アクセス情報の設定

ほとんどのシステムでは、SNMPエージェントはTCP Wrapperの管理対象になっていますので、/etc/hosts.allow, /etc/hosts.denyを適切に設定します。

▼*SNMPエージェントのアクセス制御（/etc/hosts.deny）*

```
snmpd: ALL
```

▼*SNMPエージェントのアクセス制御（/etc/hosts.allow）*

```
snmpd: 127.0.0.1  192.168.10.1
```

この例では、自サーバ内（127.0.0.1）からと、SNMPマネージャ（192.168.10.1）のみのアクセスを許可しています。また、アクセス情報の設定はSNMPエージェントの設定ファイルにも行う必要があります。

▼*SNMPエージェントのアクセス制御（/etc/snmp/snmpd.conf）*

```
com2sec snmpUser  default       public                          ←  (1)
```

```
group   snmpGroup v1           snmpUser                                ←  (2)
group   snmpGroup v2c           snmpUser

view    all           included   .1      80                           ←  (3)

access  snmpGroup ""      any       noauth    exact  all none none  ← (4)
```

SNMP エージェントへアクセスするためには、SNMP マネージャに対応したユーザと**コミュニティ**と呼ばれるパスワードが必要です。(1)の「com2sec」では snmpUser というユーザを定義し、使うことのできる SNMP マネージャと**コミュニティ**文字列を定義します。また、セキュリティグループを定義し、そのセキュリティグループで使うプロトコル、利用可能ユーザを登録します。(2)では、snmpGroup というグループを定義し、v1, v2c を snmpUser が利用できるように設定しています。また、(3)ではアクセスできる情報の範囲を定義しています。ここでは、all を定義し、すべての OID を参照できるようにしています。そして、(4)では snmpGroup に属するユーザが all を参照するための設定をしています。

自サーバの情報の設定

自サーバの情報として、管理者、サーバの場所などを設定します。

▼*自サーバの設定（/etc/snmp/snmpd.conf）*

```
syslocation  Rack #1 in 1F Machine room
syscontact admin <admin@designet.jp>
```

syslocation にはこのコンピュータの置き場所を定義し、syscontact には管理者の情報を定義します。

拡張管理対象の設定

NET-SNMP では Linux のシステム管理を容易にするため、独自の MIB 変数をサポートし、ucdavis サブツリーとして提供しています。それにより、ファイルシステム、ロードアベレージ、メモリの利用状況、プロセスの稼働状況を知ることができます。

次は、その設定例です。

▼*拡張管理情報の設定（/etc/snmp/snmpd.conf）*

```
proc sendmail 20 1                  ← sendmail プロセスが最大 20 個、最小 1 個存在する
disk /    10%                       ← /ファイルシステムが 10%以上空いている
disk /var 10%                       ← /var ファイルシステムが 10%以上空いている
disk /home 10%                      ← /home ファイルシステムが 10%以上空いている
load 12 12 12                       ← 1、5、10 分平均のロードアベレージが 12 以下である
```

拡張情報では、この例のように、システムにとって正しい状態を設定しておきます。この設定によって、ucdavis の各 OID でプロセス、ディスク、ロードアベレージなどの情報が取得できるようになります。この設定範囲から外れると、エラーフラグがセットされます。このエラーフラグを外部の SNMP マネージャから監視することで、プロセスの稼働状況、ディスク容量、ロードアベレージなどを管理することができます。

SNMP ユーティリティ

NET-SNMP には、SNMP エージェントから値を取得して表示するユーティリティプログラムが用意されています。snmpget は、指定した OID から値を取得します。

▼*snmpget の実行例*

```
$ snmpget -v1 127.0.0.1 -c public system.sysName.0
SNMPv2-MIB::sysName.0 = STRING: centos5
```

引数の「-v1」は SNMP バージョンの指定、「127.0.0.1」は SNMP エージェントの指定、「-c public」はコミュニティの指定です。また、snmpwalk は、指定した OID ツリーにあるすべての MIB 変数を表示します。

▼*snmpwalk の実行例*

```
$ snmpwalk -v1 localhost -c public ucdavis.prTable
UCD-SNMP-MIB::prIndex.1 = INTEGER: 1
UCD-SNMP-MIB::prNames.1 = STRING: sendmail
UCD-SNMP-MIB::prMin.1 = INTEGER: 1
UCD-SNMP-MIB::prMax.1 = INTEGER: 20
UCD-SNMP-MIB::prCount.1 = INTEGER: 2
UCD-SNMP-MIB::prErrorFlag.1 = INTEGER: 0
UCD-SNMP-MIB::prErrMessage.1 = STRING:
UCD-SNMP-MIB::prErrFix.1 = INTEGER: 0
UCD-SNMP-MIB::prErrFixCmd.1 = STRING:
```

7.4.3　SNMP マネージャ

SNMP マネージャは、ネットワーク上の様々な機器やサーバの SNMP エージェントから、システムの情報を取得して管理するソフトウェアです。取得したデータは、データベースや専用のファイルなどに保管します。また、グラフや表などの視覚的に分かりやすいイメージにして表示します(図 7-9)。

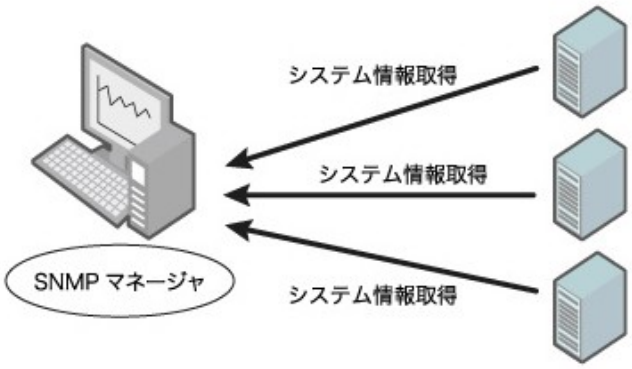


図 7-9:SNMP マネージャの動作イメージ

Linux での実装

Linux の SNMP マネージャの実装としては、非常に多くのソフトウェアがリリースされています。中でも MRTG、Cacti、Zabbix などが有名です。また、SNMP マネージャの機能だけでなく、ログ管理やサービス監視などの管理機能を一括して導入することのできるソフトウェアもあります。Hinemos はそうしたソフトウェアで、日本の NTT データが開発してオープンソースソフトウェアとして公開しています。

MRTG

MRTG は非常に古くからある SNMP マネージャの実装です。SNMP エージェントから取得した情報を保管し、グラフで表記し、HTML 形式のページとして出力します。図 7-10 は、MRTG のページの例です。標準では、日、週、月の 3 つのグラフを作成します。

MRTG の最大の特徴は、cfgmaker というコマンドラインの設定ツールを利用して、簡単にこうしたグラフを作成できることです。次は、その実行例ですが、データを取得する SNMP エージェントのコミュニティと IP アドレスだけを引数で設定しています。

▼*cfgmaker の実行例*

```
# cfgmaker public@127.0.0.1
```

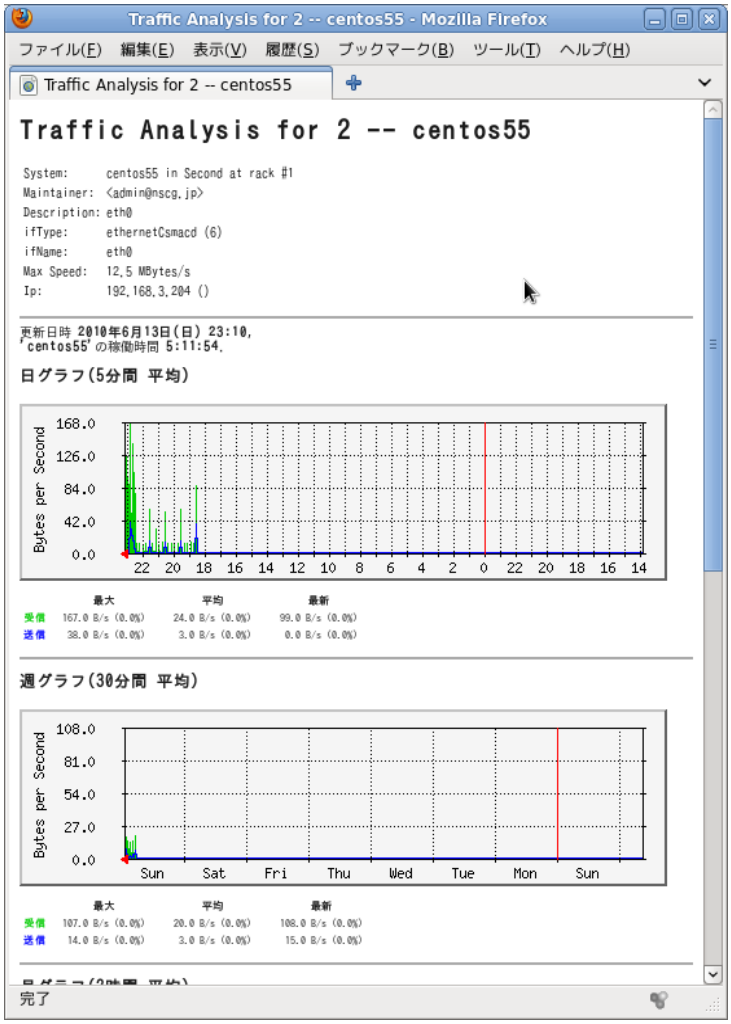
このコマンドを実行するだけで、自動的に対象機器のネットワークインタフェースを調査し、各インタフェースのトラフィック状況のデータを収集し、グラフを作成するための設定ファイルを作成することができます。また、OID を指定すれば、トラフィックだけでなく、様々なグラフを作成することができます。

MRTG は非常に便利ではありますが、データ収集毎(標準では 5 分)にグラフを再生成します。グラフを生成するための処理の負荷が高いため、データを取得する対象数が増加すると十分にデータを収集することができなくなってしまうという欠点があります。

MRTG では、こうした欠点を克服するため、データの保管と表示を RRDtool で行うように構成することができます。RRDtool は、ラウンドロビンという形式のデータベースに数値を蓄積し、グラフを作

成するツールです。RRDtoolと連携すると、管理者がデータを閲覧した時にグラフが生成されるようになるため、より多くの対象を管理することができるようになります。

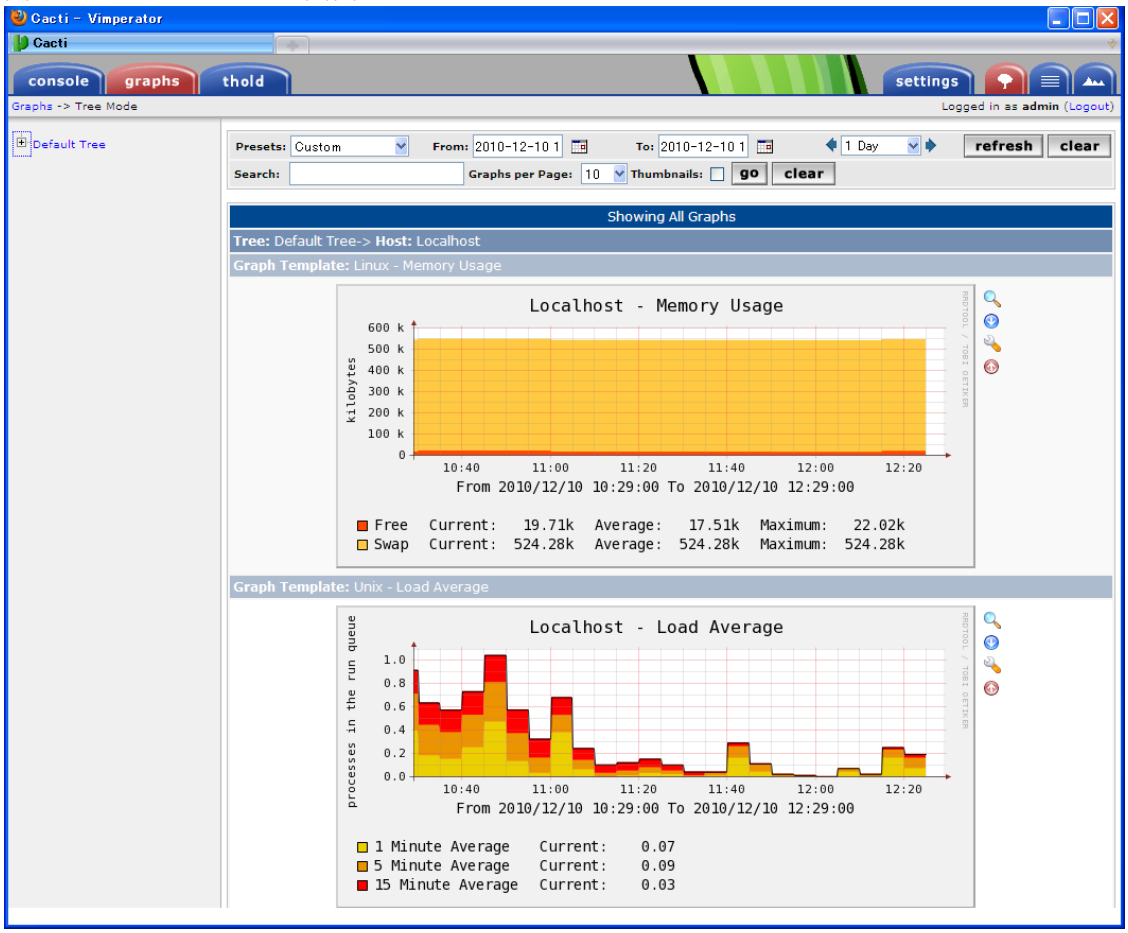
図 7-10:MRTGのグラフ



Cacti

Cactiは、RRDtoolを使ったSNMPマネージャです。Cactiでは、SNMPエージェントの登録、グラフの設定、収集すべきデータの登録などをWebベースで管理することができます(図7-11)。MRTGに比べて、高速に動作することも可能です。収集したデータは、MySQLなどのデータベースに格納することも可能です。

図 7-11:Cacti のグラフ画面



Cacti は、様々なプラグインを導入することができるように設計されています。プラグインを導入することで、この後解説するサービス監視システムや統合監視ツールとして利用することも可能です。

7.5　サービス監視システム

システムの状態管理とは別に、正常にシステムやサービスが動作しているかを定期的に調べて管理するのがサービス監視ツールです。

7.5.1　サービス監視システムの概要

SNMPを使って、ネットワーク上の様々な機器の状態を取得し、ネットワーク全体の性能低下や障害を監視する仕組みを**ネットワーク監視システム**と呼びます。ネットワーク監視システムでは、障害を検知すると電子メール、ランプ、電話などの方法を使って、システム管理者に異常を通知する**障害通知機能**も提供されます。

これをさらに発展させて、ネットワークの性能低下だけでなく、サービスのレベルでの性能や稼働状態までを管理しようとするのが**サービス監視システム**です。近年は、SLA（Service Level Agreement）が重視される傾向にあり、サービス監視システムが注目されています。

7.5.2　Linuxでの実装（Nagios）

サービス監視システムのLinuxでの実装として、Nagiosが使われています(図 7-12,13)。Nagiosは、次のような機能を提供します。

- SNMPによるシステム状態のモニタリング
- サービス状態のモニタリング
- ユーザ定義のモニタリング
- ログの統括管理
- メールやSMSによる障害通知
- 障害の履歴（**インシデント**）管理

図 7-12:Nagios のホストの状態表示画面

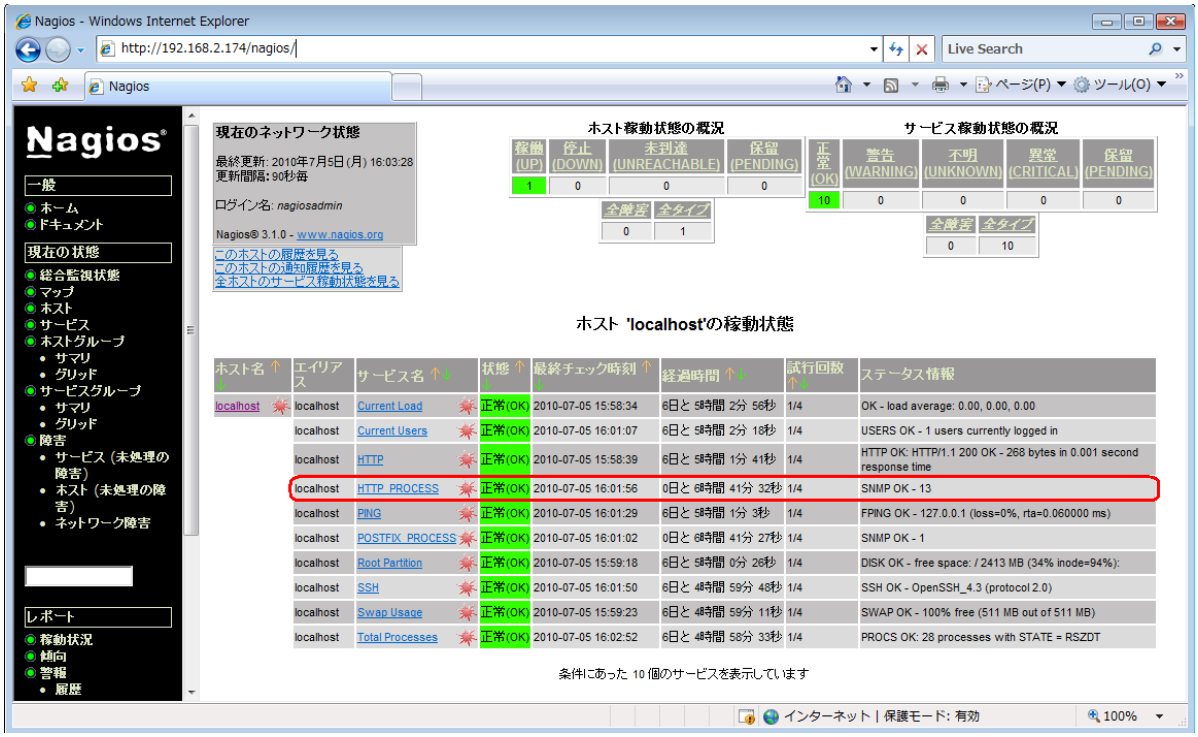
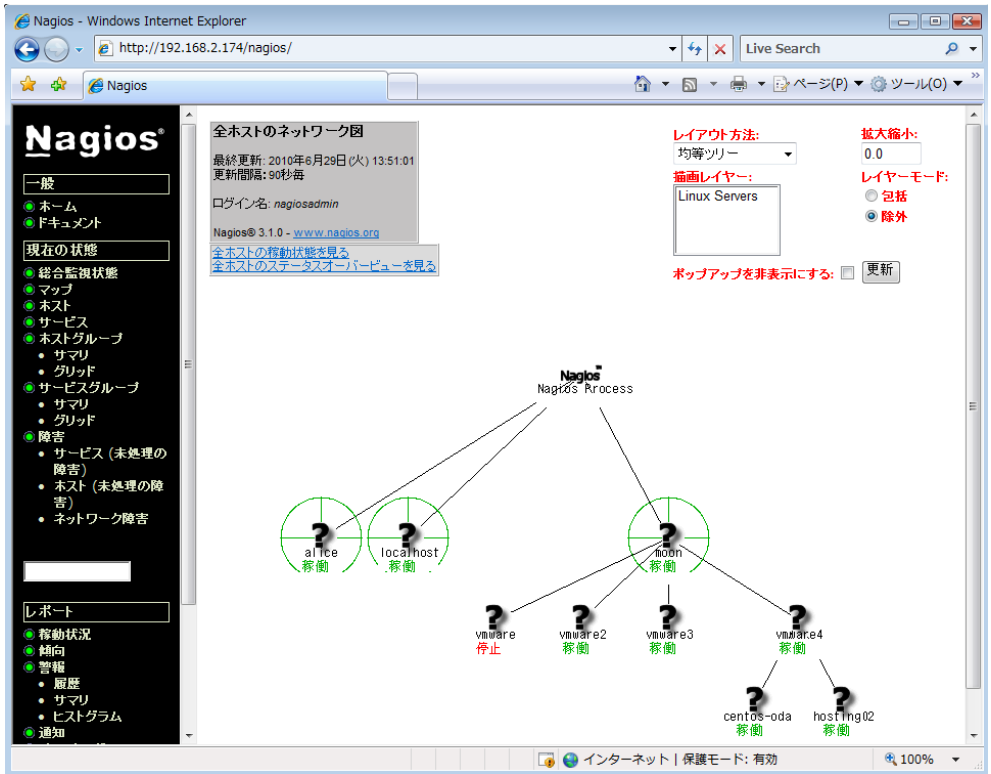


図 7-13:Nagios の経路図表示



7.6 統合監視ツール

MRTG や Cacti は SNMP マネージャとして動作し、グラフなどを利用してシステム状態を管理することができます。また、Nagios などのサービス監視システムでは、サービスの動作状況をモニタリングし、障害通知までを行うことができます。

このような状態管理から、サービス監視、障害通知までのすべての機能を 1 つのアプリケーションで提供するのが統合管理ツールです。

7.6.1 Linuxでの実装

Linux で利用できる統合監視ツールの実装としては、Zabbix や Hinemos があります。どちらのソフトウェアも、状態管理、サービス監視、障害通知までをサポートします。

Zabbix

Zabbix は、Web インタフェースから利用することのできる統合管理ツールです(図 7-14,15)。次のような特徴があります。

- 状態管理機能
  - SQL データベースにデータを保管します
  - 必要に応じてグラフを生成することができます
  - SNMP マネージャとして動作することもでき、SNMPv1、v2、v3 をサポートしています
- サービス監視機能
  - Unix、Linux、BSD、Windows(Win32)、MacOS X、NetWare など、幅広い OS を管理することができます。
  - SNMP エージェントだけでなく、各 OS で動作する独自のエージェントも提供します。
  - ユーザ定義の監視スクリプトを利用することができます。
- 障害通知機能
  - 障害を検知し、メールで通知する機能を備えています。
  - 障害時に特定のプログラムを自動的に実行することができます。
  - 障害の項目、通知先などによって、詳細な通知条件の設定ができます。
- 管理画面
  - Web 画面から管理を行うことができます。
  - ホストマップなどにより、視覚的な管理が可能です。

図 7-14:Zabbix のカスタムグラフ画面

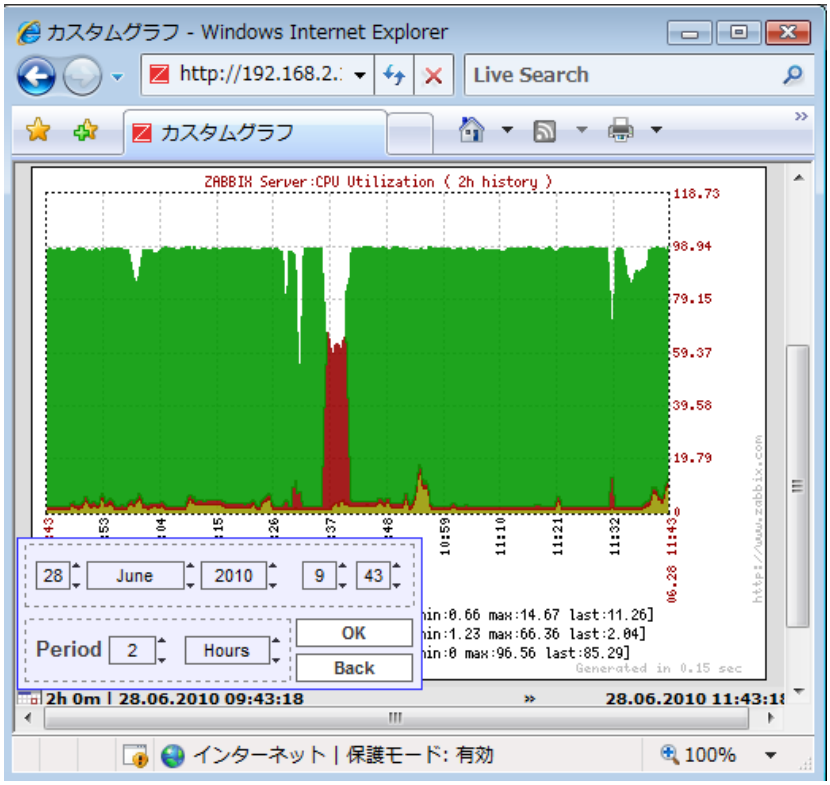
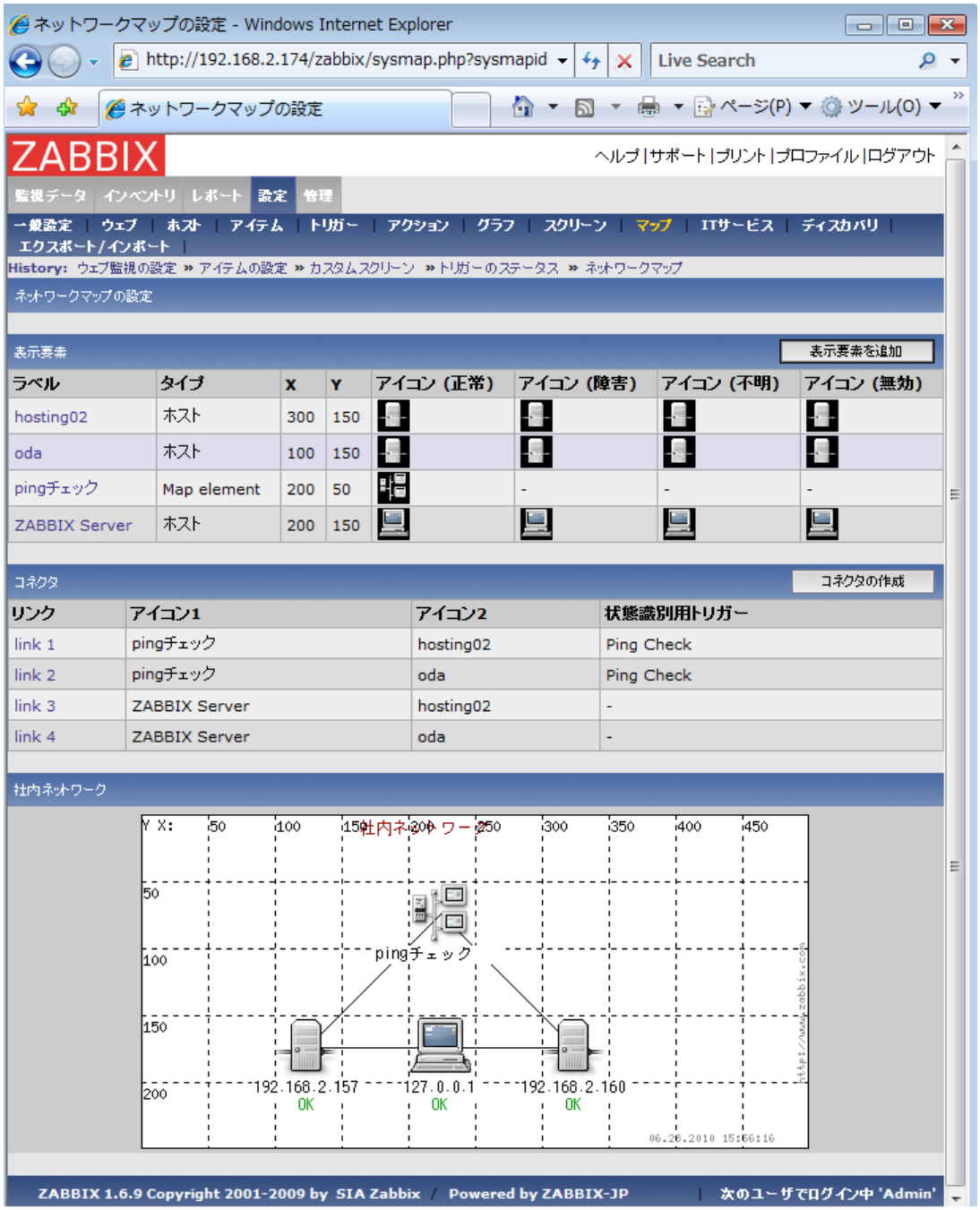


図 7-15:Zabbix の経路図作成画面



Hinemos

Hinemos は、NTT データが提供する統合管理ツールで、オープンソースソフトウェアとしても配布されています。Hinemos では、管理対象をグループ化して管理できるのが特徴です(図 7-16, 17)。それ以外にも、次のような特徴があります。

- 監視対象

    - 監視対象をレポジトリとして管理します。
    - 監視対象をグループ化して管理することができ、監視ルールの設定が容易に行えます。
    - 監視対象を階層化して管理することができるため、障害の影響までを管理することができます。
- 状態管理
    - SNMP エージェントや専用のエージェントを使って、CPU、メモリ、ディスク、ネットワークなどのリソースの状態を管理することができます。
    - イベントログやステータス情報を集中監視することができます。
- ジョブ管理
    - ユーザ定義のジョブを定義し、複数の監視対象で動作するように設定できます。
- 変更管理
    - パッチの適用、サーバの再起動などを集中管理できます。

図 7-16:Hinemos の監視管理機能画面（http://www.hinemos.info/より）

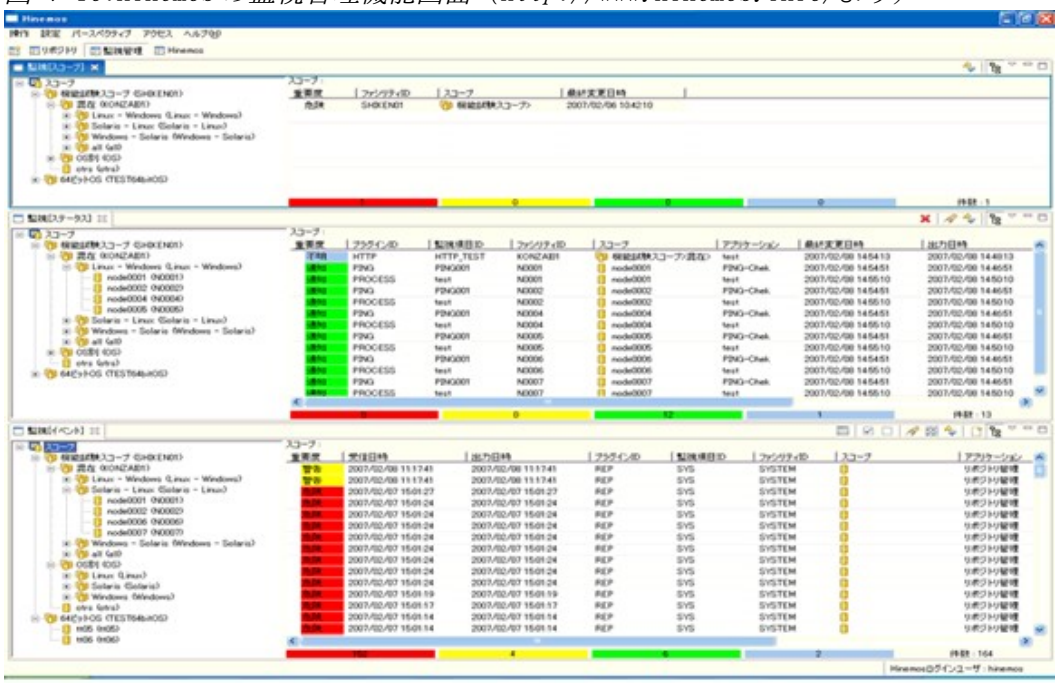
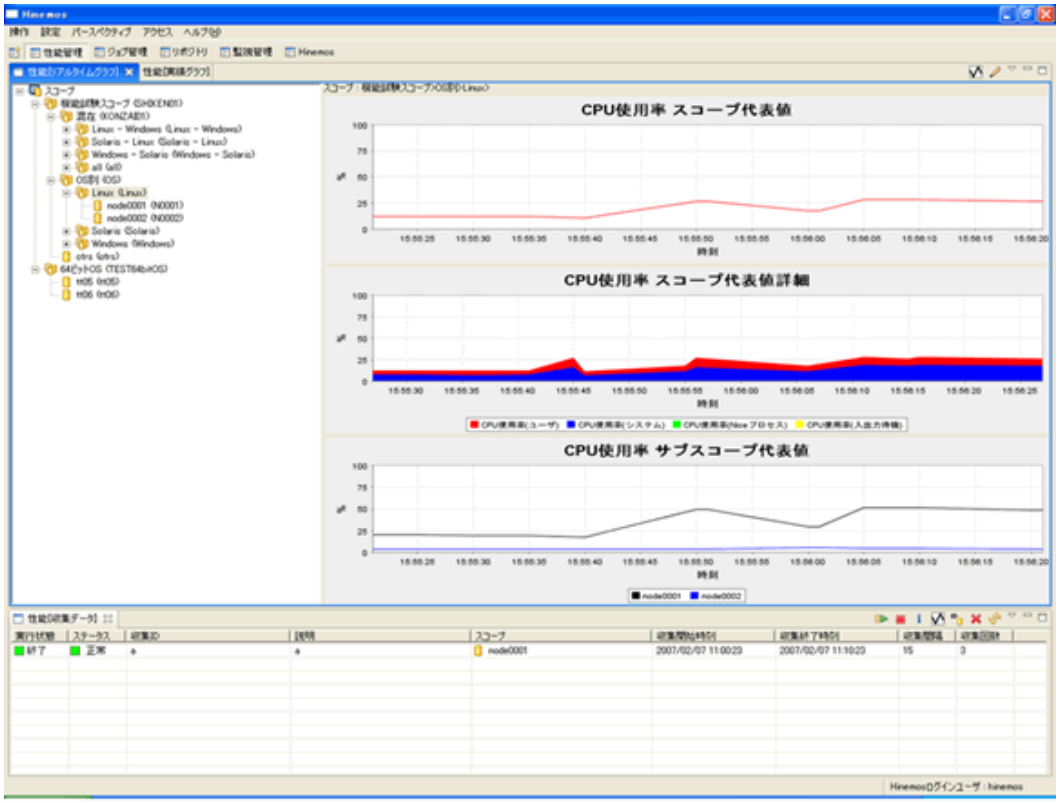


図 7-17:Hinemos の性能管理機能画面



Hinemos では、マネージャが Linux で稼働します。ただし、管理画面（クライアント）は Windows アプリケーションとして提供されます。また、専用エージェントは、RedHat Enterprise Linux 4/5、Windows 2000 Advanced Server、Windows Server 2003/2008 などの環境に対して提供されます。

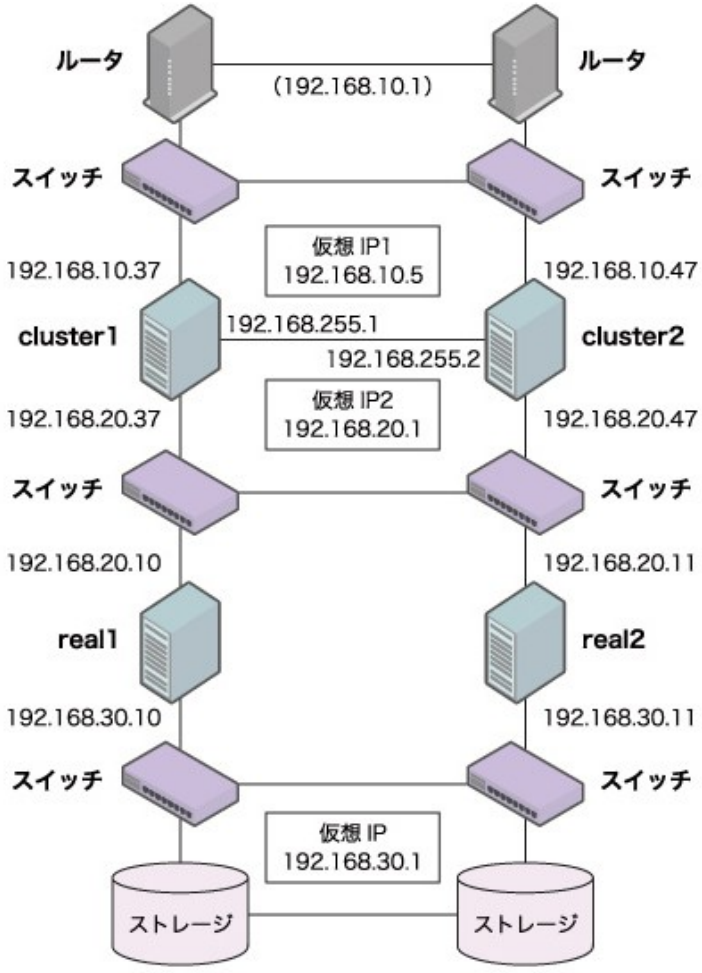
8章 ロードシェアリングによるシステムの構築

本章では、第 3 章で学習したロードシェアリングの仕組みと第 4 章で学習した共有ディスクを使い、実際にシステムを構築する事例を取り上げます。

8.1　システムの概要

ここでは、Web サーバ 2 台をロードバランサを使って冗長化する構成を事例として取り上げます。図 8-1 のように、システムに単独障害点がないように、ルータや SW も含めてすべて冗長化されています。cluster1, cluster2 はロードバランサとして構成しますが、片方が止まってもシステムが停止しないように、heartbeat を使って冗長化します。また、real1, real2 は Web サーバとして構成します。Web サーバのデータはネットワーク上のストレージに配置します。

図 8-1:Web サーバのロードバランシングシステムの構成例



8.2　ロードバランサの構築

ロードバランサの cluster1、cluster2 には次のような設定を行う必要があります。

- カーネルパラメータを設定する
- heartbeat と ldirectord(heartbeat 付属)をインストールする
- heartbeat の設定を行う
- ldirectord の設定を行う
- heartbeat の起動を行う

8.2.1　カーネルパラメータの設定

ロードバランサとして動作するためには、パケットを受け取りルーティングを行う機能を有効にしておく必要があります。

▼*/etc/sysctl.conf (cluster1, cluster2)*

```
net.ipv4.ip_forward = 1
```

8.2.2　ソフトウェアのインストール

heartbeat は、Debian、Fedora、CentOS、OpenSUSE などではパッケージも提供されています。コンパイル・インストールする場合には、heartbeat は The High Availability Linux Project のサイト(http://linux-ha.org)からダウンロードすることができます。また、heartbeat では、libnet というライブラリを使います。こちらもサイト(http://packetfactory.openwall.net/projects/libnet/)から入手する必要があります。

8.2.3　クラスタの設定

クラスタの設定は、/etc/ha.d/ha.cf で行います。最初に基本設定を行います。cluster1, cluster2 の両方のノードで、監視時間などのパラーメータは同じでなければなりません。ただし、ucast の相手 IP アドレスは必ず異なります。

▼*/etc/ha.d/ha.cf (cluster1)*

```
logfile /var/log/ha.log                    ←  ログファイル

keepalive 2                                ←  監視間隔
deadtime 30                                ←  ダウンと判定する時間
warntime 10                                ←  ログに記録するまでの時間
initdead 120                               ←  起動後、監視開始までの時間

udpport 694                                ←  LAN 監視のポート番号
ucast eth1 192.168.255.2                   ←  LAN 監視に使うポートと相手
baud 19200                                 ←  シリアルの通信速度
```

```
serial /dev/ttyS0                              ←　シリアルポートのデバイス

auto_failback on                               ←　自動フェイルバック（有効）

watchdog /dev/watchdog                         ←　watchdogデバイス

node cluster1 cluster2                         ←　クラスタ構成ノード名

respawn root /usr/local/bin/check_network      ←　サービス監視スクリプト
```

▼*/etc/ha.d/ha.cf（cluster2）*

```
logfile /var/log/ha.log

keepalive 2
deadtime 30
warntime 10
initdead 120

udpport 694
ucast eth1 192.168.255.1                       ←　ここだけが違う
baud 19200
serial /dev/ttyS0

auto_failback on

watchdog /dev/watchdog

node cluster1 cluster2

respawn root /usr/local/bin/check_network
```

次に、各ノード間の認証設定を行います。設定は、/etc/ha.d/authkeysで行います。次のように、ファイルを作成してください。両ノードとも同じ設定でなければなりません。

▼*/etc/ha.d/authkeys（cluster1, cluster2）*

```
auth 1
1 crc
```

このファイルは、heartbeatの管理ユーザしかアクセスできないようにしておく必要があります。

▼*管理ユーザしかアクセスできないようにする*

```
# chown hacluster:haclient /etc/ha.d/authkeys
# chmod 600 /etc/ha.d/authkeys
```

さらに、各ノードの/etc/ha.d/haresourcesに共有リソースの設定を行います。この設定ファイルには、先頭に稼働系サーバの名前を記載し、その後ろにリソースを列挙します。ldirectordと代表IPアドレスをリソースとして設定します。この設定も両ノードで同じでなければなりません。

▼*/etc/ha.d/haresources（cluster1, cluster2）*

```
cluster1 IPaddr::192.168.10.5/24 IPaddr::192.168.20.1/24 ldirectord
```

また、サービスが正常に動作しているかを監視するため、サービス監視スクリプトを配置する必要があります。このファイルは、必要に応じて自分で作成する必要があります。次の例は、インタフェースのリンクを確認するとともに、ldirectordのステータスを確認するものです。

▼*サービス監視スクリプト /usr/local/bin/check_network(cluster1, cluster2)*

```
#! /bin/sh
#
# check_network: リンク状態と通信状態を確認する
#
INTERVAL=5                #  監視間隔
CVIP=192.168.10.5         #  仮想IP
IFACES="eth0 eth2"           #  インタフェース名
GATEWAY=192.168.10.1      #  ゲートウェイ

#
# 監視ループ
#
while :
do
        sleep ${INTERVAL}

        #
        # 仮想IPアドレスがこのサーバについているかを確認
        #
        /etc/ha.d/resource.d/IPaddr ${CVIP} status > /dev/null 2>&1
        if [ $? -ne 0 ]
        then
                continue
        fi

        #
        # インタフェースのLinkを確認
        #
        for IF in ${IFACES}
        do
                LINK=`/sbin/ethtool ${IF} | awk '/Link detected: /{print $3;}'`
                if [ "$LINK" != "yes" ]
                then
```

```
                        /usr/lib/heartbeat/heartbeat -k
                        exit 100
                fi
        done

        #
        # pingによる通信確認
        #
        ping -c 1 -w 1 ${GATEWAY} > /dev/null 2>&1
        if [ $? -ne 0 ]
        then
                /usr/lib/heartbeat/heartbeat -k
                exit 101
        fi

        #
        # ldirectordが動作しているかを確認
        #
        /etc/ha.d/resource.d/ldirectord status > /dev/null 2>&1
        if [ $? -ne 0 ]
        then
                /usr/lib/heartbeat/heartbeat -k
                exit 101
        fi
done
```

8.2.4　ldirectordの設定

ldirectordはheartbeatとともにインストールされています。ldirectordの設定は、/etc/ha.d/ldirectord.cfで行います。

▼*/etc/ha.d/ldirectord.cf の設定例*

```
virtual=192.168.10.5:80                                          ← (1)
        real=192.168.20.10:80     masq                           ← (2)
        real=192.168.20.11:80     masq
        scheduler=rr                                             ← (3)
        protocol=tcp                                             ← (4)
        service=http                                             ← (5)
        request="check.html"                                     ← (6)
        receive="This server is active."                         ← (7)
```

（1）は代表 IP アドレス・ポートの設定です。（2）では実サーバが real1, real2 であることを設定しています。（3）は、振り分けスケジューリングの設定です。この例では、ラウンドロビンにしています。（5）～（7）はサービスの稼働チェックの設定です。この例では、check.html というファイルを参照し、内容が「This server is active.」となっていることを確認します。

8.2.5　heartbeat の起動

設定が完了しましたら、cluster1, cluster2 で heartbeat を起動します。うまく設定が行われていれば、heartbeat が順次リソースを起動します。

▼*heartbeat の起動（cluster1, cluster2）*

```
# /etc/init.d/heartbeat start
Starting High-Availability services:
                                                        [  OK  ]
```

しばらくすると、cluster1 に代表 IP アドレスが設定され、ldirecotrd が起動されます。

▼*代表 IP アドレスが付与されているかを確認*

```
cluster1# ifconfig eth0:0
eth0:0    Link encap:Ethernet  HWaddr 00:0C:29:90:1F:AB
          inet addr:192.168.10.5  Bcast:192.168.10.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:18 Base address:0x1480

cluster1# ifconfig eth2:0
eth2:0    Link encap:Ethernet  HWaddr 00:0C:29:90:1F:A1
          inet addr:192.168.20.1  Bcast:192.168.20.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:17 Base address:0x1400
```

ldirectord の設定が正しく行えたかどうかは、ipvsadm コマンドで確認することができます。例のように、Web サーバのアドレスが表示されれば、設定が正しく行えています。ただし、この時点ではまだ Web サーバの設定が正しく行えていませんので、Weight の値が 0 です。正しく設定が行えれ

ば、Weight が 1 になります。

▼*ipvsadm コマンドによるサービスの確認*

```
# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.10.5:80 rr
  -> 192.168.20.10:80             Masq    0      0          0
  -> 192.168.20.11:80             Masq    0      0          0
                                          ↑   この欄
```

なお、heartbeat をシステムの起動時に自動起動する場合には、次のように設定を行います。

▼*heartbeat の自動起動の設定（cluster1, cluster2）*

```
# chkconfig heartbeat on
```

8.3 Web サーバの構築（NFS の場合）

Web サーバでは、ほとんど特別な設定は必要ありません。次のような設定を行います。

- ロードバランサの代表アドレス（192.168.20.1）をデフォルトゲートウェイとして、ネットワークを設定する。
- 共有ディスクをマウントする。
- チェック用ページを配置する。
- 通常どおり WWW サーバを起動する。

ここでは、共有ディスクのマウントとチェック用ページの配置について解説します。

8.3.1 共有ディスクのマウント

Web サーバでは、Web コンテンツを置き、これをサーバ間で共有するために、共有ディスクを利用します。特別な必要性がなければ、共有ディスクは NFS で構いません。

ディスクマウントの設定は、/etc/fstab で行います。次は、storage という名称のサーバの/shar をマウントする場合の設定例です。WWW サーバのドキュメントルート（例では/var/www/html）に共有ディスクをマウントします。

▼*/etc/fstab の設定（real1, real2）*

```
storage:/shar  /var/www/html   nfs   default  0 0
```

設定を行ったら、実際にマウントします。

▼*NFS ディスクのマウント（real1, real2）*

```
# mount /var/www/html
```

8.3.2 チェック用ページの配置

共有ディスク上の WWW サーバのドキュメントルートに、ロードバランサからのサービス監視で使用するチェック用ページを配置します。/etc/ha.d/ldirectord.cf の receive で設定した文字列と同じものを設定する必要があります。

▼*/var/www/html/check.html*

```
This server is active.
```

8.3.3 ロードバランサからの確認

設定を行い、WWW サーバを起動したら、ロードバランサが WWW サーバが起動したことを検知して自動的にロードバランスを開始します。下記のように、ipvsadm コマンドで Weight の値が 1 になっていることを確認してください。

▼*ipvsadm コマンドによるサービスの確認（cluster1）*

```
cluster1# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.10.5:80 rr
  -> 192.168.20.10:80             Masq    1      0          1
  -> 192.168.20.11:80             Masq    1      0          1
```

8.4　Web サーバの構築（iSCSI によるセッション情報の共有）

前節では、NFS サーバを使った事例を紹介しました。しかし、NFS を使うとロックがうまくできないため、PHP などのミドルウェアが用意しているセッション情報管理の仕組みを利用することができません。このようなセッション情報の共有が必要な場合には、iSCSI のディスクを使い OCFS などのクラスタファイルシステムでディスク共有を行う必要があります。

iSCSI のディスクを利用する場合には、次のような設定を行う必要があります。

- iSCSI イニシエータとしての設定を行う
- iSCSI ターゲットを検索する
- ファイルシステムを作成し、マウントする
- ファイルシステムの自動マウントを設定する
- セッション情報保存用ディレクトリを共有ディスク上に配置する
- コンテンツを共有ディスク上に配置する
- チェック用ページを配置する
- WWW サーバを起動する

8.4.1　iSCSI の設定

iSCSI を使うために、real1, real2 の 2 台のサーバに iSCSI 名の設定を行う必要があります。第 4 章で解説しましたように、イニシエータ名はネットワーク内で一意でなければなりません。当然、real1, real2 では違う名称を設定しなければなりません。次は、その設定例です。

▼*real1 の iSCSI 名の設定例（/etc/iscsi/initiatorname.iscsi）*

```
InitiatorName=iqn.2000-03.jp.designet:initiator.real1
```

▼*real2 の iSCSI 名の設定例（/etc/iscsi/initiatorname.iscsi）*

```
InitiatorName=iqn.2000-03.jp.designet:initiator.real2
```

次に、各サーバで iscsi サービスを起動します。

▼*Open-iSCSI での iscsi サービスの起動（real1, real2）*

```
# service iscsi start                                    ← iscsi サービスを起動
iscsid dead but pid file exists
Turning off network shutdown. Starting iSCSI daemon:       [  OK  ]
                                                           [  OK  ]
Setting up iSCSI targets: iscsiadm: No records found!
                                                           [  OK  ]
# iscsiadm -m discovery -t sendtargets -p 192.168.30.1:3260        ← ターゲットを検索
192.168.30.1:3260,1 iqn.2000-03.jp.designet:storage.test
# iscsiadm -m node -T iqn.2000-03.jp.designet:storage.test -p 192.168.30.1:3260 -l
                                                                        ←ログイン
Logging in to [iface: default, target: iqn.2000-03.jp.designet:storage.test, portal:
192.168.30.1,3260]
```

```
Login to [iface: default, target: iqn.2000-03.jp.designet:storage.test, portal:
192.168.30.1,3260]: successful

# dmesg                                                    ← カーネルメッセージを取得
       :
scsi8 : iSCSI Initiator over TCP/IP
  Vendor: IET       Model: Controller        Rev: 0001
  Type:   RAID                               ANSI SCSI revision: 05
scsi 8:0:0:0: Attached scsi generic sg0 type 12
  Vendor: IET       Model: VIRTUAL-DISK      Rev: 0001
  Type:   Direct-Access                      ANSI SCSI revision: 05
SCSI device sdb: 2104515 512-byte hdwr sectors (1078 MB)
sdb: Write Protect is off
sdb: Mode Sense: 79 00 00 08
SCSI device sdb: drive cache: write back
SCSI device sdb: 2104515 512-byte hdwr sectors (1078 MB)
sdb: Write Protect is off
sdb: Mode Sense: 79 00 00 08
SCSI device sdb: drive cache: write back
 sdb: unknown partition table
sd 8:0:0:1: Attached scsi disk sdb                         ← /dev/sdb として認識した
sd 8:0:0:1: Attached scsi generic sg1 type 0
```

iscsi サービスは、システムの起動時に自動的に実行されるように設定しておくのが便利です。

▼*iscsi サービスの自動起動設定*

```
# chkconfig iscsi on
```

8.4.2　OCFS ファイルシステムの作成とディスクのマウント

iscsi のディスクをカーネルに認識させることができたら、ファイルシステムを作成することができます。OCFS ファイルシステムを作成するためには、real1, real2 に設定ファイルを作成する必要があります。次は、設定ファイルの例です。ファイル共有に参加させるノード数、各ノードの IP アドレスやホスト名などを適切に設定します。

▼*OCFS のホスト設定/etc/ocfs2/cluster.conf の例（real1, real2）*

```
cluster:
        node_count = 2
        name = ocfs2

node:
        ip_port = 7777
        ip_address = 192.168.30.10
        number = 0
```

```
        name = real1
        cluster = ocfs2

node:
        ip_port = 7777
        ip_address = 192.168.30.11
        number = 1
        name = real2
        cluster = ocfs2
```

設定ファイルができたら、各サーバで o2cb サービスを起動します。

▼*通信条件の設定とサービスの起動（real1, real2）*

```
# service 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) [n]: y
Cluster stack backing O2CB [o2cb]:
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]:
Writing O2CB configuration: OK
Loading filesystem "configfs": OK
Mounting configfs filesystem at /sys/kernel/config: OK
Loading filesystem "ocfs2_dlmfs": OK
Creating directory '/dlm': OK
Mounting ocfs2_dlmfs filesystem at /dlm: OK
Starting O2CB cluster ocfs2: OK
```

途中で通信パラメータを聞かれますが、これはすべてのホストで同じになるように設定する必要があります。o2cb サービスが起動できましたら、ファイルシステムを作成することができます。ファイルシステムの作成は、どれか 1 台のサーバで実施します。

▼*OCFS ファイルシステムの作成（real1）*

```
real1# mkfs -t ocfs2 -N 2 -L ocfs2_fs0 /dev/sdb
mkfs.ocfs2 1.4.2
Cluster stack: classic o2cb
```

```
Filesystem label=ocfs2_fd0
Block size=4096 (bits=12)
Cluster size=4096 (bits=12)
Volume size=1077510144 (263064 clusters) (263064 blocks)
9 cluster groups (tail covers 5016 clusters, rest cover 32256 clusters)
Journal size=67108864
Initial number of node slots: 2
Creating bitmaps: done
Initializing superblock: done
Writing system files: done
Writing superblock: done
Writing backup superblock: 1 block(s)
Formatting Journals: done
Formatting slot map: done
Writing lost+found: done
mkfs.ocfs2 successful
```

ファイルシステムの作成ができたら、各サーバでマウントすることができます。この例では、/data にマウントします。

▼*ファイルシステムのマウント（real1, real2）*

```
# mkdir /data
# mount /dev/sdb /data
# df /data                              ←マウント状態を確認
Filesystem           1K-ブロック      使用    使用可 使用% マウント位置
/dev/sdb               7993312     149052   7438220    2% /data
```

8.4.3　ファイルシステムの自動マウントの設定

iSCSI のディスクが、システムの起動時に自動的にマウントされるように設定します。iscsi サービスを自動的に起動されるように設定しますが、iscsi はネットワークを使ったファイルシステムですので、netfs サービスも利用します。念のため明示的に自動起動の設定を行っておきます。

▼*iscsi と netfs サービスの自動起動設定（real1, real2）*

```
# chkconfig iscsi on
# chkconfig netfs on
```

起動時にディスクをマウントする設定は、/etc/fstab で行います。次は、その設定例です。

▼*OCFS のマウント設定 /etc/fstab（real1, real2）*

```
/dev/sdb              /data                 ocfs2  defaults,_netdev      0 0
```

４つ目の欄には、“_netdev”というオプションを追加しています。これは、ネットワークの初期化後

に netfs サービスでマウント処理を行うというオプションです。

8.4.4　セッション用ディレクトリとコンテンツディレクトリの設定

共有ディスク上に、セッション管理用のディレクトリとコンテンツ用のディレクトリを作成します。ここでは、それぞれ/data/session/、/data/htdocs/という名称で作成します。また、/data/session/はWWW サーバプロセスから書き込みができるように、所有者を設定しておきます。

▼*セッション用、コンテンツ用ディレクトリの作成*

```
# mkdir /data/session /data/htdocs
# chown apache:apache /data/session
```

WWW サーバのドキュメントルートが/data/htdocs/になるように設定を変更します。

▼*ドキュメントルートの設定（/etc/httpd/conf/httpd.conf）*

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot ”/data/htdocs”                                    ←　変更
```

ドキュメントルートの変更に合わせて、アクセス権の設定も変更する必要があります。

▼*ドキュメントルートへのアクセス制御の設定（/etc/httpd/conf/httpd.conf）*

```
#
# This should be changed to whatever you set DocumentRoot to.
#
<Directory "/data/htdocs">                                          ←　変更

#
# Possible values for the Options directive are "None", "All",
# or any combination of:
#   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
#
# Note that "MultiViews" must be named *explicitly* --- "Options All" # doesn't give it
to you.
#
# The Options directive is both complicated and important.  Please see
# http://httpd.apache.org/docs/2.2/mod/core.html#options
# for more information.
#
    Options Indexes FollowSymLinks

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
    AllowOverride None

#
# Controls who can get stuff from this server.
#
    Order allow,deny
    Allow from all

</Directory>
```

また、ミドルウェアの設定を変更し、セッション情報が/data/session/に作成されるように設定します。次は、PHPのセッション情報の配置場所を変更する例です。

▼*セッションの設定（/etc/php.ini）*

```
session.save_path = "/data/session"
```

8.4.5　チェック用ページの配置

共有ディスク上のWWWサーバのドキュメントルートに、ロードバランサからのサービス監視で使用するチェック用ページを配置します。ldirectordに設定したチェック用文字列と同じものを設定す

る必要があります。

▼*/data/htdocs/check.html*

```
This server is active.
```

8.4.6　ロードバランサからの確認

設定を行い WWW サーバを起動したら、ロードバランサがサーバが起動したことを検知して自動的にロードバランスを開始します。下記のように、ipvsadm コマンドで Weight の値が 1 になっていることを確認してください。

▼*ipvsadm によるサービスの確認（cluster1）*

```
cluster1# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.10.5:80 rr
  -> 192.168.20.10:80             Masq    1      0          1
  -> 192.168.20.11:80             Masq    1      0          1
```

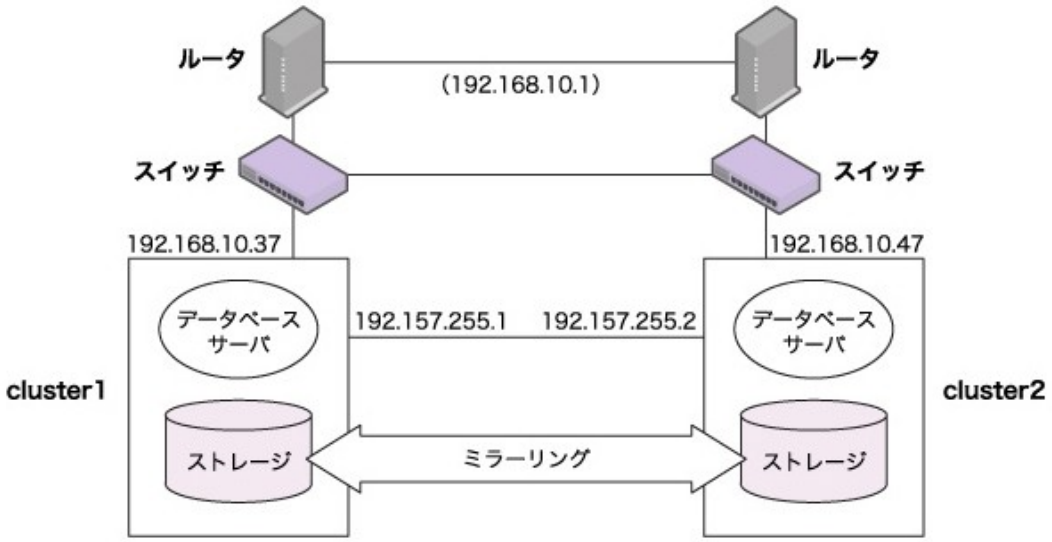
9章 アクティブ・スタンバイクラスタによるシステムの構築

本章では、第 3 章で学習したアクティブ・スタンバイクラスタの仕組みと第 4 章で学習したネットワークミラーリングの技術を使い、実際にシステムを構築する事例を取り上げます。

9.1 システムの概要

ここでは、DB サーバ 2 台をアクティブ・スタンバイクラスタを使って冗長化する構成を事例として取り上げます。図 9-1 のように、システムは単独障害点がないように、ルータや SW も含めてすべて冗長化されています。cluster1, cluster2 は heartbeat を使って冗長化します。また、サーバ間ではネットワークミラーリングを使ってデータを共有し、データベースのデータは共有ディスク上に配置します。

図 9-1:アクティブ・スタンバイクラスタの構成例



9.2　DRBD の設定

9.2.1　DRBD の入手

DRBD は、Debian、Fedora、CentOS、OpenSuSE などではパッケージが提供されています。これらのディストリビューションを使ってサーバを構築する場合には、パッケージを利用することができます。また、次の DRBD の公式ホームページでは、様々なディストリビューション用のビルト済みパッケージを公開しています。

http://www.drbd.org/

カーネルモジュールはカーネルバージョン毎に異なりますので、現在インストールされているカーネルのバージョンに合わせて入手してください。現在のカーネルバージョンは次のようにして調べることができます。

▼*カーネルバージョンの調査*

```
# uname -r
2.6.18-194.3.1.el5
```

パッケージを入手したら、それをインストールします。

▼*DRBD 関連パッケージのインストール*

```
# rpm -iv kmod-drbd83-8.3.2-6.el5_3 drbd83-8.3.2-6.el5_3
```

この例では、kmod-drbd83-8.3.2-6.el5_3 がカーネルモジュール、drbd83-8.3.2-6.el5_3 が DRBD の管理コマンドのパッケージです。

9.2.2　パーティションの準備

DRBD の設定を行う前に、DRBD で共有するデータ用のディスクパーティションを両方のサーバで用意しておく必要があります。また、管理用のメタデータを書き込むためのパーティションも必要です。まずは fdisk を起動し、ディスクパラメータを表示します（この例では未使用のディスクを使う場合を想定しています）。

▼*パーティションの表示*

```
# fdisk /dev/sdb
Command (m for help): p                              ← パーティション情報の表示

Disk /dev/sdb: 858 MB, 858993152 bytes
64 heads, 32 sectors/track, 819 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes      ← ディスクの管理単位

   Device Boot      Start         End      Blocks   Id  System
```

Units の欄に、ディスクの管理単位が表示されています。この例では、1,048,576bytes（1Mbyte）であることが分かります。最初にメタデータを格納する領域を作成します。

▼*メタデータ用パーティションの作成*

```
Command (m for help): n                                    ← 新しいパーティション作成
Command action
   e   extended
   p   primary partition (1-4)
p                                                          ← プライマリパーティションを指定
Partition number (1-4): 1                                  ← パーティション番号: 1
First cylinder (1-819, default 1): 1
Last cylinder or +size or +sizeM or +sizeK (1-819, default 819): 128 ← サイズ

Command (m for help): p                                    ← パーティション情報の表示

Disk /dev/sdb: 858 MB, 858993152 bytes
64 heads, 32 sectors/track, 819 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1         128      131056   83  Linux
```

DRBD のメタデータ用パーティションは、共有するディスクリソース 1 つに対して、最低でも 128Mbyte が必要です。そのため、この例ではサイズに 128 を指定しています（ディスクがもっと大きい場合には、第 4.6.4 項にあるメタ領域サイズの計算式に基づいて計算する必要があります）。先ほど確認した管理サイズは 1Mbyte でしたので、これで 128Mbyte になっています。fdisk では +128M のようにも指定できますが、この場合約 128Mbyte が割り当てられ、正確には 128Mbyte に満たない可能性があります。ですから、この例のように自分で計算して必要なユニット数を指定します。

次にデータ用のパーティションを設定します。次の例では、ディスク全体をデータ用パーティションにしています。

▼*データパーティションの作成*

```
Command (m for help): n                                    ← 新しいパーティション作成
Command action
   e   extended
   p   primary partition (1-4)
p                                                          ← プライマリパーティションを指定
Partition number (1-4): 2                                  ← パーティション番号: 2
First cylinder (129-819, default 129):                     ← 開始ブロック
Using default value 129
Last cylinder or +size or +sizeM or +sizeK (129-819, default 819): ← 最終ブロック
```

```
Using default value 819

Command (m for help): p                                  ← パーティション情報の表示

Disk /dev/sdb: 858 MB, 858993152 bytes
64 heads, 32 sectors/track, 819 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1         128      131056   83  Linux
/dev/sdb2             129         819      707584   83  Linux
```

パーティション設定が完了したら、最後にパーティション情報を書き込んで fdisk を終了します。

```
Command (m for help): w                                  ← 情報の書き込み
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

9.2.3　DRBD 設定ファイル

パーティションの準備ができましたら DRBD の設定を行います。設定は、/etc/drbd.conf で行います。

▼*DRBD 設定ファイル /etc/drbd.conf （cluster1, cluster2）*

```
resource r0 {                                                    ← リソース名の定義
  protocol C;                                                    ← 同期プロトコル
  handlers {
    local-io-error "echo o > /proc/sysrq-trigger ; halt -f";     ← I/O エラーハンドラ
  }
  startup {
    degr-wfc-timeout 120;    # 2 minutes.                        ← 起動時の待ち時間
  }
  disk {
    on-io-error   call-local-io-error;                           ← I/O エラー時の処理
  }
  syncer {
    rate 300M;                                                   ← 同期速度
  }
  on cluster1 {                                                   ← clusrer1 の設定
    device     /dev/drbd0;                                       ← DRBD デバイス名
    disk       /dev/sdb2;                                        ← データデバイス
    address    192.168.255.1:7788;                               ← 同期アドレス
```

```
    meta-disk  /dev/sdb1[0];                                    ← メタデータ
  }
  on cluster2 {                                                 ← clusrer2の設定
    device     /dev/drbd0;
    disk       /dev/sdb2;
    address    192.168.255.2:7788;
    meta-disk  /dev/sdb1[0];
}
```

degr-wfc-timeoutでは、コネクション待ちを行う時間を設定します。この時間が経過すると、DRBDはデグレードモードで起動します。その場合、相手サーバは強制的に切り離され、単独で動作します。また、on-io-errorでは、ディスクデバイスレベルでエラーが発生した場合の処理を設定します。次の3つを指定することができます。

- detach　　ローカルディスクを切り離しディスクレスで稼働を続ける
- call-local-io-error　　local-io-errorハンドラを呼び出す
- pass_on　　エラーをファイルシステムにそのまま通知する

この例では、call-local-io-errorハンドラを呼び出して、/proc/sysrq-triggerに“o”を書き込んでいますが、これによってシステムを強制的にダウンさせます。それでも駄目な場合には、haltコマンドを使って強制終了を試みます。

設定が終了したら、両方のノードでメタデータを作成しておきます。

▼*メタデータの作成例（cluster1, cluster2）*

```
# drbdadm create-md r0
Writing meta data...
initializing activity log
NOT initialized bitmap
New drbd meta data block successfully created.
```

r0は、/etc/drbd.confで設定したリソース名です。

9.2.4　DRBDの初期設定

ここまでの作業が完了したら、DRBDを起動することができます。プライマリノード（この例ではcluster1）から起動します。

▼*DRBDの起動（cluster1）*

```
cluster1# service drbd start
Starting DRBD resources: [ d(r0) s(r0) n(r0) ].........
***************************************************************
 DRBD's startup script waits for the peer node(s) to appear.
 - In case this node was already a degraded cluster before the
   reboot the timeout is 120 seconds. [degr-wfc-timeout]
 - If the peer was available before the reboot the timeout will
```

```
  expire after 120 seconds. [wfc-timeout]        ← degr-wfc-timeout の値
  (These values are for resource 'r0'; 0 sec -> wait forever)
To abort waiting enter 'yes' [ 119]:                    ← カウントアップ
```

この時点では、セカンダリノード(cluster2)でDRBDが起動していませんので、メッセージのようにセカンダリノードの接続確認待ちになります。接続出来ない場合は、degr-wfc-timeoutで設定された時間(この例では120秒)だけ待機した後、デグレードモードで起動します。この状態でプライマリノードでDRBDの状態を確認すると次のようになっています。

▼*起動待ち中のDRBDの状態*

```
host1# service drbd status
drbd driver loaded OK; device status:
version: 8.3.2 (api:88/proto:86-90)
GIT-hash:    dd7985327f146f33b86d4bff5ca8c94234ce840e    build    by    mockbuild@v20z-x86-
64.home.local, 2009-08-29 14:02:24
m:res  cs            ro                 ds                      p  mounted  fstype
0:r0   WFConnection  Secondary/Unknown  Inconsistent/DUnknown  C       ← 接続状態
```

各表示欄は、次のようなことを示しています。

- cs　コネクションの状態
- ro　自ノードと相手ノードの状態
- ds　自ノードと相手ノードのデータの状態

ro、dsの表示は、「自ノードの状態/相手ノードの状態」のように表示されます。したがって、この状態は、WFConnection(接続待ち)、自ノードの状態がSecondary、相手ノードの状態はUnknown(つまり不明)、自ノードのデータ状態がInconsistent(一貫性が取れていない)、相手ノードのデータ状態がDUnknown(不明)になります。

次にセカンダリノード(cluster2)でDRBDを起動します。

▼*DRBDの起動 (cluster2)*

```
cluster2# service drbd start
Starting DRBD resources:    [ d(r0) s(r0) n(r0) ].
```

DRBDの状態は次のように変化します。

▼*起動待ち中のDRBDの状態*

```
cluser1# service drbd status
drbd driver loaded OK; device status:
version: 8.3.2 (api:88/proto:86-90)
GIT-hash:    dd7985327f146f33b86d4bff5ca8c94234ce840e    build    by    mockbuild@v20z-x86-
64.home.local, 2009-08-29 14:02:24
m:res  cs         ro                   ds                         p  mounted  fstype
0:r0   Connected  Secondary/Secondary  Inconsistent/Inconsistent  C
```

9.2.5　データの同期

プライマリノード（cluster1）をプライマリ状態へ移行します。初回は、データの状態がInconsistentですので、強制的に移行処理を行う必要があります。

▼*プライマリへの強制移行*

```
cluster1# drbdadm -- --overwrite-data-of-peer primary all
```

強制移行を行うと、データの同期が開始されます。

▼*データ同期中の状態*

```
cluster1# service drbd status
drbd driver loaded OK; device status:
version: 8.3.2 (api:88/proto:86-90)
GIT-hash:    dd7985327f146f33b86d4bff5ca8c94234ce840e    build    by    mockbuild@v20z-x86-
64.home.local, 2009-08-29 14:02:24
m:res  cs          ro                 ds                     p  mounted  fstype
0:r0   SyncSource  Primary/Secondary  UpToDate/Inconsistent  C
...    sync'ed:    11.0%              (699316/779156)K
```

接続状態がSyncSourceになり、同期の進捗状況（この例では11%）が表示されます。最終的に、次の例のように接続状態がConnected（接続中）、DRBDの状態がそれぞれ”Primary/Secondary”となり、ローカルディスクの状態が“UpToDate/UpToDate”（最新の状態で同期済み）になれば完了です。

▼*同期完了時の状態*

```
cluster1# service drbd status
drbd driver loaded OK; device status:
version: 8.3.2 (api:88/proto:86-90)
GIT-hash:    dd7985327f146f33b86d4bff5ca8c94234ce840e    build    by    mockbuild@v20z-x86-
64.home.local, 2009-08-29 14:02:24
m:res  cs         ro                 ds                 p  mounted  fstype
0:r0   Connected  Primary/Secondary  UpToDate/UpToDate  C
```

9.2.6　ファイルシステムの作成とマウント

DRBDのデータ同期が完了したら、ファイルシステムを作成し、マウントすることができます。作業は必ずプライマリノードで実施します。この例では、/dataにマウントしています。

▼*ファイルシステムの作成*

```
cluster1# mke2fs -j /dev/drbd0            ←ext3ファイルシステムを作成
mke2fs 1.39 (29-May-2006)
       :
```

```
       :
cluster1# mkdir /data
cluster1# mount /dev/drbd0 /data          ←マウント
cluster1# df /data                         ←マウント状態を確認
Filesystem           1K-ブロック     使用    使用可 使用% マウント位置
/dev/drbd0              7993312    149052   7438220    2% /data
```

ここまで確認できたら、一旦ファイルシステムはアンマウントしておきます。

▼*ファイルシステムのアンマウント*

```
cluster1# umount /data
```

最後にシステム起動時に自動的に DRBD が起動されるように設定しておきます。

▼*DRBD の自動起動の設定*

```
# chkconfig --add drbd
# chkconfig drbd on
```

9.3 クラスタの設定

cluster1、cluster2 をアクティブ・スタンバイクラスタとして構成するためには、heartbeat をインストールし設定を行う必要があります。

9.3.1 heartbeat の設定

クラスタの設定は、/etc/ha.d/ha.cf で行います。最初に基本設定を行います。cluster1, cluster2 の両方のノードで監視時間などのパラメータは同じでなければなりません。ただし、ucast の相手 IP アドレスは必ず異なります。

▼*/etc/ha.d/ha.cf (cluster1)*

```
logfile /var/log/ha.log                    ←  ログファイル

keepalive 2                                ←  監視間隔
deadtime 30                                ←  ダウンと判定する時間
warntime 10                                ←  ログに記録するまでの時間
initdead 120                               ←  起動後、監視開始までの時間

udpport 694                                ←  相手ノードのポート番号
ucast eth1 192.168.255.2                   ←  相手ノードの IP アドレス
baud 19200                                 ←  シリアルの通信速度
serial /dev/ttyS0                          ←  シリアルポートのデバイス

auto_failback on                           ←  自動フェイルバック（有効）

watchdog /dev/watchdog                     ←  watchdog デバイス

node cluster1 cluster2                     ←  クラスタ構成ノード名
```

▼*/etc/ha.d/ha.cf (cluster2)*

```
logfile /var/log/ha.log

keepalive 2
deadtime 30
warntime 10
initdead 120

udpport 694
ucast eth1 192.168.255.1                   ←  ここだけが違う
baud 19200
serial /dev/ttyS0
```

```
auto_failback on

watchdog /dev/watchdog

node cluster1 cluster2
```

次に、各ノード間の認証設定を行います。設定は、/etc/ha.d/authkeys で行います。次のようにファイルを作成してください。両ノードとも同じ設定でなければなりません。

▼*/etc/ha.d/authkeys（cluster1, cluster2）*

```
auth 1
1 crc
```

このファイルは、heartbeat 管理ユーザしかアクセスできないようにしておく必要があります。

▼*ユーザ hacluster しかアクセスできないようにする*

```
# chown hacluster:haclient /etc/ha.d/authkeys
# chmod 600 /etc/ha.d/authkeys
```

さらに、各ノードの/etc/ha.d/haresources に共有リソースの設定を行います。この設定ファイルには、先頭に稼働系サーバの名前を記載し、その後ろにリソースを列挙します。

▼*/etc/ha.d/haresources（cluster1, cluster2）*

```
cluster1 drbddisk Filesystem::/dev/drbd0::/data::ext3 IPaddr::192.168.10.5/24
```

この例では、DRBD のプライマリにする drbddisk リソース、DRBD ファイルシステムをマウントするための設定、共有 IP アドレスの設定を行っています。このファイルの設定は、両ノードとも同じにしておく必要があります。

9.3.2　heartbeat の起動

設定が完了したら、cluster1, cluster2 で heartbeat を起動します。うまく設定が行われていれば、heartbeat が順次リソースを起動します。

▼*heartbeat の起動（cluster1, cluster2）*

```
# /etc/init.d/heartbeat start
Starting High-Availability services:
                                                    [  OK  ]
```

しばらくすると、cluster1 に代表 IP アドレスが設定され、DRBD のディスクもマウントされます。

▼*代表 IP アドレスが付与されているかを確認*

```
cluster1# ifconfig eth0:0
eth0:0    Link encap:Ethernet  HWaddr 00:0C:29:90:1F:AB
          inet addr:192.168.10.5  Bcast:192.168.10.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:18 Base address:0x1480
cluster1# df /data                          ←マウント状態を確認
Filesystem           1K-ブロック     使用    使用可 使用% マウント位置
/dev/drbd0             7993312    149052   7438220    2% /data
```

9.4 アプリケーションの設定

heartbeat, DRBD の設定が完了したら、その上で動作するアプリケーションの設定を行うことができます。どのようなアプリケーションでも、基本的な設定は同じです。次のような作業を行う必要があります。

- アプリケーションが扱うデータと設定ファイルを共有ディスク上に配置する
- 共有ディスク上の設定ファイルや、データを参照するようにシステムを設定する
- アプリケーションの稼働状態を確認するプログラムを作成する
- リソーススクリプトを作成する
- heartbeat のリソースとして登録する

この事例では、アプリケーションとしてデータベース（PostgreSQL）を使い、両方のサーバに PostgreSQL がインストールされているものとして解説します。

9.4.1 共有ディスクへのファイルの配置

PostgreSQL のデータを配置するためのディレクトリを作成します。作業は、DRBD のディスクをマウントしているプライマリノード（cluster1）で行う必要があります。

▼*共有ディスクへのディレクトリの作成*

```
cluster1# mkdir -p /data/pgsql/etc /data/pgsql/data          ← ディレクトリ作成
cluster1# chown postgres:postgres /data/pgsql/data           ← アクセス権設定
```

作成したディスク上にデータベースファイルを作成します。

▼*データベースの作成*

```
cluster1# su - postgres                                      ← ユーザ切り替え
cluster1$ initdb -D /data/pgsql/data -E UTF8                 ← データベースファイルの作成
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale ja_JP.UTF-8.
initdb: could not find suitable text search configuration for locale ja_JP.UTF-8
The default text search configuration will be set to "simple".

fixing permissions on existing directory /data/pgsql/data ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers/max_fsm_pages ... 24MB/153600
creating configuration files ... ok
creating template1 database in /data/pgsql/data/base/1 ... ok
```

```
initializing pg_authid ... ok
initializing dependencies ... ok
creating system views ... ok
loading system objects' descriptions ... ok
creating conversions ... ok
creating dictionaries ... ok
setting privileges on built-in objects ... ok
creating information schema ... ok
vacuuming database template1 ... ok
copying template1 to template0 ... ok
copying template1 to postgres ... ok

WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the -A option the
next time you run initdb.

Success. You can now start the database server using:

    /usr/bin/postgres -D /data/pgsql/data
or
    /usr/bin/pg_ctl -D /data/pgsql/data -l logfile start
```

初期化を行うときに引数「-D /data/pgsql/data」を指定していますが、ここではデータベースの保管先をDRBD領域上に指定しています。また、「-E UTF8」では、データベースに格納する文字コードをUTF8にしています。実際には、用途に合わせて修正する必要があります。

データベースのデータが初期化できたら、データベースの設定を行う必要があります。ここでは詳細な設定は紹介しませんが、少なくともデータベースへのアクセスが代表IPアドレスに対して行われるように、dataディレクトリ配下のpostgresql.confに設定する必要があります。

▼*受け付けIPアドレスの追加（/data/pgsql/data/postgresql.conf）*

```
listen_addresses = '192.168.10.5,localhost'
```

また、ネットワークからの参照を許可するようにアクセス制御の設定も行う必要があります。これは、dataディレクトリ配下のpg_hba.confファイルで行います。

▼*アクセス制御の設定（/data/pgsql/data/pg_hba.conf）*

```
host    all     all     192.168.10.0/24   trust
```

アクセス制御の設定については、実際には、用途に合わせて修正する必要があります。

9.4.2　システム設定の変更

PostgreSQLは、標準で/data/pgsql/data/を参照するようになっていません。そのため、サービ

スの起動時にこのディレクトリを使うように設定を行う必要があります。PostgreSQL の postmaster を起動するときに、オプションを指定することでデータを保管するディレクトリを変更することができます。CentOS、Fedora などでは、/etc/sysconfig/pgsql/postgresql ファイルに次のように記載することで変更できます。

▼*/etc/sysconfig/pgsql/postgresql1(cluster1, cluster2)*

```
PGDATA=/data/pgsql/data
```

このような起動スクリプトの設定を変更できないような場合には、標準的なデータ保管用ディレクトリにシンボリックリンクを設定します。

▼*起動スクリプトなどで調整できない場合(cluster1, cluster2)*

```
# mv /var/lib/pgsql/data /var/lib/pgsql/data.bak
# ln -s /data/pgsql/data /var/lib/pgsql
```

どちらの方法でも構いません。この設定は、cluster1, cluster2 の両方で行う必要があります。

9.4.3　サービス監視スクリプトの導入

設定ができたら、クラスタ上でサービスの動作が正しく行われているかを確認するためのチェックスクリプトを作成します。この事例では、PostgreSQL のデータベース上にチェック用のデータベースを作成し、そのデータベースへのクエリ結果が正しく返却されることを確認するようにスクリプトを作成します。

一旦 PostgreSQL を起動します。

▼*postgresql サービスの起動*

```
cluster1# service postgresql start
データベースを初期化中:                                    [  OK  ]
postgresql サービスを開始中:                               [  OK  ]
```

チェック用のテーブルを作成します。

▼*動作確認用テーブルの作成*

```
cluster1#  su - postgres                                ← ユーザ切替
cluster1$ createdb check                                ← データベース作成
cluster1$ psql -d check                                 ← データベースへ接続
Welcome to psql 8.3.5, the PostgreSQL interactive terminal.

Type:  ¥copyright for distribution terms
       ¥h for help with SQL commands
       ¥? for help with psql commands
       ¥g or terminate with semicolon to execute query
```

```
       ¥q to quit

check=# create table checktbl (                          ← チェック用テーブル作成
id int,
data char(10)
);
CREATE TABLE
check=# insert into checktbl values(1, 'OK');            ← チェック用データ登録
INSERT 0 1
check=# ¥q                                               ← 終了
```

チェック用テーブルにアクセスして、動作を確認します。

▼*SQL の実行例*

```
cluster1$ echo 'select data from checktbl where id=1;' | psql -A -t check
OK
```

OKと表示されれば、動作確認は終了です。この仕組みを組み込んだ、サービス監視スクリプトを作成し、cluster1, cluster2 に配置します。次は、その例です。

▼*サービス監視スクリプト /usr/local/bin/check_network　(cluster1, cluster2)*

```
#! /bin/sh
#
# check_network: リンク状態、通信状態、データベースサービス状態を確認する
#
INTERVAL=3                                                    #  監視間隔
CVIP=192.168.10.5                                             #  仮想 IP
IFNAME=eth1
GATEWAY=192.168.10.1                                          #  ゲートウェイ

#
# 監視ループ
#
while :
do
        echo "Checking..."

        sleep ${INTERVAL}

        #
        # 仮想 IP アドレスがこのサーバについているかを確認
        #
        /etc/ha.d/resource.d/IPaddr ${CVIP} status > /dev/null 2>&1
        if [ $? -ne 0 ]
        then
```

```
                continue
        fi

        #
        # インタフェースのLinkを確認
        #
        LINK=`/sbin/ethtool ${IFNAME} | awk '/Link detected: /{ print $3}'`
        if [ "$LINK" != "yes" ]
        then
                /usr/lib/heartbeat/heartbeat -k
                exit 100
        fi

        #
        # pingによる通信確認
        #
        ping -c 1 -w 1 ${GATEWAY} > /dev/null 2>&1
        if [ $? -ne 0 ]
        then
                /usr/lib/heartbeat/heartbeat -k
                exit 101
        fi

        #                                                    ← ここから
        # postgresqlサービスの確認
        #
        /etc/init.d/postgresql status > /dev/null 2>&1
        if [ $? -ne 0 ]
        then
                /usr/lib/heartbeat/heartbeat -k
                exit 102
        fi
        #
        # データベースへの接続確認
        #
        ANSWER=`echo 'select data from checktbl where id=1;' | ¥
               psql -A -t -U postgres check`  ← SQL発行
        if [ x$ANSWER != xOK ]
        then
                echo "$ANSWER" > /tmp/answer
                 /usr/lib/heartbeat/heartbeat -k
                 exit 103
        fi
done
```

ここまで設定できたら、一旦PostgreSQLを終了しておきます。

▼*PostgreSQLの停止*

```
cluster1# service postgresql stop
Stopping postgresql service:                               [  OK  ]_
```

9.4.4　リソーススクリプトの作成

最後にpostgresqlをリソースとして登録するためのリソーススクリプトを作成します。次のようにサービス起動スクリプトを実行し、“running/stopped”という単語を含むメッセージが表示される場合には、サービス起動スクリプトをそのままリソーススクリプトとして使えますが、そうでない場合には、独自のスクリプトを作成する必要があります。

▼*サービス起動スクリプトの検査*

```
# /etc/init.d/postgresql status
postmaster (pid 7861 7860 7859 7857 7855) を実行中...                ← running でない
# /etc/init.d/postgresql stop
Stopping postgresql service:                               [  OK  ]
# /etc/init.d/postgresql status
postmaster は停止しています                                          ← stopped でない
```

リソーススクリプトを自分で作成した場合には、/etc/ha.d/resource.d/に配置します。次の例は、こうしたリソーススクリプトの作成例です。

▼/etc/ha.d/resource.d/**postgresql**

```
#! /bin/sh
ORIGINAL=/etc/init.d/postgresql

if [ "$1" == "status" ]
then
    ${ORIGINAL} status > /dev/null 2>&1
    RET=$?
    if [ $RET -eq 0 ]
    then
        echo running
    else
        echo stopped
    fi
else
    ${ORIGINAL} $*
    RET=$?
fi
exit $RET
```

9.4.5　アプリケーションのクラスタへの組み込み

データベースの設定が完了しましたら、データベースをクラスタサービスに組み込みます。サービスの定義は、/etc/ha.d/haresources ファイルで行います。次は、その設定例です。

▼*データベースを組み込んだリソースファイル /etc/ha.d/haresource（cluster1, cluster2）*

```
cluster1 drbddisk Filesystem::/dev/drbd0::/data::ext3 postgresql IPaddr::192.168.10.5/24
```

データベースが、IP アドレスが付与される前に起動されることになるため、それを許可するようにカーネルパラメータを設定し、有効にしておく必要があります。

▼*カーネルパラメータの設定 /etc/sysctl.conf （cluster1, clster2）*

```
net.ipv4.ip_nonlocal_bind = 1
```

▼*カーネルパラメータを有効にする （cluster1, clster2）*

```
# sysctl -p
```

サービス監視スクリプトを heartbeat の設定に組み込みます。

▼*/etc/ha.d/ha.cf に追加（cluster1, cluster2）*

```
respawn root /usr/local/bin/check_network
```

設定が完了したら、cluster1, cluster2 で heartbeat を再起動します。適切に設定ができていれば、cluster1 で postgresql サービスが起動します。

10章 サーバの仮想化

1 つの物理的なハードウェアの中に、いくつもの仮想的なハードウェアを作成し、何台ものコンピュータがあるように見せる技術を仮想化（Virtualization）と呼びます。この章では、仮想化の技術の概要について学びます。

10.1　仮想化の概要

仮想化は、1 台の物理コンピュータの様々なリソースを分割し、それを使って複数の仮想サーバを作成する技術です。近年、仮想化の技術が注目されていますが、それは次のようなメリットがあるからです。

- エネルギー消費の低減
  ハードウェアが進化し、非常に大容量で高速なコンピュータが利用できるようになりましたが、その一方で電力消費量とコンピュータの発熱量が急速に増加しています。複数のコンピュータで実現していた機能を、仮想化により少数のコンピュータで実行できるようになれば、エネルギーの消費を抑えることができます。
- システムリソースの有効利用
  例えば、科学計算の場合には CPU、ファイルサーバではハードディスク、動画サーバではネットワークというように、サーバコンピュータは用途によって利用するリソースの傾向が異なります。こうした処理を別々のコンピュータで行うと、様々なリソースが利用されていない状態になります。仮想化により利用するリソースが異なる複数の処理を 1 つのコンピュータで実行できれば、コンピュータリソースをより有効に使うことができます。
- 古いシステムの継続利用
  コンピュータ上で利用しているソフトウェアが安定していれば、それを継続して利用したいと思うのは自然なことです。しかし、コンピュータのハードウェアは 3～5 年という比較的短い期間で入れ替えが必要になります。このとき、最新の OS でそのソフトウェアが動作しなかったり、古い OS が最新のコンピュータで動かなかったりすれば、そのソフトウェアを利用し続けることができなくなってしまいます。サーバの仮想化の技術を使えば、新しいハードウェアで古い OS が動作する環境を作ることができ、このようなソフトウェアを利用し続けることできます。

図 10-1:仮想サーバの動作イメージ

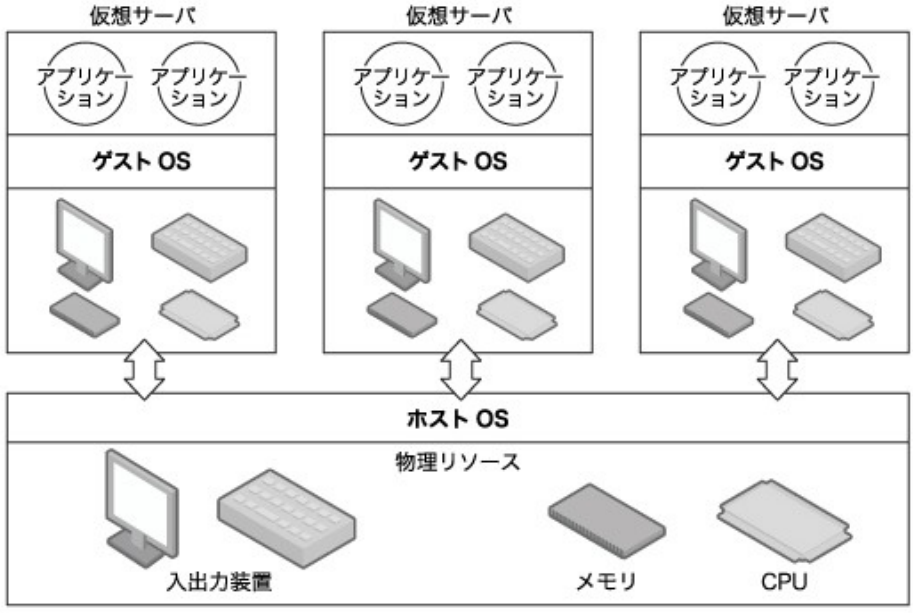


図 10-1 のように各仮想サーバでは、入力装置、CPU、メモリなどの各種のハードウェアリソースをエミュレートします。そのため、それぞれの仮想サーバは、論理的には独立したサーバ環境となります。1 つのサーバの中で複数の仮想サーバを動作させることができます。もちろん、Linux と Windows といった別々のオペレーティングシステムを動作させることも可能です。この仮想サーバの OS のことを**ゲスト OS** と呼び、ハードウェアを直接制御している OS のことを**ホスト OS** と呼びます。

10.2 仮想化の実現方式

仮想化には、ゲスト OS とホスト OS の動作方法が異なる、2 つの実現方法があります。

- 完全仮想化
  ホスト OS が様々なデバイスの動作を完全にエミュレートし、ゲスト OS が物理サーバ上で動作しているのとまったく同じ条件で稼働する方法です。仮想化の技術が使われるようになる前の古い OS でも動作する可能性が高いのが長所です。ただし、ホスト OS がデバイスの動きをすべてエミュレートするための、オーバーヘッドが大きいという短所があります。

- 準仮想化
  ゲスト OS が仮想サーバ上で動作していることを認識していて、専用のデバイスドライバなどを通じてホスト OS と連携する方法です。ホスト OS と連携するための機能をサポートしたゲスト OS しか動作しないという欠点がありますが、仮想化によるオーバーヘッドを最小限に抑えることができます。

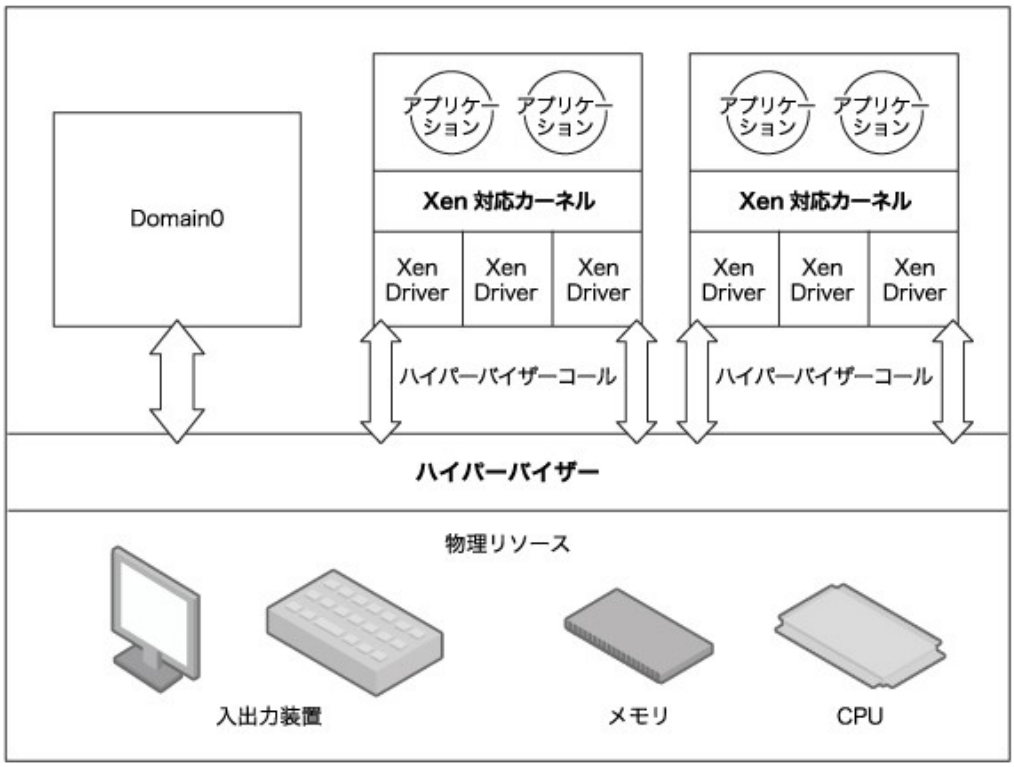
完全仮想化を利用する場合には、Intel VT-x または AMD Pacifica hardware virtualization(AMD-V)のどちらかのハードウェアサポートが必要です。そのため、完全仮想化は、これらの機能をサポートしたハードウェアでしか利用することができません。

Linux での実装

Linux では、複数の仮想化技術を利用することができます。主なものは次の通りです。

- Xen
  ケンブリッジ大学が開発した仮想化技術で、現在は Citrix 社が開発・サポートを行っています。Linux では、もっとも早くから採用された仮想化ソフトウェアです。Xen の開発は Microsoft 社とも協力して行われているため、Microsoft 社の仮想化技術である Hyper-V とは基本的な技術が同じで、Windows との親和性が高いのが特徴です。Xen では、完全仮想化も準仮想化もサポートしており、ホスト OS 上で稼働する Domain 0 が仮想 OS の制御とリソースの分割を行います(図 10-2)。また、CPU の準仮想化を行うことができ、Intel VT-x などのサポートがないハードウェアでも準仮想化モードで動作することができます。Python などを利用した GUI の管理ツールも利用することが可能です。

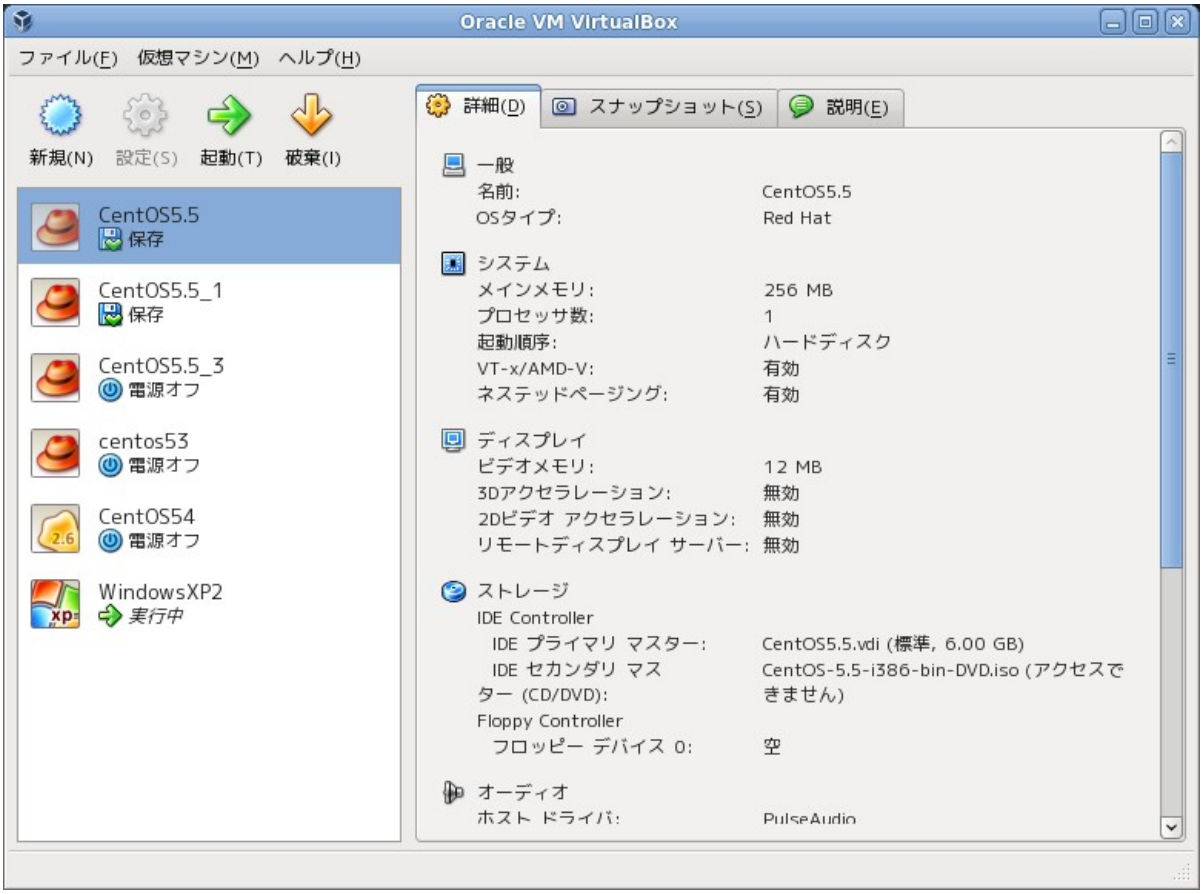
図 10-2:Xen のイメージ



- KVM（Kernel-based Virtual Machine）
  KVM は、Linux カーネルが標準サポートしている仮想化技術です。Linux Kernel 2.6.20 以降で利用することができます。カーネルモジュールとして提供されていますが、Linux の開発ロードマップではカーネルへ組み込まれることになっています。主に完全仮想化をサポートしています。I/O ドライバのための準仮想化用のドライバが提供されていますが、CPU の準仮想化には対応していません。そのため、KVM では Intel VT-x などのハードウェアによる仮想化のサポートが必須です。KVM は、Linux カーネルの一部として動作し、Xen に比べると非常にシンプルな実装となっています。KVM では、ゲスト OS のスケジューリングやリソースの分割は、Linux カーネルが行います。まだ開発の歴史が浅いためか、GUI 管理ツールなどはあまり用意されていません。
- VirtualBox
  VirtualBox は、Innotek 社が開発した仮想化技術で、現在は Oracle 社が開発を行っています。VirtualBox には、RDP（Remote Desktop Protocol）、USB デバイス、iSCSI をサポートした商用ライセンス版（個人の利用や教育あるいは評価目的の利用は無料）と、GPL V2 ライセンスの元で公開されているオープンソース版（VirtualBox-OSE）の 2 つの版があります。GUI コンソールや Web インタフェースなどの管理ツールが提供され、日本語化もされています(図 10-3)。VirtualBox は Intel VT-x などのハードウェアサポートを利用した完全仮想化をサポートしています。しかし、ハードウェアサポートを使わなくても、できるだけホスト OS の CPU 上で稼働するように作られていて、効率よく処理を行うことができます。これ

は、Xen や KVM の準仮想化とは異なる技術です。

図 10-3:VirutalBox の管理画面



11章 仮想サーバを構築する（Xen 編）

本章では、第 10 章で学習した仮想サーバの仕組みのうち、Xen を使って実際に仮想サーバシステムを構築する事例を取り上げます。

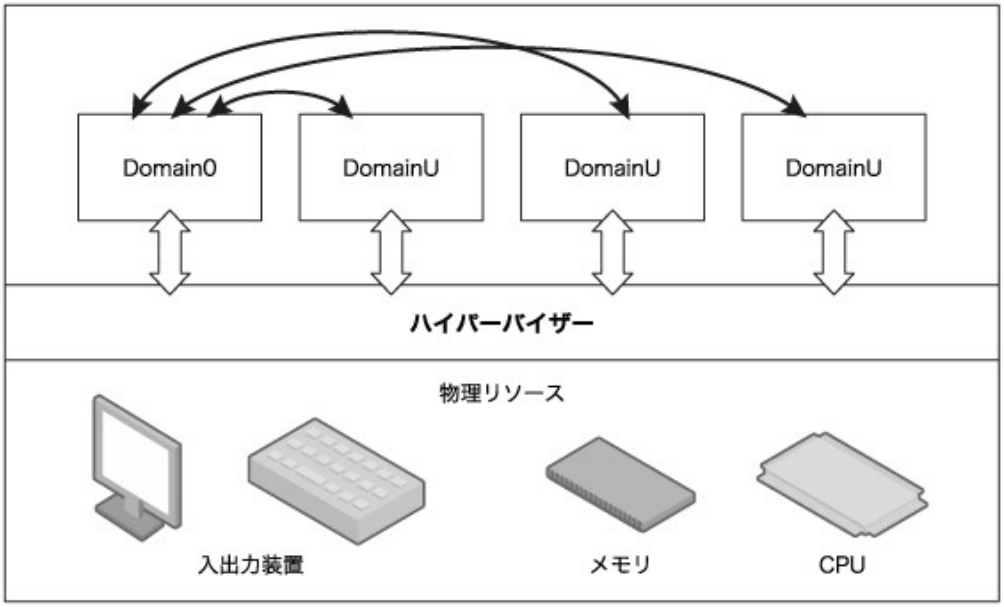
11.1 Xen とは

Xen は、広域分散コンピューティングの実装である XenoServer Project の一部として、英国 University of Cambridge Computer Laboratory で開発されました。その後、開発者が中心となって XenSource が設立されました。現在は、XenSource を中心とした Xen コミュニティで開発・管理が行われています。また、XenSource を買収した Citrix 社からは、Xen を製品化した商用仮想サーバシステムも販売されています。

Xen は、オープンソースの仮想サーバとしては比較的古くから利用されてきたこともあり、以前は広く様々なディストリビューションでサポートされていました。そのため、様々な機能の開発も進んでいます。

図 11-1 は、Xen のシステム構成を示したものです。

図 11-1:Xen のシステム構成



Xen では、ホスト OS をハイパーバイザー、ゲスト OS をドメインと呼びます。ドメインのうち、ハイパーバイザーの起動とともに自動的に準備される特殊なドメインを Domain 0 と呼びます。それ以外のドメインを Domain U と呼びます。Xen では、Domain 0 が物理デバイスの制御を行います。そのため、Domain U は Domain 0 を通して物理装置へアクセスすることになります。

Xen は、完全仮想化と準仮想化の両方をサポートしています。また、Xen では仮想マシンの設定を XML データベースで管理し、複数のホスト間で共有することができます。それを利用し、実行中の仮想マシンを別の物理ハードウェアへほぼ無停止で移動させることができます。これを**ライブマイグレーション**（Live Migration）と呼びます。

11.2 Xen のインストール

Xen をサポートした Linux ディストリビューションでは、通常のカーネルとは別に Xen 用のカーネルが用意されていますので、それをインストールします。また、python の仮想サーバ管理ツールが同梱されていれば、それもインストールしておきます。

次は CentOS5 でのインストール例です。

▼*CentOS での Xen のインストール*

```
# yum install xen kernel-xen python-virtinst
```

11.3 Xen ハイパーバイザーの設定

Xen によるサーバ仮想化を行うためには、通常のカーネルではなく Xen ハイパーバイザーを起動する必要があります。また、サーバ起動時に xend サービスが起動するようにしておく必要があります。

システムによっては自動的に起動されるようになっている場合もありますが、必要に応じて自動起動の設定を行っておきます。

▼*xend の自動起動設定*

```
# chkconfig xend on
```

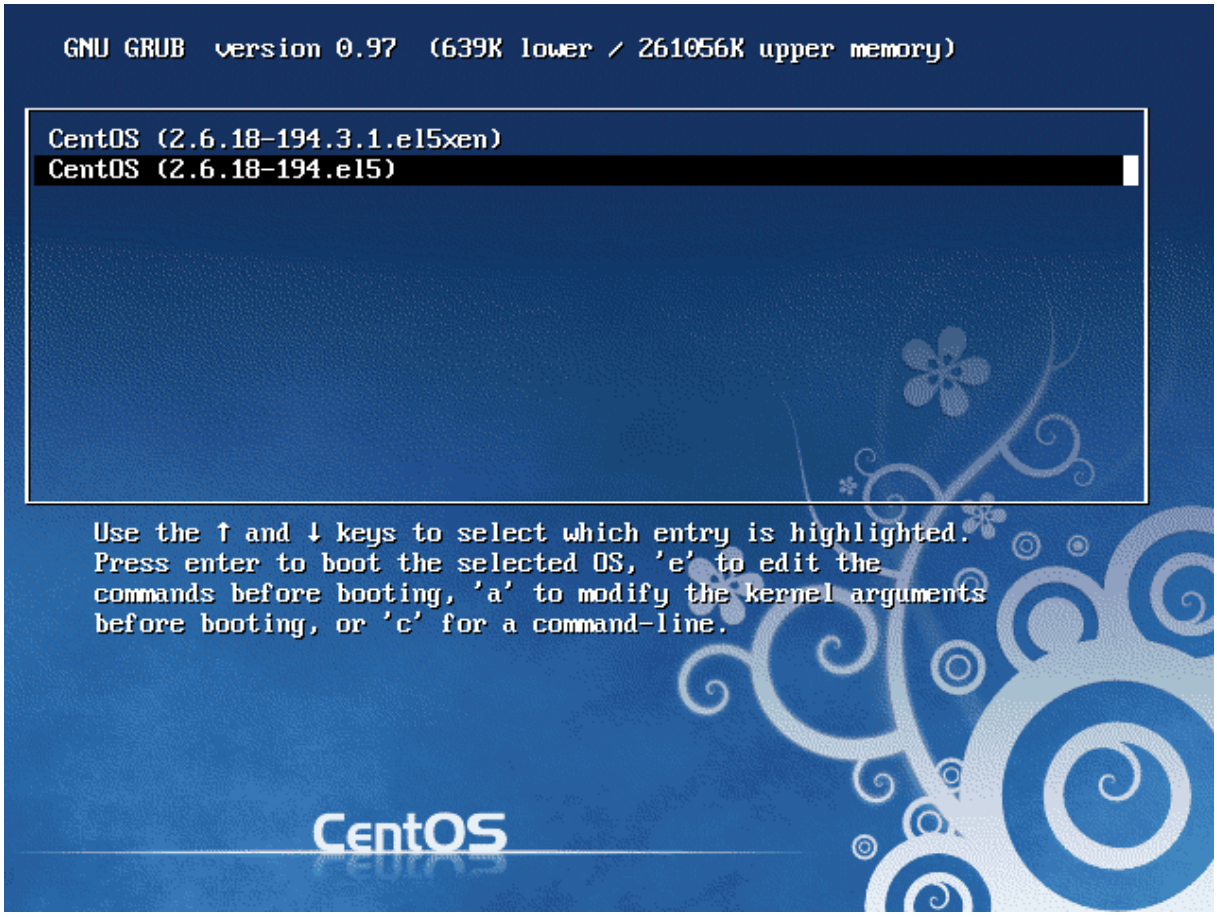
サーバを再起動し、起動時の GRUB メニュー(図 11-2)で xen を選択し、Xen ハイパーバイザーを起動します。

自動的に Xen のハイパーバイザーを起動したい場合には、/etc/grub.conf を変更しておく必要があります。次は、/etc/grub.conf の例です。

▼*Xen カーネルの起動設定 /etc/grub.conf*

```
Default=0                                                    ←変更
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.18-194.el5xen)                             ←0 番目 Xen ハイパーバイザー
       root (hd0,0)
       kernel /xen.gz-2.6.18-194.el5
       module /vmlinuz-2.6.18-194.el5xen ro root=/dev/VolGroup00/LogVol00 rhgb quiet
       module /initrd-2.6.18-194.el5xen.img
title CentOS (2.6.18-194.el5)                                ←1 番目 Linux カーネル
       root (hd0,0)
       kernel /vmlinuz-2.6.18-194.el5 ro root=/dev/VolGroup00/LogVol00 rhgb quiet
       initrd /initrd-2.6.18-194.el5.img
```

図 11-2:起動時の GRUB メニュー



Xen ハイパーバイザーが無事起動すれば、Domain 0 も自動的に起動されます。「xm list」コマンドで確認することができます。

▼*ドメインの確認*

```
# xm list
Name                                        ID Mem(MiB) VCPUs State   Time(s)
Domain-0                                     0     1896     4 r-----     22.3
```

11.4 ゲスト OS のインストール（コマンドライン）

Xen では、ドメインは XML 形式の設定ファイルとデータベースで管理されています。これを手動で作成するのは難しいため、python などで作成された管理ツールを利用します。

次は、virt-install コマンドでドメインを作成し、ゲスト OS をインストールする例です。

▼*virt-install によるドメイン OS の作成*

```
# virt-install --prompt
Would you like a fully virtualized guest (yes or no)? This will allow you to run
unmodified operating systems. no                          ←完全仮想化にするか否かを選択
 What is the name of your virtual machine? centos5        ←ドメインの名称
 How much RAM should be allocated (in megabytes)? 512     ←ゲスト OS のメモリの大きさ(MB)
 What would you like to use as the disk (file path)?
/var/lib/xen/images/centos5.img                           ←ゲスト OS のイメージファイル
 How large would you like the disk (/var/lib/xen/images/centos5.img)
to be (in gigabytes)? 5                                   ←ゲスト OS のディスクサイズ (GB)
 What is the install URL?
http://ftp.jaist.ac.jp/pub/Linux/CentOS/5.5/os/i386/      ←インストールイメージがある URL


インストールを開始しています...
```

11.5 ドメインの管理

仮想サーバの管理は、xm コマンドで行うことができます。ここでは、xm コマンドを使ったドメインの管理方法について解説します。

ドメインの起動

ドメインの起動は、xm create コマンドで行います。

```
xm create [-c] <configname> [name=value]...
```

引数の<configname>は、virt-install で指定したドメインの名称です。-c オプションをつけると、該当ドメインのゲスト OS を起動した後、コンソールに接続します。また、引数の[name=value]の部分に設定名と設定値を渡すことで、保存されている設定内容と異なる設定で起動することができます。

ドメインの一覧

ドメインの一覧は、xm list コマンドで取得できます。

```
xm list [--long|--label] [<domain id>...]
```

特に引数を指定しなければ、すべてのドメインの状態を表示します。<domain id>はドメイン番号で、指定した場合には該当のドメインの状態だけを表示します。

▼*Xm list の実行例*

```
# xm list
Name                                        ID Mem(MiB) VCPUs State   Time(s)
Domain-0                                     0     1509     4 r-----    558.7
centos5                                     10      511     1 -b----    207.2
```

表示している項目は次の通りです。

- Name　　ドメインの名称
- ID　　ドメイン ID
- Mem　　割り当てられたメモリ
- VCPUs　　割り当てられた CPU
- State　　ドメインの状態
  - r(run) 稼働中
  - b(blocked)　I/O 待ち
  - p(paused)　停止中

    - s（shutdown）停止処理中
    - c（crashed）　異常停止
    - d（dying）　　強制停止待ち
- Times　　ドメインの稼働時間（秒）

--long を指定すると、ドメインのより詳細な情報を表示します。また、--label を指定すると、ラベルステータスを表示するカラムを追加します。

コンソールの接続

ドメインのコンソールへの接続は、xm console コマンドで行います。

```
xm console <domain id>
```

引数の<domain id>は、xm list コマンドの ID 欄に表示される数値か、Name 欄に表示されるドメイン名です。

ドメインの停止・再起動

ドメインの停止は、xm shutdown コマンドで行います。また、再起動は xm reboot コマンドで行います。

```
xm shutdown [-a|-w] <domain id>
xm reboot [-a|-w] <domain id>
```

引数の<domain id>は、xm list コマンドの ID 欄に表示される数値です。実行すると、ドメインを shutdown/reboot します。ドメイン内にインストールされている仮想 OS が、シャットダウンに対応していない場合にはうまく動作しない場合もあります。-w オプションを指定していると、シャットダウンが終了するまで待ちます。また、-a オプションはすべてのドメインを停止します。

ドメインの一時停止・再開

xm pause コマンドを実行すると、ドメインを一時的に停止状態にすることができます。また、停止したドメインは、xm unpause コマンドで再開することができます。

```
xm pause <domain id>
xm unpause <domain id>
```

ドメインのセーブ・リストア

xm save コマンドを実行すると、ドメインの状態を指定したファイルに保管します。また、xm restore コマンドを実行すると、ドメインの状態を保管したファイルから復元します。

```
xm save <domain id> <file>
xm restore <file>
```

ハイパーバイザーを再起動しなければならない場合などに、ドメインの状態を保管しておき、再起動後に復旧させることができます。

ドメインの強制終了

xm destroy は、ドメインの現在の状態に関わらず強制的に仮想サーバを停止します。これは、通常のコンピュータで電源を抜くのと同じ状態になります。

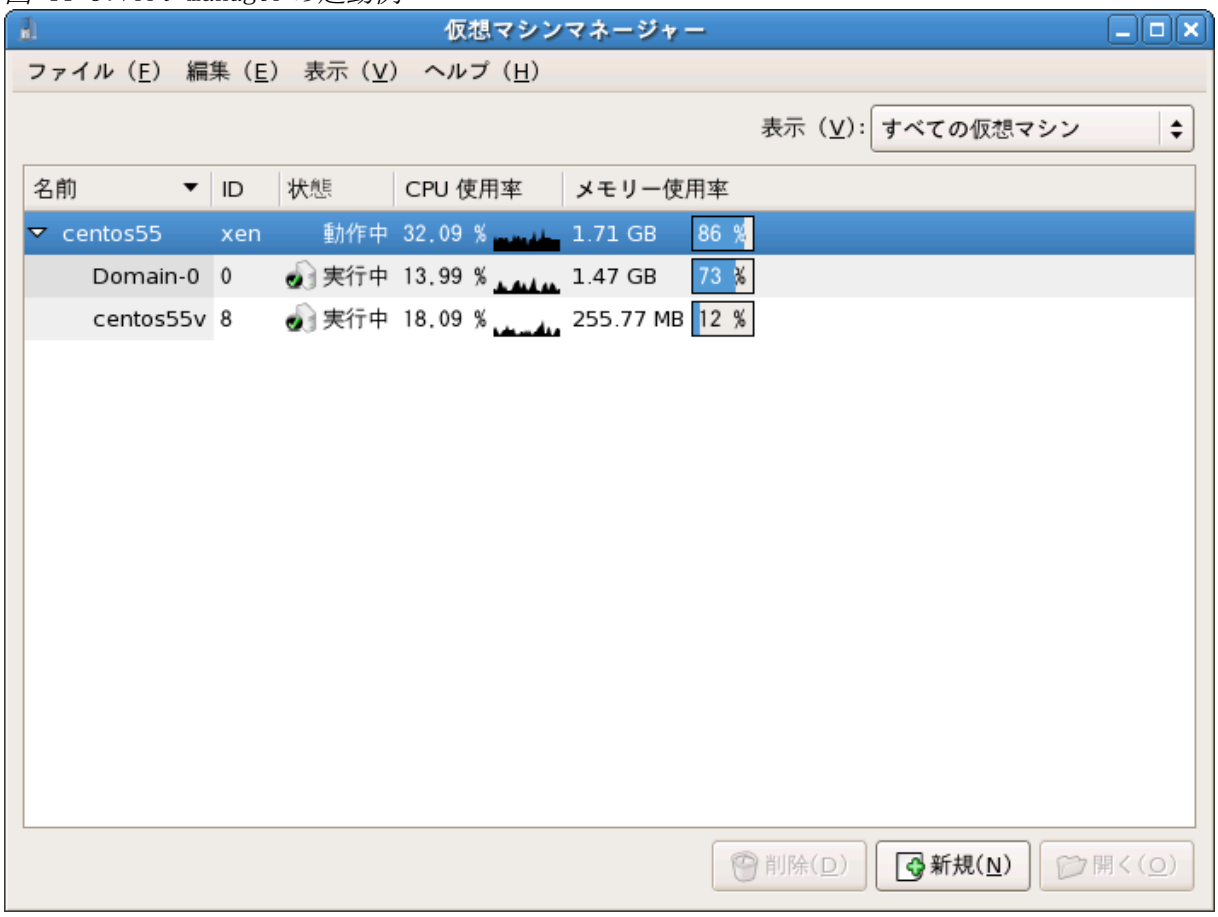
```
xm destroy <domain id>
```

11.6　GUI ツールでの管理

Xen には virt-manager と呼ばれる、Xen 管理用の GUI ソフトウェアがあります。このツールが使える環境では、より簡単かつ詳細にドメインを管理することができます。

図 11-3 は、virt-manager の起動例です。

図 11-3:virt-manager の起動例

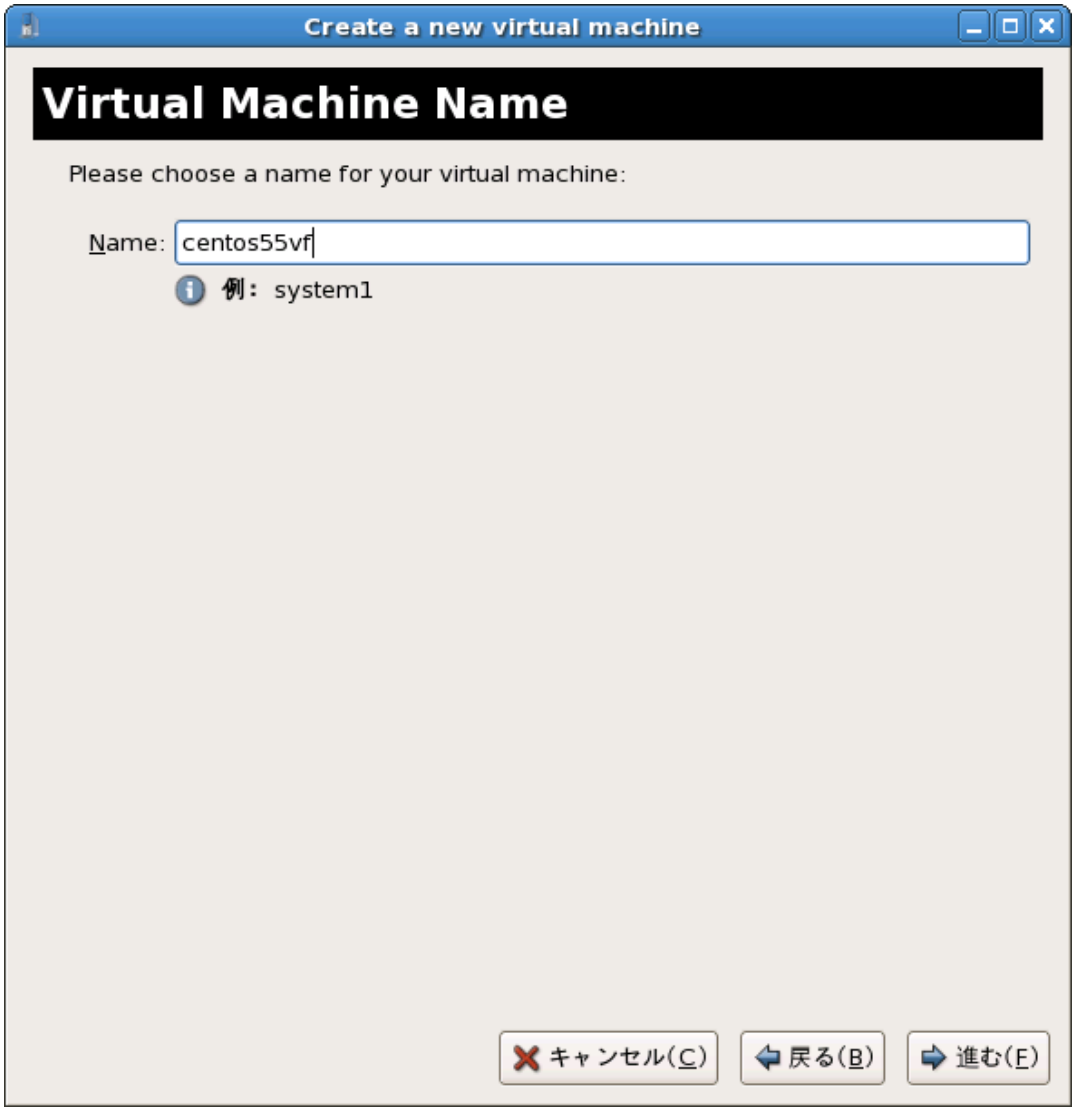


この画面から、新規のドメインの作成、現在のドメインの管理などを行うことができます。

GUI によるインストール

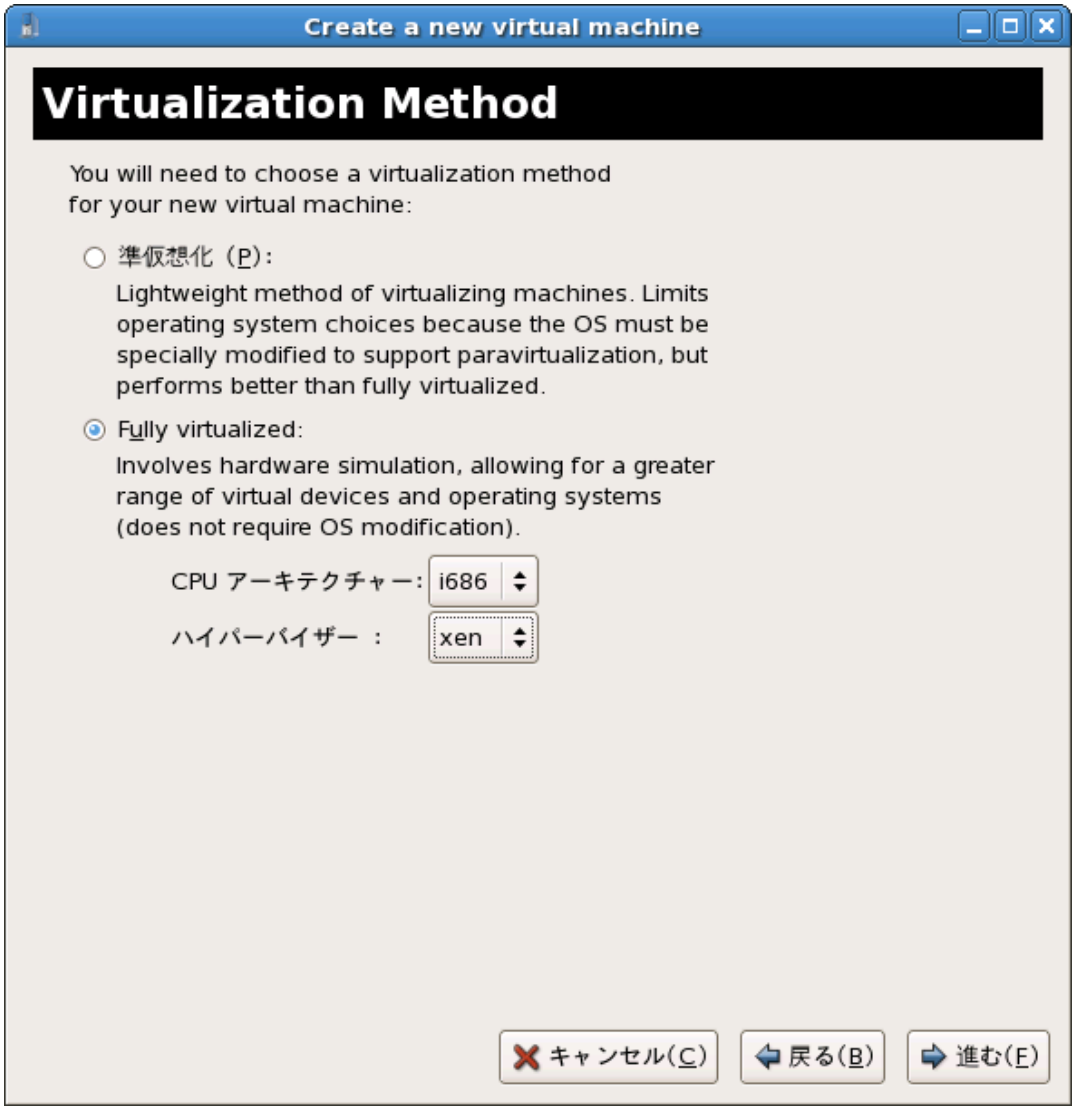
virt-manager の画面の「新規」のボタンを選択すると、「新規の仮想システムを作成」画面が表示され、ここから仮想サーバを作成することができます(図 11-4)。

図 11-4:新規サーバの作成（仮想マシン名の入力画面）



仮想サーバ名を入力したら、「進む」をクリックします。すると、図 11-5 のような仮想化のタイプを選択する画面が表示されます。

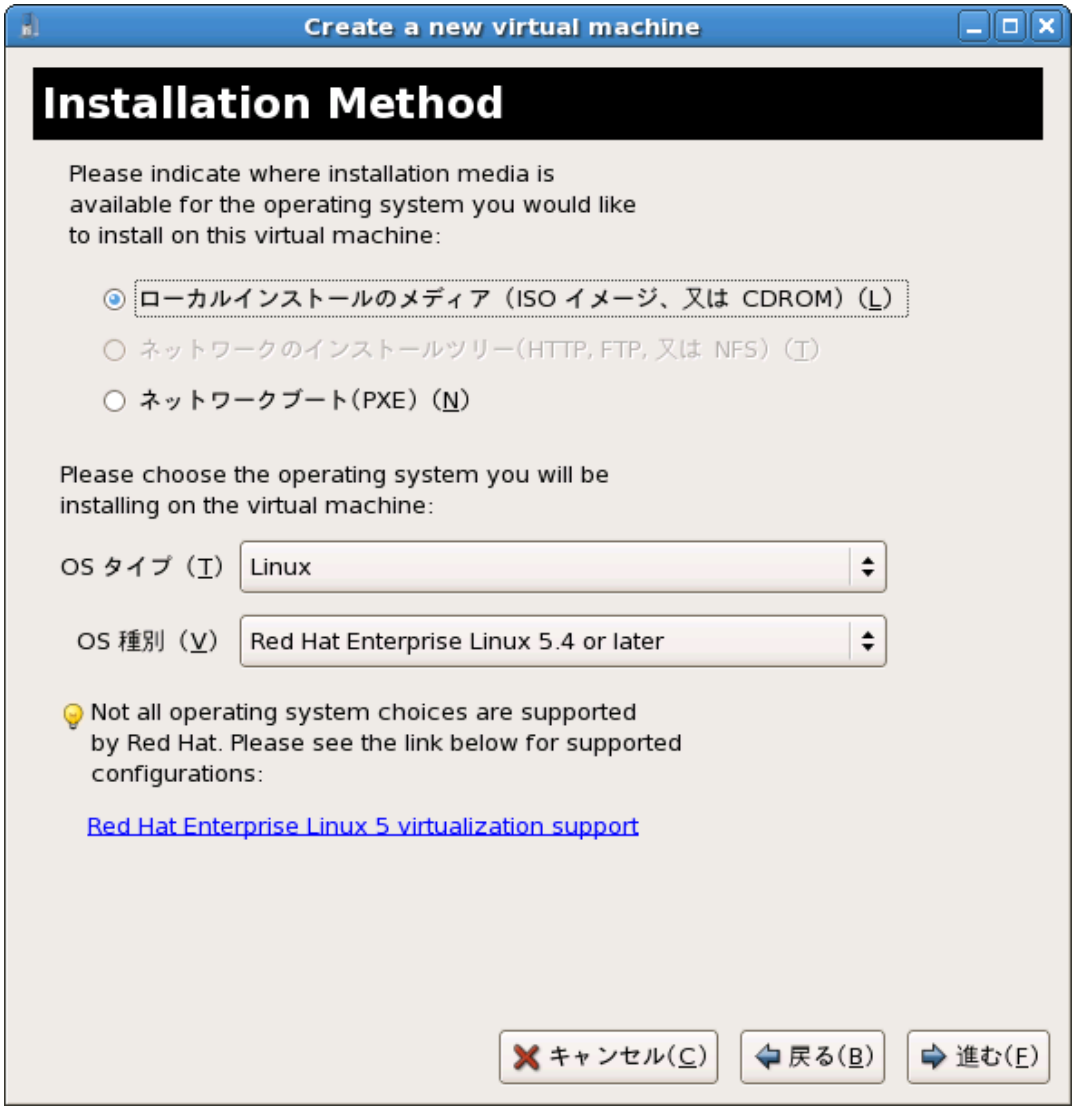
図 11-5:仮想サーバタイプの入力画面



この画面では、準仮想化、完全仮想化（Fully virtualized）を選択することができます。完全仮想化を選んだ場合には CPU アーキテクチャも選択できます。ただし、Intel-VTx のような完全仮想化をサポートするためのハードウェアがない場合には、完全仮想化を選ぶことはできません。

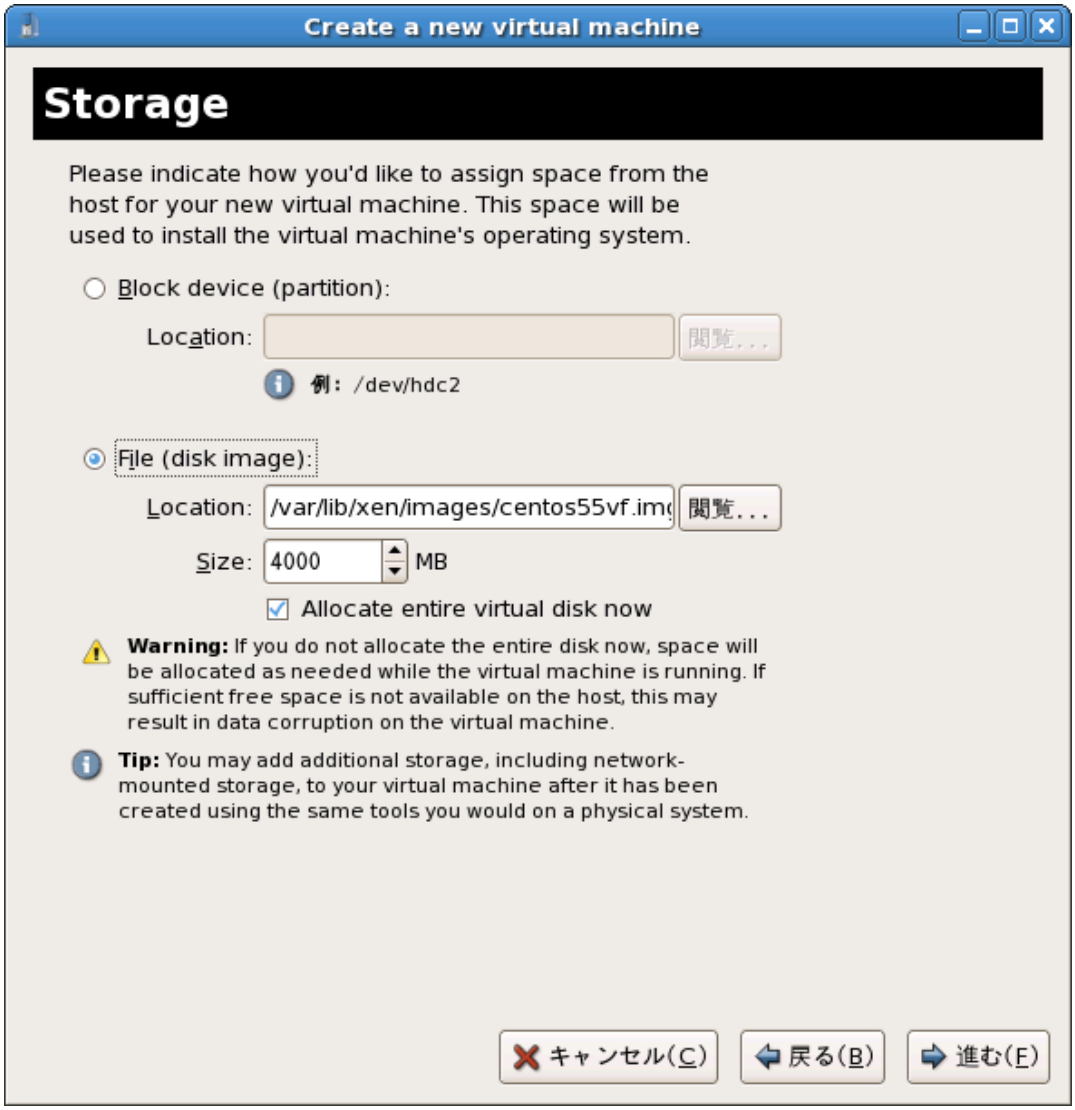
仮想化のタイプを決めて「進む」をクリックすると、図 11-6 のようなインストールメディアの設定画面が表示されます。

図 11-6:インストールメディアの設定画面



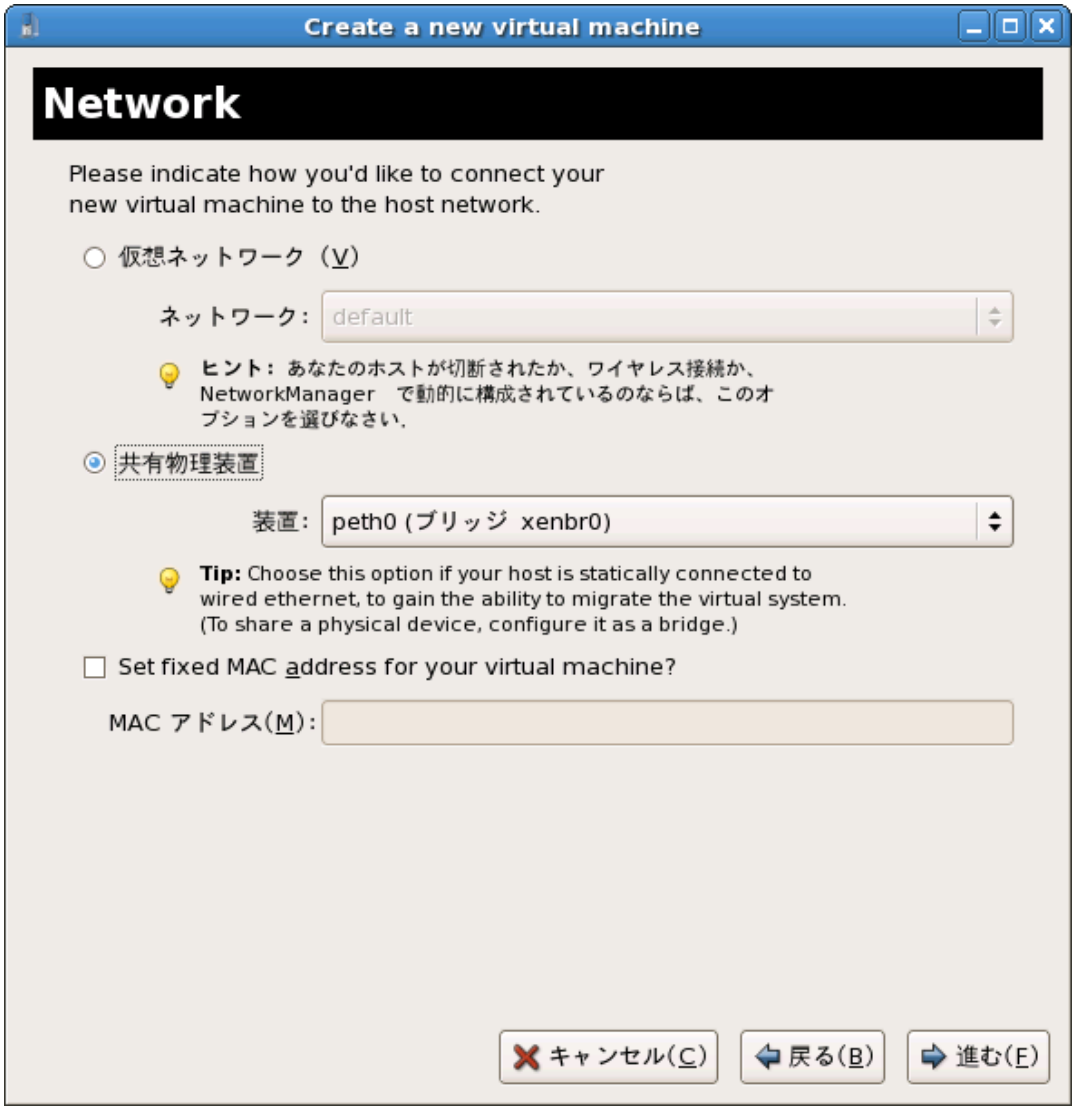
インストールに必要なメディアを選択し、「進む」をクリックします。すると、図 11-7 のようなストレージの設定画面が表示されます。

図 11-7:ストレージの選択



　ストレージファイルの作成場所や、サイズを設定します。設定後、「進む」をクリックすると、図 11-8 のようなネットワークの設定を行う画面が表示されます。

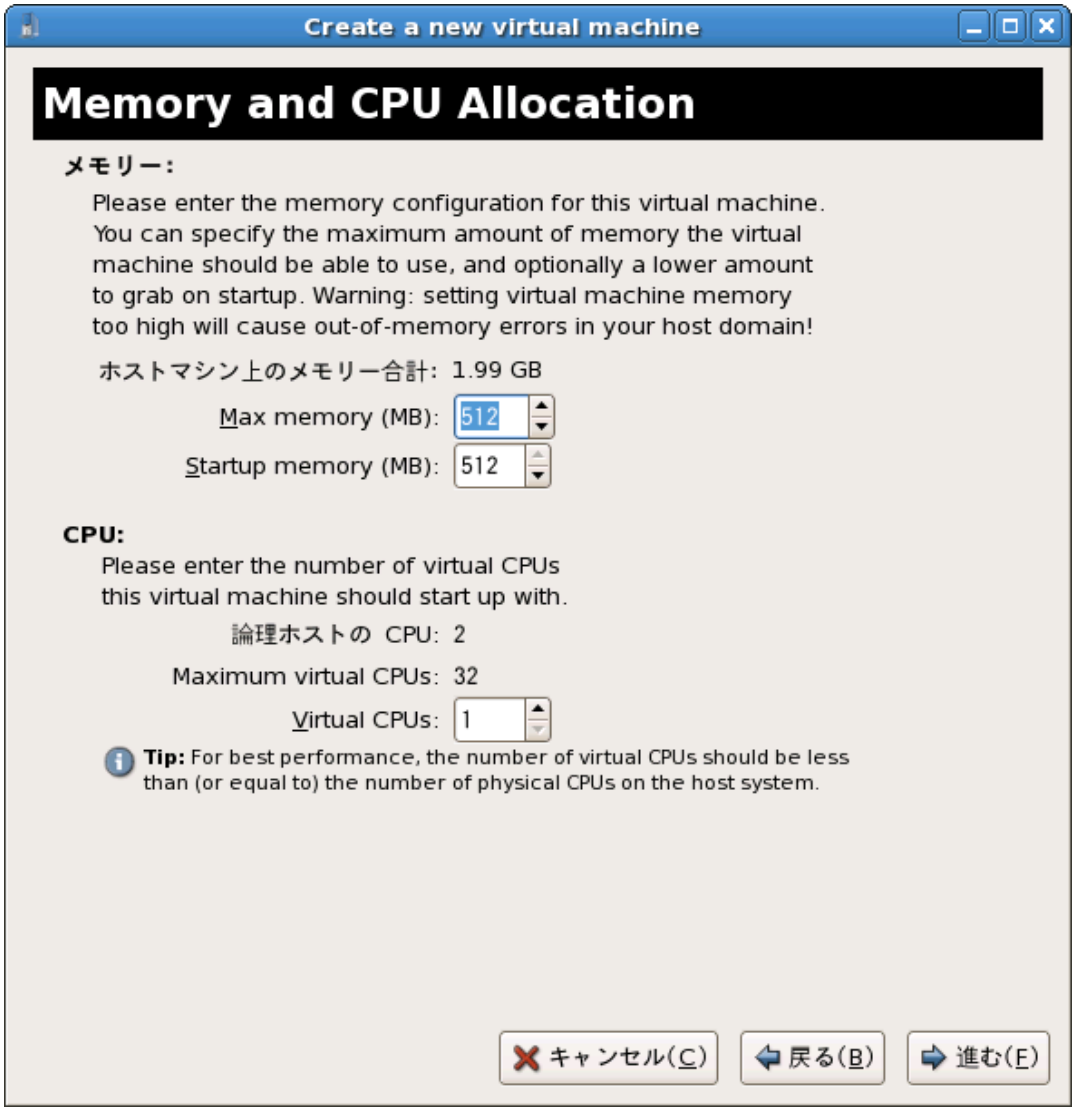
図 11-8:ネットワークの設定



ハイパーバイザーのインストールされているホストの物理ネットワーク（LAN）を共有する場合には、共有物理装置を選択します。物理デバイスが無線 LAN などの動的に変化するネットワークの場合には、「仮想ネットワーク」を選択します。

「進む」をクリックすると、図 11-9 のようなメモリと CPU の設定を行う画面が表示されます。

図 11-9:メモリと CPU の設定画面



この画面では、作成するドメインで使用するメモリの大きさと CPU 数を設定します。この画面で「進む」をクリックすると、インストールのために必要な設定がすべて完了し、図 11-10 のような確認の画面が表示されます。

図 11-10:設定の確認画面

設定の確認ができ、「完了」をクリックすると、自動的に仮想マシンを作成し、OS のインストールが開始されます。内容の変更が必要な場合には、「戻る」をクリックして適切な画面まで戻り、設定を修正します。

12章 仮想サーバを構築する（KVM 編）

本章では、第 10 章で学習した仮想サーバの仕組みのうち、KVM を使って実際に仮想サーバシステムを構築する事例を取り上げます。

12.1 KVM とは

KVM は、Linux Kernel 2.6.20 以降に標準で組み込まれた仮想化の仕組みです。KVM は、カーネルモジュールとして設計されています。KVM には、次のような特徴があります(図 12-1)。

- 特別なハイパーバイザーが不要
  Xen が独自のハイパーバイザーを利用して動作するのと異なり、KVM は Linux カーネルそのものをハイパーバイザーとして動作します。
- Linux プロセスとして動作
  KVM 上のゲスト OS は、一般の Linux プロセスとして動作し、すべてのデバイスへのアクセスは/dev/kvm というドライバを経由して行われます。
- 完全仮想化のみをサポート
  完全仮想化で動作し、Intel VT-x、AMD-V などの機能が必須です。
- QEMU を利用
  オープンソースの仮想マシンエミュレータの QEMU を利用します。
- I/O の性能が優れる
  Xen では、I/O はハイパーバイザー（Xen カーネル）が受け取りますが、Domain 0 によって制御されていました。それに対して、KVM は直接カーネルが I/O を処理します。そのため、I/O 性能の面で優位な構成になっています。

図 12-1:KVM のアーキテクチャ

12.2　ホストOSの設定

KVMは、Linuxカーネルに統合されています。そのため、Linux Kernel 2.6.20以降を採用しているほとんどのLinuxディストリビューションで利用することができます。KVMをサポートしたディストリビューションでは、KVMはパッケージで提供されています。パッケージ（qemu-kvmなど）をインストールするだけで利用することができます。

ホストOSでは、事前にカーネルモジュールkvm、kvm-intel（またはkvm-amd）の読み込みが必要です。

▼*モジュールの読み込み*

```
# modprobe kvm
# modprobe kvm-intel
```

12.3 ゲスト OS のインストール

ゲスト OS のインストールや起動は、qemu の機能を使って行うことができます。次のような手順でゲスト OS を作成します。

- ゲスト OS 用イメージを作成
- ゲスト OS を起動し、インストール

ゲスト OS イメージの作成

ゲスト OS のディスクにあたるイメージを作成します。qemu-img コマンドにサイズとイメージファイル名を指定します。次は、guest1.img という名称で 4GB のイメージを作成した場合の例です。

▼*guest1.img の作成*

```
$ qemu-img create guest1.img 4GB
Formatting 'guest1.img', fmt=raw size=4294967296
```

この例では、イメージをカレントディレクトリに配置しています。利用用途に合わせて、適切なディレクトリに作成してください。なお、qemu-img コマンドはディストリビューションによっては、kvm-img などの名称の場合があります。

ゲスト OS のインストール

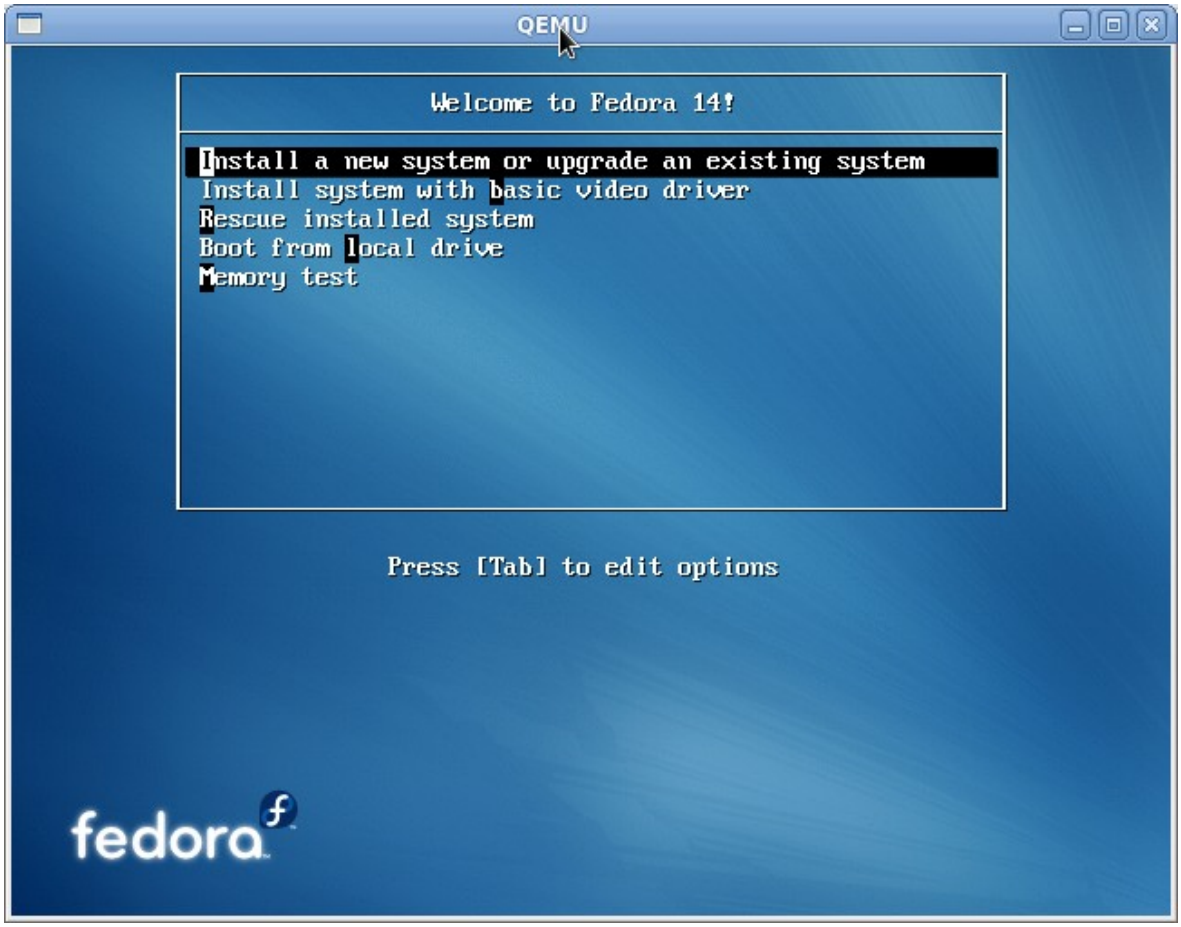
ゲスト OS のインストールは、qemu-kvm コマンドで行います。引数に、作成したゲスト OS のイメージと、インストール用の CD/DVD のパス、メモリなどを指定します。次の例では、ISO ファイルから、メモリ 512MB を指定してインストールを行っています。

▼*ゲスト OS のインストール例*

```
$ qemu-kvm -hda guest1.img -boot d -cdrom Fedora-14-i386-DVD.iso -m 512
```

この例では ISO ファイルを指定していますが、/dev/cdrom などの物理デバイスを指定することもできます。コマンドを実行すると、仮想マシンが起動され、X-Window 上に画面が表示されます(図 12-2)。通常のインストール手順でインストールを行うことができます。なお、qemu-kvm コマンドはディストリビューションによっては、kvm などの名称の場合があります。

図 12-2:インストール画面



ゲスト OS の起動

ゲスト OS の起動方法は、CD-ROM からのブートオプションを外すだけで、先ほどのインストールとほぼ同様です。

▼*ゲスト OS のインストール例*

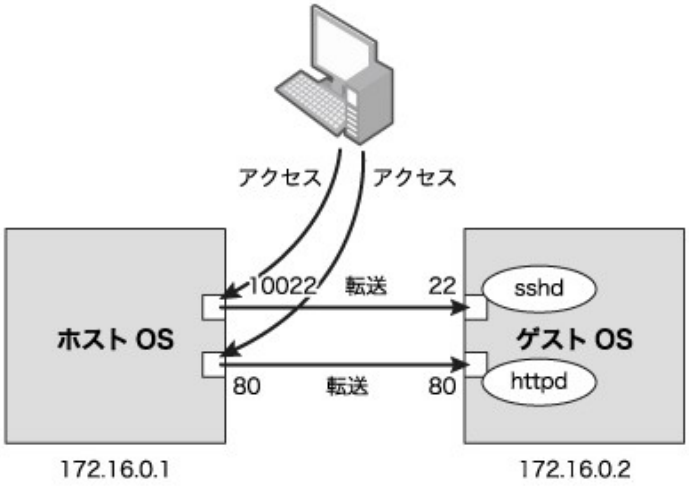
```
$ qemu-kvm -hda guest1.img -m 512
```

なお、この起動方法では仮想サーバには eth0 というネットワークが割り当てられるものの、実際にそのネットワークに接続することができません。ネットワークへ接続するには、qemu-kvm のオプションで利用用途に合わせたネットワークの設定が必要です。

12.4 ポートフォワーディングの構成

ゲスト OS が、外部との通信で限定的なサービスだけを提供すれば良い場合には、ホスト OS のポートからゲスト OS へ通信をリダイレクトすることができます(図 12-3)。

図 12-3:ゲスト OS の特定のポートだけを接続する



ゲスト OS の起動時に次の例のようにオプションを設定することで、このような構成を実現することができます。

▼*ポートフォワーディングの設定*

```
$ qemu-kvm -hda guest1.img -net nic ¥
-net user,net=172.16.0.0/16,host=172.16.0.1,¥
hostfwd=tcp:127.0.0.1:2222-172.16.0.2:22,hostfwd=tcp::80-172.16.0.1:80
```

最初の「-net nic」は、ゲスト OS にネットワークインタフェースを 1 つ割り当てる設定です。そして、「-net user」以降の長い引数がその NIC に対する設定です。

- net=172.16.0.0/16
  ゲスト OS 用ネットワークの設定。指定しない場合には、10.0.2.0/8 となります。
- host=172.16.0.1
  ホスト OS 側の IP アドレス。指定しない場合には、ゲスト OS 用ネットワークの最初の IP アドレス（10.0.2.1）となります。
- hostfwd=tcp:127.0.0.1:2222-172.16.0.1:22
  ホスト OS の TCP ポート 2222 に届いたパケットをゲスト OS のポート 22 へ転送する設定
- hostfwd=tcp::80-172.16.0.1:80
  ホスト OS の TCP ポート 80 へ届いたパケットをゲスト OS のポート 80 へ転送する設定。

この例では、「¥」をつけて改行をつけて表示しています。実際には、引数の間にスペースを入れないで「,」区切りで指定する必要があります。hostfwd は次のような書式で指定します。

```
hostfwd=[tcp|udp]:[hostaddr]:hostport-[guestaddr]:guestport
```

[hostaddr]を省略するとホスト OS のすべてのインタフェースのポートに適用されます。[guestaddr]を省略するとゲスト OS 用ネットワークの 2 番目の IP アドレス（10.0.2.2）が適用されます。

なお、古いバージョンの KVM では、hostfwd ではなく-redir というオプションを利用していました。次の例は、上記と同様に 2222 ポートをゲスト OS の 22 番ポートへフォワーディングする設定です。

▼*古い KVM でのポートフォワーディング*

```
# qemu-kvm -hda guest2.img -net nic -net user,hostname=172.16.0.1 -redir
tcp:2222:10.0.2.15:22
```

Hostname に設定されている 172.16.0.1 は、ホスト OS 側の IP アドレスで、この例では 172.16.0.1:2222 への通信を 10.0.2.15:22 へフォワーディングします。-redir で指定を行う場合には、ゲスト OS 用のネットワークは 10.0.2.0/8 の固定となります。そのため、ゲスト OS には 10.0.2.15 など、このネットワークに所属する IP アドレスをつけなければなりません。

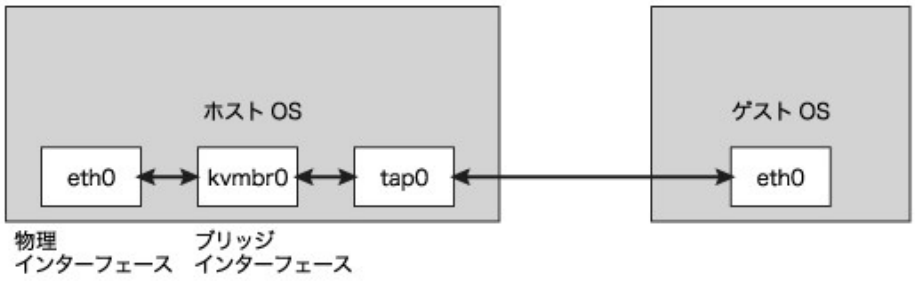
12.5 ブリッジネットワークの構成

ホスト OS にゲスト OS のパケットを転送するブリッジの役割を持たせることで、ゲスト OS がホスト OS の NIC をあたかも共有しているかのように動作させることも可能です。そのためには、次のような設定を行う必要があります。

- 必要なパッケージをインストール
  brctl コマンドなどがインストールされていない場合には、ブリッジを制御するために必要なパッケージ（bridge-utils など）をインストールします。
- ブリッジ設定
  KVM のゲスト OS からネットワークを利用するために、ブリッジの設定を行います。
- ゲスト用インタフェース設定
  ゲスト OS の起動時に、ゲスト OS のネットワークをブリッジに接続する設定を行います。

ここでは、図 12-4 のようなシステム構成を例にとって解説します。

図 12-4:ゲスト OS をブリッジ接続する



ブリッジの設定

ゲスト OS からネットワークが利用できるようにブリッジの設定を行います。例えば、eth0 に対するブリッジ(kvmbr0)を設定する場合には、まず eth0 の設定をコピーしてブリッジ用の設定ファイルを作成します。

```
# cd /etc/sysconfig/network-scripts/
# cp ifcfg-eth0 ifcfg-kvmbr0
```

ifcfg-eth0 を修正しブリッジ設定を追加します。また、ifcfg-kvmbr0 にはデバイス名とデバイスタイプの設定を行います。次は、その修正例です。

▼*/etc/sysconfig/network-scripts/ifcfg-eth0 の修正例*

```
DEVICE=eth0
BOOTPROTO=static
```

```
HWADDR=5C:26:0A:09:F7:AE
ONBOOT=yes
IPADDR=192.168.3.200
NETMASK=255.255.255.0
BRIDGE=kvmbr0                                                    ← 追加
```

▼*/etc/sysconfig/network-scripts/ifcfg-kvmbr0 の修正例*

```
DEVICE=kvmbr0                                                    ← 修正
BOOTPROTO=static
HWADDR=5C:26:0A:09:F7:AE
ONBOOT=yes
TYPE=Bridge                                                      ← 追加
IPADDR=192.168.3.200
NETMASK=255.255.255.0
```

複数のネットワークカードに対応させるためには、必要に応じてこの設定を追加する必要があります。設定が完了しましたら、network サービスを再起動します。

▼*ネットワークサービスの再起動*

```
# service network restart
インターフェース eth0 を終了中:                          [  OK  ]
ループバックインターフェースを終了中                      [  OK  ]
ループバックインターフェイスを呼び込み中                  [  OK  ]
インターフェース eth0 を活性化中:                        [  OK  ]
インターフェース kvmbr0 を活性化中:                      [  OK  ]
```

ゲスト OS 用インタフェース制御スクリプト

ゲスト OS が起動するときに、ゲスト OS のネットワークインタフェースをブリッジデバイスに関連付ける設定をする必要があります。起動時の設定を/etc/qemu-ifup、終了時の設定を/etc/qemu-ifdown に行います。次は、インタフェース制御スクリプトの例です。

▼*/etc/qemu-ifup*

```
#! /bin/sh
/sbin/ifconfig $1 0.0.0.0 promisc up
/usr/sbin/brctl addif kvmbr0 $1
```

▼*/etc/qemu-ifdown*

```
#! /bin/sh
/usr/sbin/brctl delif kvmbr0 $1
/sbin/ifconfig $1 0.0.0.0 down
```

ゲスト OS の起動

ゲスト OS の起動時に TAP を割り当てます。ブリッジの設定(図 12-5)などを行う必要があるため、root ユーザで起動しなければなりません。

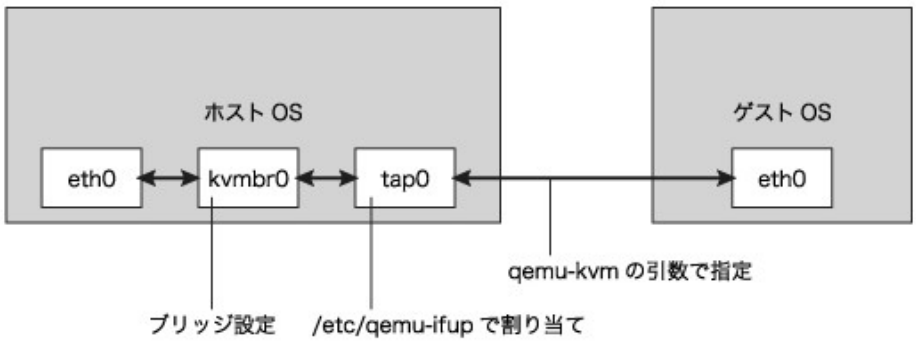
▼ *ブリッジでの起動*

```
# qemu-kvm -hda guest1.img  -m 512 -net nic,vlan=0 -net tap,vlan=0,ifname=tap0
```

-net 以降の引数は次のような意味です。

- -net nic,vlan=0
  ゲスト OS にネットワークインタフェースを 1 つ割り当て、vlan 0 番を割り当てます。「vlan=0」は省略することができます。
- -net tap,vlan=0,ifname=tap0
  ゲスト OS に TAP を割り当てます。TAP は、vlan 0 番に接続し、デバイスの名称は tap0 とします。「vlan=0」、「ifname=tap0」は省略することができます。

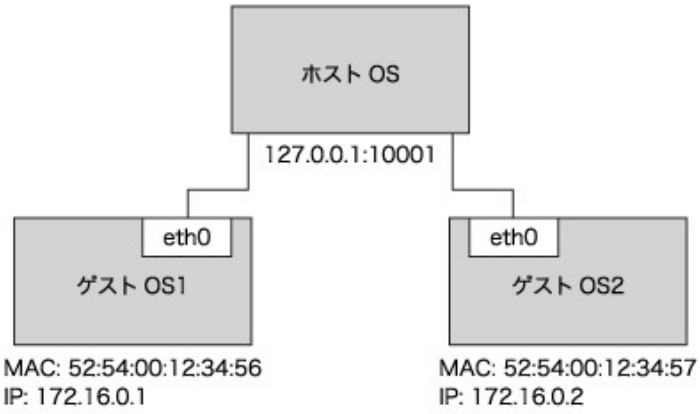
図 12-5:ゲスト OS をブリッジ接続する



12.6 ゲストOS間通信

kvmのバージョン0.10.6以降では、ゲストOSとゲストOSのネットワークインタフェースを接続し、ゲストOS同士が通信できるように設定することができます。ここでは、図12-6のようなシステム構成を例にとって解説します。

図 12-6:ゲストOS間通信



ゲストOSのMACアドレスは、デフォルトでは52:54:00:12:34:56が割り当てられます。ゲストOSを複数立ち上げる場合には、明示的にMACアドレスを指定して分ける必要があります。

ゲストOSの起動

次は、ゲストOS1をMACアドレス52:54:00:12:34:56で起動し、127.0.0.1のポート10001のソケットで接続を待つようにして起動しています。

▼*ゲストOS1の起動*

```
$ qemu-kvm -hda guest1.img -net nic,macaddr=52:54:00:12:34:56 ¥
-net socket,listen=127.0.0.1:10001
```

ゲストOS2は、MACアドレスを52:54:00:12:34:57と変更して起動し、127.0.0.1のポート10001のソケットへ接続するように起動します。

▼*ゲストOS2の起動*

```
$ qemu-kvm -hda guest2.img -net nic,macaddr=52:54:00:12:34:57 ¥
-net socket,connect=127.0.0.1:10001
```

なお、ゲストOS間通信はバグのため利用できない場合があるようです。

12.7 ゲスト OS の管理

KVM では、ゲスト OS は通常のプロセスと同様に管理することができます。つまり、起動後の停止は kill コマンドなどで行うことができます。例えば、起動しているゲスト OS を探したい場合には、次のように ps コマンドで確認することができます。

```
$ ps -C qemu-kvm -f
UID        PID  PPID  C STIME TTY          TIME CMD
root      8639  7541  2 19:02 pts/0    00:00:07 qemu-kvm -hda guest1.img -net ni
root      8648  7541  2 19:03 pts/0    00:00:06 qemu-kvm -hda guest2.img -net ni
```

ゲスト OS を強制終了したい場合には、kill コマンドなどで停止することも可能です。

12.8　libvirtを使った管理

libvirtをサポートしているシステムでは、Xenと同様にvirt-installを使ったゲストOSのインストールや、virt-managerを使ったゲストOSの管理を行うことができます。こうしたゲストOS管理ツールを使うと、qemuを意識することなくKVMを利用することができます。

12.8.1　libvirtの利用準備

libvirt、python-virtinst、virt-manager、virt-viewerなどのパッケージをインストールすることで、virt-installやvirt-managerを利用することができます。パッケージをインストールしたら、利用のための準備を行う必要があります。

libvirtdの起動

CentOS 5などRedHat系のディストリビューションではlibvirtdを起動するだけで、kvm、kvm-intelなどのカーネルモジュールの読み込みが自動的に行われます。

```
# service libvirtd start
Starting libvirtd daemon:                                  [  OK  ]
```

ブリッジの設定

前節の解説にしたがって、ブリッジの設定を行います。本節では、kvmbr0というブリッジインタフェースを作成した場合を例にとって解説します。

12.8.2　ゲストOSのインストール

virt-installを使って、対話形式でゲストOSをインストールすることができます。ただし、対話形式でも、ネットワークの指定はコマンドラインで行う必要があります。次のようにブリッジのインタフェース名を設定します。

▼*ゲストOSのインストール*

```
# virt-install --prompt --network bridge=kvmbr0
Would you like to use KVM acceleration? (yes or no) yes
 What is the name of your virtual machine? vhost1
 How much RAM should be allocated (in megabytes)? 512
 What would you like to use as the disk (file path)? vhost1.img
 How large would you like the disk (vhost1.img) to be (in gigabytes)? 4
 What is the install CD-ROM/ISO or URL? /home/admin/kvm/Fedora-14-i386-DVD.iso
```

12.8.3　ゲスト OS の管理

virt-install で作成したゲスト OS は、virsh コマンドなどを利用して管理することができます。

ゲスト OS の起動

インストールが完了したゲスト OS は、virsh start コマンドを使って起動します。引数としてインストール時に指定した仮想マシン名を指定します。次は、vhost1 を起動する例です。

▼*virsh によるゲスト OS の起動*

```
# virsh start vhost1
ドメイン vhost1 が起動されました
```

ゲスト OS への接続

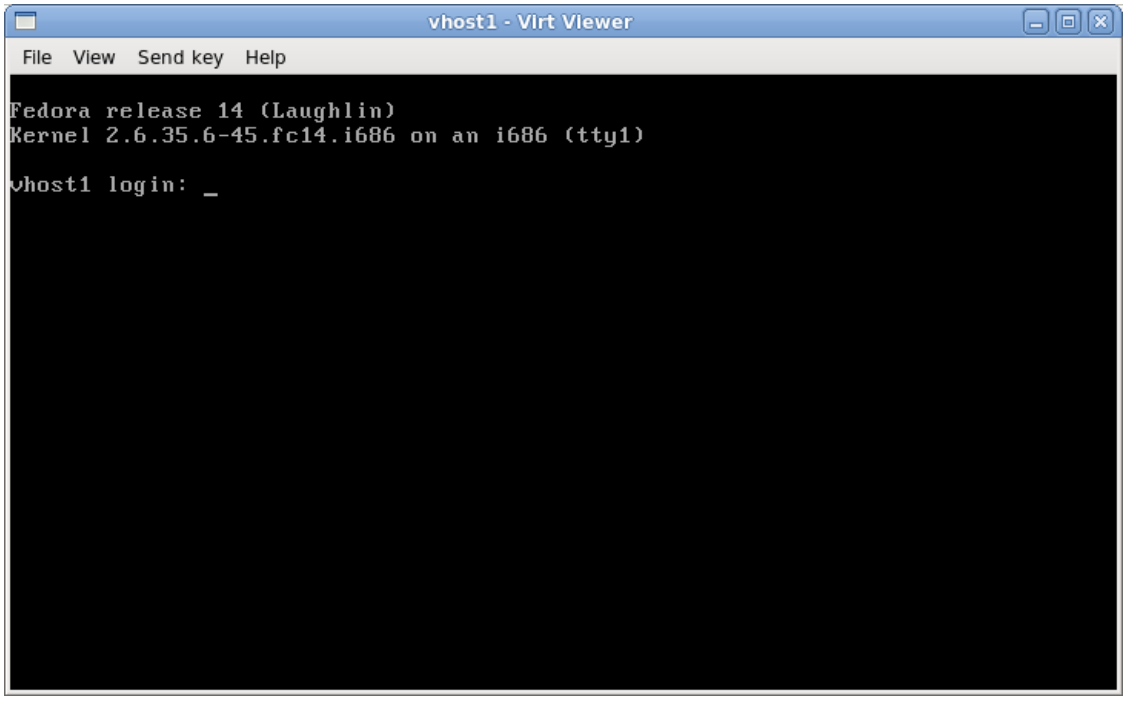
virsh コマンドには、connect という引数が用意されていますが、KVM では正しく動作しない場合があります。コンソールへの接続は、virt-viewer コマンドを使って行います。引数としてインストール時に指定した仮想マシン名を指定します。次は、vhost1 のコンソールを表示する例です。

▼*virsh によるゲスト OS コンソールへの接続*

```
# virt-viewer vhost1
```

X-Window 上にコンソール画面が表示されます(図 12-7)。

図 12-7:virt-viewer で表示されるコンソール画面



ゲストOSの一覧

ゲストOSの一覧は、virsh listコマンドを使って取得することができます。各ゲストOSの状態が表示されます。

▼*virshによるゲストOSの一覧の取得*

```
# virsh list
 Id 名前                 状態
----------------------------------
  6 vhost1                実行中
```

ゲストOSの一時停止・再開

ゲストOSの一時的に停止はvirsh suspendコマンドを使って、再開はvirsh resumeコマンドを使って行うことができます。引数にはvirsh listコマンドで表示されたゲストOSのIDを指定します。

▼*virshによるゲストOSの一時停止と再開*

```
# virsh suspend 6
ドメイン 6 は一時停止されました
# virsh resume 6
ドメイン 7 が再開されました
```

ゲストOSの保存・リストア

ゲストOSの保存はvirsh saveコマンドを使って、リストアはvirsh restoreコマンドを使って行うことができます。引数にはvirsh listコマンドで表示されたゲストOSのIDと保存ファイル名を指定します。

▼*virshによるゲストOSの一時停止と再開*

```
# virsh save 6 /var/kvm/images/vhost1.sv
ドメイン 6 は /var/kvm/images/vhost1.sv に保存されました
# virsh restore /var/kvm/images/vhost1.sv
ドメインは /var/kvm/images/vhost1.sv から復元されました
```

ゲストOSの削除

インストールしたゲストOSを削除したい場合には、virsh undefineコマンドを使います。引数には、ゲストOS名を指定します。

▼*virshによるゲストOSの一時停止と再開*

```
# virsh undefine vhost1
ドメイン vhost1 の定義は削除されています
```

この例のようにゲスト OS の定義を削除しても、ゲスト OS のイメージは削除されません。イメージの削除が必要な場合には、手動でファイルを削除しなければなりません。

12.8.4　virt-manager による管理

Xen と同様に、virt-manager によって GUI で管理することも可能です。図 12-8 は、virt-manager の実行例です。

図 12-8:virt-manager の管理画面

www.lpi.or.jp