from Linux Seminar

Diskless Cluster Linux

「Linux Seminar」という勉強会が月に1回の割合で開催されている（詳細についてはコラム参照）。しかし、開催地が東京であるため、全国のLinuxユーザーがこのセミナーに参加することはできない。そこで、本誌では今後、Linux Seminarで行われた発表のうちいくつかを誌面上で再現し、その雰囲気だけでも全国のLinuxユーザーにお届けしたいと思う。今回は、茨城大学でディスクレスシステムからクラスタリング環境を利用して数値計算を行っている、野澤 恵氏に「Diskless Cluster Linux」を紹介していただく。また、後半には番外編としてディスクレスシステムの実装方法を坂田 智之氏に解説していただく。

はじめに

先日、Linux Seminar（http://www.linet.gr.jp/lswg/）の第9回に講師として出席しました。初めて Linux 関係の場で話す機会を持てたことをとても光栄と思い、関係者に深く感謝しています。講演自体は私の稚拙な発表にもかかわらず、聴衆の方々が熱心であり、とても有意義な時間が過ごせ、非常にうれしかったです（終ったあとの宴会がまたよかったです）。

今回ここで記事にするのは「Diskless Cluster Linux」（Clusterは並列計算機の意味）ですが、誌面に限りがあるので、私が重要だと思う点についてのみ紹介します。もし読者の方で読み足りないという方がおられましたら、編集部まで要望を出してくださいい。

ディスクレスシステムとは

「Diskless」という言葉を聞き慣れないと思う読者もいるかもしれませんが、その名のとおり「ディスクがない」という意味です。ディスクとは、フロッピードライブやハードディスクのことを指していますが、本稿では「フロッピードライブはあるが、ハードディスクがまったくない」（ハードディスクがあっても使用しない場合も含む）システムを想定しています。

では、なぜディスクレスシステムを使うのか？　その利点は「堅牢性」にあります。ハードディスクという回転系の動作部品は、いつかは回転が止まる（壊れる）という欠点を抱えています。もちろん、CPUの非動作部品も壊れる可能性はありますが、ハードディスクに比べてその可能性は小さいのです。そこで、できる限り回転する物をマシンから排除することで、堅牢性の高い環境を構築することが可能になるのです。

このディスクレスシステムを構築するには、まず今までハードディスク上にあったプログラムやデータをネットワーク経由でサーバーと共有するように設定します。そのため、ディスクレスシステムではプログラムやデータを自分で保持することができず、ネットワークを介してサーバーに常時依存するという欠点ともいうべき特徴があります。しかし、この特徴は逆にサーバーだけでプログラムやデータの一元管理ができるという長所も併せ持つことになります。たとえば、なんらかの理由でディスクレスシステムのマシンがハングアップしたとしても、サーバーさえ堅牢であれば再起動するだけで問題はありません。

また、あるソフトをすべてのマシンにインストールすることになった場合、ディスクレスシステムでないと1つ1つにインストールするか、自動配布するような特別な仕組みが必要になります。しかし、ディスクレスシステムの環境なら、サーバーに一度インストールするだけでディスクレスシステムからはすぐに利用できるようになります。

ディスクレスシステムは個人使用とは違う世界、たとえば大規模システムや数値計算用の並列計算機群の構築や

Webサーバーの分散処理のイメージがありますが、個人使用の環境でも使えるはずです。たとえば、Windows 98が動いているパソコンに、Linuxもインストールしようとすると、Windows 98のバックアップを取って、ハードディスクのパーテーションを切り直してからインストールしなければなりません。しかし、Linuxが動いているマシン（Linux Box）とネットワーク環境さえ構築してあれば、Linux Boxにちょっとしたサーバーの設定を行い、Windows 98のパソコンをフロッピーで起動させれば、ディスクレスシステムの環境がすぐに手元に実現できます。また、Windows 98の設定そのものにはまったく手をつけないので、安心して使用することができます。

筆者のLinux Diskless Clusterの環境

現在、私の所属する講座では8台のPC/AT互換機で、ディスクレスシステムでクラスタリングされたLinuxを利用するシステム「Linux Diskless Cluster」を作成し、並列シミュレーションを行っています。

私は6年ほど前よりLinuxを使用しています。理由はLinuxになにかしら既存のオペレーティングシステムにない新しさがあって、その魅力にひかれたからです。数値計算を専門としていたために、個人所有の数値計算用のマシンで、自分のプログラムを無制限に動作させたいとずっと考えていました。そして今回のLinux Clusterは個人でスーパーコンピュータを所有するという夢の実現であり、長年の悲願が達成つつあると感傷に浸っています。

私が使用しているLinux Clusterは、通常の学内LANのTCP/IPのネットワークに接続されています（**写真1**）。しかし、なにせ8台もあるので、置き場所に困っています。一度きれいにロッカーに並べたことがあったのですが、放熱の問題のために隙間を開ける必要があり、現在はあまりきれいな状態になっていません。

最先端Linux Cluster環境

Beowulf Project（http://www.beowulf.org/）が、Linux Clusterによってスーパーコンピュータを作ることを目指しており、昨年CPUにAlphaを用いて**写真2**のような大規模なLinux Clusterを作成しました（http://cnls.lanl.gov/avalon/）。このシステムは、1999年6月10日現在で世界のスーパーコンピュータの中で160位にランクされています（http://www.netlib.org/benchmark/top500.html）。日本国内でも同様のプロジェクトがあります（http://www.rwcp.or.jp/）。

このように、Linux Clusterは、安価でコストパフォーマンスの高いシステムを作成できるため、非常に注目が集まっています。なぜ、このようにクラスタリングシステムがもてはやされているのかというと、スーパーコンピュータなどの専用機に比べてCPUなどの部品が大量に生産されているため安価で、修理も部品を交換するだけなので手軽だからです。また、それほど大がかりな電源や冷却のシステムは必要がないという利点もあります。

さらに、私が注目しているのは、計算のための実効メモリの量です。スーパーコンピュータなどの専用機では、メモリの上限がアーキテクチャによって決まってしまうことが多いのに対し、Linux Clusterでは1台あたり1Gバイト以上のメモリを搭載させることが可能です。また、パソコン用のメモリは安価であるため、巨大な実効メモリを搭載させたクラスタリングシステムを作成することが可能だと

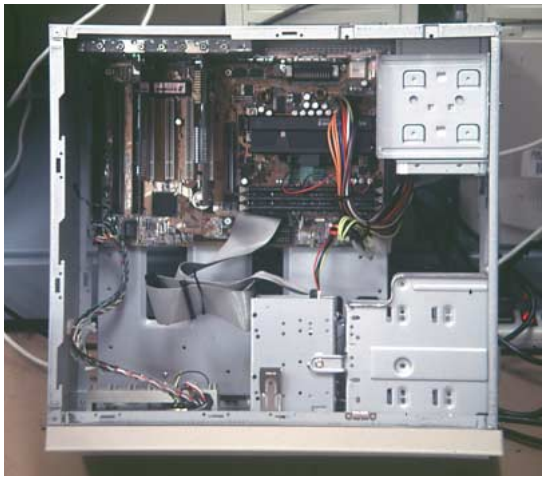


写真1　筆者のLinux Cluster環境「apollo」

いう利点もあります。

クラスタリングのための並列化ライブラリ

ただし、Linux Clusterといっても並列用のアプリケーションが入っているわけでないので、そのままでは並列化の恩恵にあずかることはできません。そこで、並列用のソフトウェアを「自力で」開発する必要があります。これらを実現するために並列ライブラリが提供されています。このライブラリは、基本的にはプログラムを起動させたあと、ネットワークを介してデータ転送の手順を決めているだけです。しかし、CPUやOSが異なるとデータの表現方法が異なる場合があり（エンディアンなど）、並列化ライブラリはソフトウェアによってその差異を吸収しています。そのため、具体的なプログラムのプロセスの動作やネットワークでのデータのやり取りについて、ユーザーは考慮する必要がないように作られています。

並列化ライブラリの代表に、PVM（Parallel Virtual Machine）とMPI（Message Passing Interface）があります。これらは、FORTRANのサブルーチン、またはC/C++の関数として、外部ライブラリを提供しています。たとえば、FORTRANではPVM、MPIを呼び出すサブルーチンを記述し、コンパイル時にリンクさせることで並列アプリケーションを作成することができます。

現状では既存のプログラムを自動的にPVM、MPI化す

写真2　最先端のLInux Cluster「Avalon」

Column

PVMとPVI

並列計算とは、ある1つの処理を複数のプロセッサで協調して演算させることで、高速性、信頼性、拡張性の向上を図るものである。

既存のプログラムを自動的に並列化する方法もあるが、実用に十分な速度を達成するものは、当分は実現しそうにない。そのため、既存のプログラムを手動で並列化するのが一般的である。この際に利用される並列化ライブラリとして、PVM、MPI、HPF（High Performance Fortran）、専用のライブラリなどがある。ここでは、特にPVMとMPIについて説明する。

PVM（http://www.epm.ornl.gov/pvm/pvm_home.html）は、1991年にオークリッジ国立研究所とテネシー大学で開発された並列化ライブラリである。PVMは、設定により各々の計算機の実行速度比を与えることができ、処理速度の異なる計算機間でも有効にその能力を引き出すことができる。また、PVMは仕様の定義だけではなく、実際のライブラリを開発しているため、対応するOSが多い。管理者でなくてもインストールでき、使用者のために利便が図られている。

PVMは、並列のプログラムに関しては、マスター／スレーブ方式（マスターのプログラムが親として動作し、子であるスレーブのプログラムを制御して計算を行なう方式）とホストレス方式（各々のプログラムには親子の関係がないもの）の2つをサポートしている。また、PVMは次に述べるMPIより先に開発されたため、使用者が多く開発状況などは活発である。関数もMPIよりずっと少なく、基本的なものばかりである。

MPI（http://www.mpi-forum.org/¦）は、MPI Fourum（MPIF）によって、1992年より標準仕様の定義や検討がなされている並列化ライブラリである。MPIFでは、基本的にメッセージ転送などの通信機能の実装を定義しているだけなので、実際のライブラリなどの実装は実装者に任されている（http://WWW.ERC.MsState.Edu/mpi/implementations.htmlでは、さまざまな実装が紹介されている）。

並列プログラムは、基本的にホストレス方式を基に実装されているため、マスター／スレーブ方式のものについては工夫が必要になっている。PVMに比べて後発であるため使用者が少なく、開発状況などがさほど活発な状況でない。しかし、関数の数はMPI-1では130あり、複雑な処理が1つの関数ですませるようになっている。現在は、MPI-1の次の段階のMPI-2が公開され、MPI-1では取り込めなかったPVMの機能を含むようになっている。

本文でも述べたように、両者とも基本的にデータの転送とサーバーからクライアントへのプロセスの起動や終了を行うだけであり、並列プログラムとしての基本的な部分は同じである。

（野澤 恵）

るための万能変換ツールはないため、基本的に人間が既存のプログラムを「手動」でPVM、MPI化するしかありません。そのため、プログラムの書き換えの手間が面倒だとPVM、MPIが敬遠する人たちもいます。

筆者の環境は、処理速度やメモリなどが同一のパソコンで使用しているため、ファイルの書き出しなどのI/O処理を工夫すれば、元のプログラムをMPIの得意なホストレス方式に簡単に直せるので、主にMPIを使用しています。世の中の流れも、PVMからMPIへのシフトがあるように思われます。それぞれの特徴を**グラフ1～4**に示します。

一般の応用例として

ここまで、MPI、PVMで並列専用のプログラム開発について述べてきましたが、これは一般的な読者にはあまり身近な例とはいえません。そこで、音楽用CDから作成したWAVファイルをMP3形式に圧縮する場合について、パフォーマンスを調べてみました。

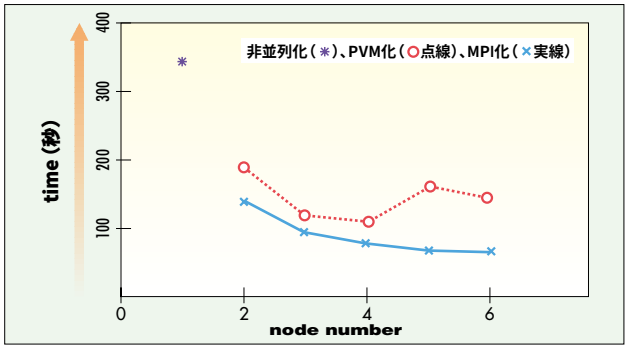
今回は、1つのWAVファイルを複数のCPUで圧縮させるのでなく、1つのCPUに1つのWAVファイルを圧縮させ、その複数の作業を複数のCPUに分けて実行させました。このようにすると、作業間の通信が発生しないので、並列化の効果が非常によく働くと予想できます。

具体的には、NFSマウントされた/home/hogeディレクトリ以下にaudio_[1-9].wavの9つのファイルがあるとき、host[1-3]にrshを使用可能にして、**リスト1**のようなシェルスクリプトを実行すると、並列にWAVからMP3への変換を行うことができます（この場合は3CPUの利用です。http://home.swipnet.se/~w-82625/の BladeEnc 0.80のLinux Pentium/K6-2 etc.（static link）を用いました）。

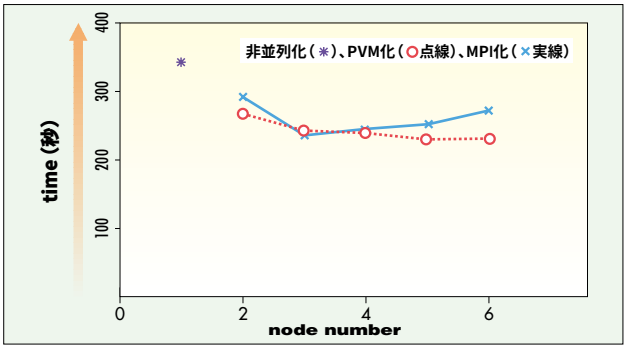
ある音楽用CDから9曲取り出した実験結果を、**表1**に示します。この結果からわかるように、変換時間が台数に

▼リスト1

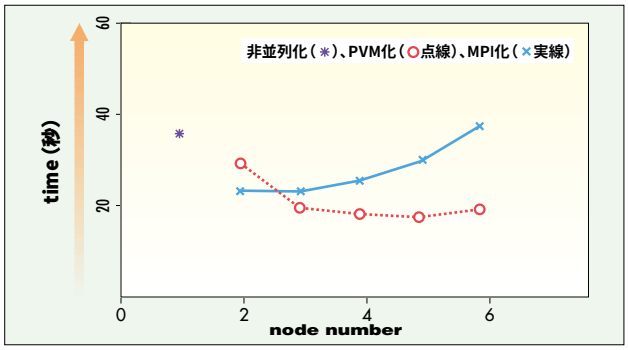
```
#! /bin/sh
time rsh host1 'cd /home/hoge;bladeenc -quiet audio_[1-3].wav' 1> /dev/null 2>> log &
time rsh host2 'cd /home/hoge;bladeenc -quiet audio_[4-6].wav' 1> /dev/null 2>> log &
time rsh host3 'cd /home/hoge;bladeenc -quiet audio_[7-9].wav' 1> /dev/null 2>> log &
```



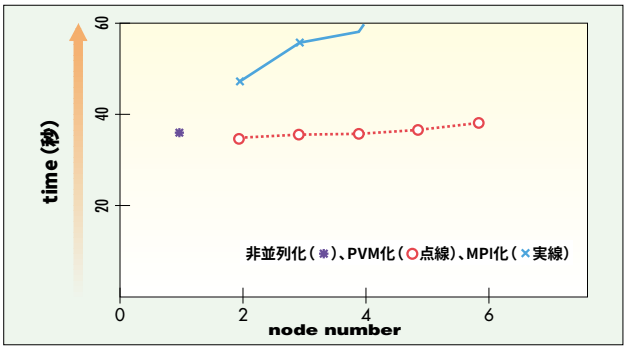
グラフ1　台数を増やし並列度を上げた場合
PVM：CPUを増やしてもあまり並列化の効率は上昇せず
MPI：CPUが増えると並列化の効率が上昇



グラフ2　1台で並列度を上げた場合
MPI、PVMともに並列化により効率アップ。PVMは、プロセスが増えても、並列化の効率が上昇。



グラフ3　台数を増やし並列度を上げた場合（通信量大）
PVM：CPUが増えると、並列化の効率が上昇
MPI：CPUが増えると、並列化の効率が悪化



グラフ4　1台で並列度を上げた場合（通信量大）
PVM：CPUが増えると、並列化の効率が上昇
MPI：CPUが増えると、並列化の効率が非常に悪化

応じて減少しました。ただし、この場合はどの曲も同じぐらいの長さで、かつ音楽の構成も似たものだったのでうまくいったのだと思います。もしも、均一な構成を取らないものについては注意が必要です。

以上、WAVからMP3への変換については並列化の効果が非常によく出ました。ほかにもgzipするファイルが多数ある場合など、応用例は考えればまだまだたくさんあるでしょう。読者の方々もいろいろ挑戦してみてください。

最後に

先ほど紹介した8CPU Linux Diskless Clusterは、現在1CPUあたり384Mバイトの実メモリを載せ、100BASE-TXで接続し、160×160×160のプラズマ並列シミュレーションに使用しています。200～400MFlops程度の実効速度が出ていると考えていますが、期待したよりは遅いです。問題がコードにあるのか、ネットワークなどの要因なのか、原因はまだ特定できていません。しかし、このシステムは昨年夏に1CPUあたり20万円程度で購入し、200万円弱で強力な並列計算機を作り出すことができたのです。この計算速度は1985年ごろのスーパーコンピュータ並の速さですが、値段はその1/100以下程度なので数値計算家にとっては夢のような時代がやってきたといえるでしょう。

しかし、まだまだ問題点もあります。特に、Linux Clusterを構成している計算機が1つでも停止してしまった場合については何も考慮されていません。使用者がそれを踏まえたプログラムを作成するのは非常に困難なため、現在そんな不測の事態が起こったときはいさぎよくあきらめて、もう一度最初から再計算するしかないと考えています。

最後に、数値計算だけに機能を限定したLinuxのインストール用のディストリビューションはほとんど存在しないため、個人的にLinux Diskless Clusterに特化したディストリビューションを作成したいと考えています。機会があれば、この誌面で発表するかもしれませんので、ご期待ください。

CPU数	CPU1の経過時間	CPU2の経過時間	CPU3の経過時間	合計時間
1CPU	46m54s	0	0	46m54s
2CPU	21m19s	25m13s	0	46m32s
3CPU	12m16s	17m28s	18m35s	48m19s

表1　CPU数に伴うMP3変換時間

Column

Linux Seminar

Linux Seminarは、Linux Seminar Working Group（LSWG）によって運営されているLinux勉強会で、東京の六本木にあるGLOCOM（国際大学グローバル・コミュニケーション・センター）を会場に、ほぼ月に1回の割合で開かれています。主に初級者から中級者を対象に、Linux活用に関する話題、面白いアプリケーション、関連する話題などを取り上げて紹介しています（今まで扱ったトピックは次のとおり）。

従来、Linuxに関する集まりは関係者の集う宴会がほとんどでしたが、Linux利用者の裾野が広がるにつれ、よりユーザーを意識した交流の場が必要になっています。そこで、Linux Seminarでは定期的にLinuxに関する各種の活動を行っているメンバーが集まって、活動内容を報告したりするための場を提供していくつもりです。また、勉強会終了後は希望者が集って懇親会を行っており、勉強会とは違う雰囲気であれこれ議論を楽しんでいます。

MLや開催情報などは公式Webページ（http://www.linet.gr.jp/lswg/）を参照してください。　（編集部 北野）

■これまでに扱われたトピック

第1回
A: Plamo Linux （1）　小島 三弘

第2回
A: Plamo Linux （2）　小島 三弘
B: xDSL　小山 裕司

第3回
A: ノートPC　川井 和正
B: GPL　市川 哲郎

第4回
A: Linux向けPC購入ガイド　後藤 敏也
B: PPP概要　真鍋 敬士

第5回
Debian GNU/Linux　吉山 晃

第6回
A: Linux環境の日本語化　芦田 幸治
B: Linuxでのマイコン開発　三岩 幸夫

第7回
A: PostgreSQLとPHP3　桑村 潤
B: SPARCとLinux　鈴木 大輔

第8回
A: Linuxのデスクトップ環境の紹介　川井 和正（LSWG）/野村 明広（LSWG）
B: UNIXコマンド入門　木村 稔

第9回
A: Diskless Cluster Linux　野澤 恵
B: PC-9800でLinuxが動くまで　井伊 亮太

第10回
A: Linux/Alpha入門　後藤 和茂

ディスクレスシステムの構築

ディスクレスシステムの構築の方法論としては、おおまかに分けて次の2つが考えられます。

- **FDで最小限のシステムを作り、ブートまでは自前でやる方法**
- **単にFDをブートストラップのみに使用し、カーネルイメージや/以下のファイルもサーバーから取得する方法**

今回、カーネルイメージの変更を含めたメンテナンスが容易であること、簡潔な美しさ、などの理由から後者の方法を選び、Netbootパッケージを利用しました。このNetbootパッケージは、ドイツのGero Kuhlmannによるネットワーク起動用のパッケージで、GPLで配布されています（**画面1**）。

また、同種のソフトウェアに、Ehterboot、http://www.slug.org.au/etherboot/というものもあります。

なお、前者の方法にも直観的であるという利点があり、特にマシンの構成が個々で大幅に違う場合などは、こちらを採用したほうがトラブルを抑えられるでしょう。

■前説

クライアントマシンのシステムの起動までの流れは、

1. BOOTPの発呼によるIPアドレスの取得
2. tftpによるカーネルイメージの取得、展開、実行
3. NFSによる/以下の取得
4. swap領域の獲得 (任意)
5. NFSによる/usr、/home以下の共有

となります。

▼リスト1　/etc/inetd.conf

```
tftp    dgram   udp wait    root
/usr/sbin/tcpd in.tftpd
bootps  dgram   udp wait    root
/usr/sbin/tcpd /usr/sbin/bootpd
```

▼リスト2　/etc/services

```
bootps          67/udp          # BOOTP server
tftp            69/udp          # TFTP server
```

なお、本稿ではカーネルのビルドの方法、ブートの仕組みについての知識は既知のものとします。理解に不安のある方はJFドキュメント（http://www.linux.or.jp/JF/）のKernel-HOWTO、Diskless、NFS-Rootなどを参照してみてください。

今回、我々はVine Linux 1.0をベースに作業を行いましたが、同じRed Hat系列ならあまり違いはないでしょう。他のディストリビューションの方はディレクトリ構造などを適宜読み替えてください。

■ブートFD作成

最初にNetboot用のFDを作っておきます。まず、READMEの指示（make;make install）に従って、前述のNetbootを導入します。それから、FDをマウントし、Netbootを展開したディレクトリに移動して、

```
# make bootrom
```

とし、質問に答えてFDに書き込むイメージを構築します。その後、

```
# dd if=image.flo of=/dev/fd0
```

で書き込みます。

■サーバーマシン（yadamon）の準備

サーバーマシン側のデーモンの設定を行います。特に関係するデーモンは、bootpdとrpc.nfsdとtftpdです。入っていない場合はインストールしておいてください。

まず、/etc/inetd.confの**リスト1**に示した項目について確認し、コメントになっていたらコメントを外してください。

また、/etc/servicesに**リスト2**に示した記述が含まれていることも確認してください。

続いて、設定ファイルの記述に移ります。tftpには特に必要ありません。bootpdは、**リスト3**に示したように/etc/bootptabファイルに設定します。

ここで、クライアントマシンのMACアドレスを指定する必要がありますが、わからない場合は任意で結構です。NetbootのFDでの起動時にMACアドレスが表示されるので、確認のうえ、再度記述してください。また、NFSの許可を与えるため、/etc/exportsを変更します。

ここまでの作業が終了したら、inetdにHUPシグナルを送って、設定ファイルの再読み込みをさせ、rpc.nfsd、

rpc.mountdは再度起動してください。

■クライアントマシンにNFSするディレクトリの準備

tftpで公開するディレクトリは、習慣的に/tftpboot以下となっています。変更したい場合、/etc/inetd.confのtftpの引数を書き換える必要があります。ここでは例として、/tftpboot/taimon/としましょう。ここをクライアントマシンのルートディレクトリとします。

必要なディレクトリを作成しましょう。まず、マウントポイントとして、

```
# mkdir -p /tftpboot/taimon
# cd /tftpboot/taimon
# mkdir usr home proc
```

空のディレクトリとして、

```
# mkdir tmp
```

を用意します。そして、実体が存在する必要があるディレクトリをコピーします。ただし、シンボリックリンクごとまとめてコピーしなければならないため、単にcpするのでなく、

```
# tar Ccf / -  var etc sbin bin dev lib | tar Cxfp
/tftpboot/taimon -
```

とします。

少しばかり面倒なのが/tftpboot/taimon/etc以下の設定です。これは、サーバーと同一のままでは使えないので、適宜変更する必要があります。ただ、どこをどう変更したらいいのかは環境によって変わってくるので、一概には言えません。最低限、以下の3点について注意してください。

- **マシン固有のもの**
  たとえば、IPアドレス（/etc/sysconfig/のnetwork、network-scripts/ifcfg-eth0）やファイルマウント（/etc/fstab）。
- **余計なデーモンを立ち上げない**
  rc.d/init.d/以下のsendmail、lpd などを止める。
- **/がNFSマウントであることによる障害回避**
  rc.sysinitは大幅な改造が必要です。

ここで注意してほしいのは、/tftpboot/taimon/etcを編集しようとして、誤って/etcを編集してしまうことのないようにしてほしいということです。本来は/var以下のlogも編集する必要がありますが、とりあえず今回は気にしないことにします。

■サーバーマシンのカーネルの準備

起動時の認証にRARPを使うので、カーネルにその機能が含まれていないといけません。つまり、Linuxのソースコードを展開しているディレクトリ（/usr/src/linuxなど）の.configファイルに、

```
CONFIG_INET_RARP=y （もしくは m）
```

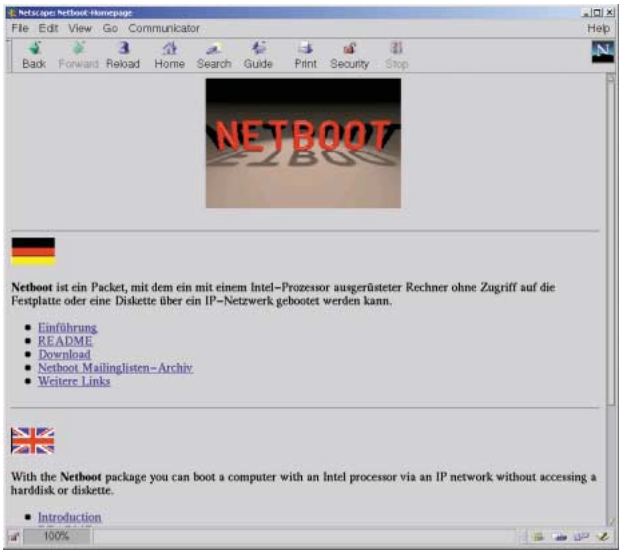
という記述が含まれている必要があります。

■クライアントマシンのカーネルの準備

クライアントのネットワークカード用のドライバサポート、NFSのサポート、そして、“ROOT on NFS”を“y”にしておくことが必須です。あとは運用形態に合わせてください。さらに、実験を簡単にするため、最初はサウンド

▼リスト3　/etc/bootptab

```
.default:¥
                :sm=255.255.255.0:¥
                :ds=client.host.ip.address:¥
                :gw=gateway.host.ip.address:¥
                :ht=ethernet:¥
                :bf=bootImage:
biscuit:hd=/tftpboot/taimon:tc=.default:ha=0000
E320350D:ip=client.host.ip.address
```



画面1　Netboot（http://www.han.de/~gero/netboot）

カードなど余計なドライバは削ってしまうことをお勧めします。これは、あとからも変更が可能です。

また、通常の場合とインストールするディレクトリが異なってくるので、Makefileを**リスト4**のように変更します。

そして、次のとおり実行します。

```
# make zImage
# make modules
# make modules_install
```

ここで、通常のカーネルアップデート時のようにmake zlilo”や“make install”を行うのではなく、次のような手順でインストールします。

```
# cp System.map /tftpboot/taimon/boot
# cd arch/i386/boot
# mknbi-linux -d rom --i rom -k zImage -o bootImage
# cp bootImage /tftpboot/taimon
```

■ブート

さて、お待たせしました。さきほど作成したnfsrootのFDをクライアントマシンに挿入して、ブートしてみてください。うまくいきましたか？ :-)

■問題があった場合

ブート後、途中で固まってしまうときには、クライアントがなんらかのファイルを要求しているのに、それが存在しないためにハングアップしているという場合が多いようです。そのときには、現在のrpc.nfsdを殺して、“rpc.nfsd -F -d call”として起動することにより、リアルタイムにデバッグ情報を表示できます。

bootpdは“-d4”オプションで、デバッグログを“/var/log/message”に残せます。これを有効にするようinetd.confに指定してください。tftpdは、実際に試してみるのが早いでしょう。他のホストから、

```
# tftp serverMachineName
> get  /tftpboot/taimon/bootImage
```

してみてください。

■課題およびTODO

本格的に運用する場合、まず考えなければならないのはswap領域をどうするかということでしょう。もちろん、NFS越しに利用することもできますが、パフォーマンスのみならずネットワークに多大な影響を与えてしまいます。我々は、Windows NTがインストールされているマシンを使用しているので、vfatfsを用いてswapだけはハードディスク上に置くことも検討しています。

我々の環境では単に“shutdown now”として、rc.d/init.d/に従いデーモンを殺していくと正常にshutdownできません。そのため、正常にshutdownするには“shutdown -np now”とする必要があります。これは、システムがhaltする前にnfsdが殺されて、/が読めないことによる障害と思われます。といっても、NFS越しでハードディスクを壊してしまうということは（まず）ありえないので、現在は無視していますが、気になる方は/etc/rc.d以下を調節してみてください。

現状の/etc、/varなどは共有を前提とした仕組みにはなっていないので、複数クライアントにそれぞれ/をもつ必要があります。X の配布に含まれるlndirを使えばいくぶん楽になりますが、これはどうにも美しくないです。ディストリビューション側での対処が望まれます。

なお、パフォーマンスを上げるにはどうしてもNFSがボトルネックになるので、サーバー、クライアントともにメモリはできるだけ多めに積むとよいでしょう。

■終わりに

この記事は、株式会社リサーチのmacさん（http://www.research.co.jp/biscuit/index.shtml）の多大なるご協力によりました。心からの謝意を表します。

（坂田　智之）

▼リスト4　Makefileの変更

```
+ROOT_DEV = /dev/nfsroot
+NFS_ROOT = /tftpboot/taimon/

 modules_install:
        @( \
+
MODLIB=$(NFS_ROOT)/lib/modules/$(VERSION).$(PAT
CHLEVEL).$(SUBLEVEL);
\
        cd modules; \
        MODULES=""; \
```