StarBEDにおけるインタラクティブなノード制御

知念 賢一[†§]、**三角 真**[#]、**宮地 利幸**[§†]、**篠田 陽一**[†§]
北陸先端科学技術大学院大学 インターネット研究センター[†]、情報通信研究機構 北陸リサーチセンター[§]
東京工業大学 理工学研究科[#]

概　要

情報通信研究機構 北陸リサーチセンターのネットワーク実験設備 StarBED では、数百台の PC 上で様々な実験が並行して実施されている。その内容に応じて、様々な OS の切替えや電源管理が必要である。また、設備は数百台規模であるため、極力自動化することが望ましい。我々はネットワーク実験をスクリプトで自動化するソフトウェア群を開発してきたが、それを補完するインタラクティブな枠組も管理に有効であるという知見を得た。本論文では、インタラクティブなノード管理を行うユーザ・インターフェイスとそれを支える要素技術を紹介する。

An Interactive Node Control on StarBED

Ken-ichi Chinen[†§]*, Makoto Misumi*[#]*, Toshiyuki Miyachi*[§†]*and Yoichi Shinoda*[†§]
INTERNET RESEARCH CENTER, JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY[†]
HOKURIKU RESEARCH CENTER, NATIONAL INSTITUTE OF INFOMATION AND COMMUNICATION TECHNOLOGY[§]
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING, TOKYO INSTITUTE OF TECHNOLOGY[#]

Abstract

In large scale network testbed like StarBED, people which do experiments require mechanisms to change OS and to control power of these nodes. We proposed and developed batch control framework as script language processing system. However, we found that interactive node control has remarkable potential though the operation of StarBED. We have addressed ourself to the interactiveness of them since then. The interactive control is adaptive and it helps batch control, also. This paper describes the interactive node control and fundamental technologies to support that.

1 は じ め に

ネットワーク実験では様々な装置を制御する必要がある。特に PC を用いる場合には、PC の持つ汎用性に対応して、ソフトウェアの導入や設定変更等、様々な制御が必要である。円滑に実験を進行させるためには自動化が欠かせない。大規模に運用する場合には、電源投入や電源切断等の手間も無視できない。スイッチを一つ押すような単純作業も数百台では長い時間を必要とする。

これまで我々は、StarBED のようなテストベッドにおけるネットワーク実験支援ソフトウェアの集合体である SpringOS を開発してきた [1]。その中心は Kuroyuri というスクリプト言語処理系で、実験で使用する資源やその手順を記述したスクリプトによって実験を駆動する [2]。これはいわばバッチ的なノード制御である。この記述言語はノードやネットワークを抽象的かつマクロ的に扱うため、ミクロな機能は内蔵しているものの、構文として提供することは難しい。たとえば、ノードの電源投入や OS のディスクイメージ導入は、ノード設定という大きな処理の内部に含まれており、それぞれを取り出して実行することはできない。

実際の実験現場では、スクリプトを記述する前の予備実験のためノードの電源投入を行いたい、というような場面が出てくる。我々はユーザの入力を逐次実行

する機構でこのような要望を解消する。すなわちインタラクティブなユーザ・インタフェースを設ける。実験準備や予備実験、後始末時に用いるため、スクリプトがない時点でノード制御せねばならない。そのため、物理的な名前でノードを扱うよう実装した。

2 背 景 と 目 的

本章では、本研究の背景であるネットワーク実験設備 StarBED を紹介し、目的を述べる。

2.1 StarBED

StarBED は独立行政法人 情報通信研究機構 北陸リサーチセンターに設置されているネットワーク実験設備、およびその設備を用いて研究を行う組織（プロジェクト）の名称である [3, 4]。設備は 680 台（2008 年 10 月現在）の PC とそれを接続するスイッチ群で構成される。各種アプリケーションプロトコルや装置の動作試験および耐久試験、複雑なトポロジを構築したルーティングの検証等の用途に用いられている。

実験を希望する者は利用資源や期間を StarBED へ申請する。申請に応じて StarBED の管理者が申請者（以下、**ユーザ**）へ資源を割り当てる。各実験に合わせて PC (OS 等) やスイッチ類（VLAN 等) へユーザ自身が設定を施し、ネットワークを構築する。管理者は装置故障時のメンテナンスやユーザ間の各種調停を行う。

ユーザは実験内容に応じて、多種多様な OS を各 PC（以下、**ノード**と呼ぶ）で起動する。たとえば、ディスク中のインストール済の OS を起動したり、新たな OS をディスクへインストール後、起動する。ディスク操作のオーバヘッドや後始末を省くため、ディスクレスとしてネットワークから起動する場合もある。多数のノードを使うことから、ノード間の情報共有やディスク占有量を軽減するため、NFS などの分散ファイルシステムを併用することもある。

このような OS の入れ換えが頻繁に起きるのが StarBED の特徴である。極端な場合は百台単位の OS 入れ替えが週に数回発生することもある。また、この OS 入れ替え作業に伴って、リブートや電源投入・切断が頻繁に発生する。

2.2 SpringOS

SpringOS は我々が StarBED を念頭に設計・開発したネットワーク実験支援ソフトウェア・スイーツである。SpringOS は多くのプログラムで構成され、ネットワーク実験の用途や形態によって、その中からプログラムをいくつか取り出して組み合わせる。すなわち、

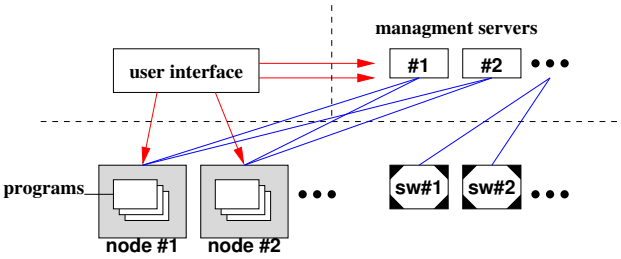

図 1: SpringOS プログラムの種別

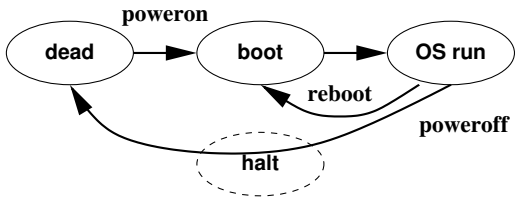

図 2: ノード活動サイクル

SpringOS は様々なソフトウェア集合の総和の名称である。ソフトウェアはノードで稼働するもの、設備全体で稼働する管理サーバ群、そしてユーザ・インターフェイスに大別される（図 1）。

前述のように、ユーザ・インターフェイスはバッチ的なノード制御としてスクリプト言語処理系 kuroyuri が実装済である。これは、実験全体を見たトップ-ダウン的な視点で動作するため、細かなノード制御には向いていない。これは一括実行という性質上の特徴であり、スクリプト言語にノード制御向けの細かな構文を設けるより、ノード制御向けのインターフェイスを別に設けるのが自然であろう。

2.3 目 的 と 要 件

ノードの起動や停止の流れ、いわば活動サイクルを図 2 に示した。ノードは電源が入っていない状態（dead）からブート処理を経て OS 実行となる。そして、OS 実行中から電源が入っていない状態へ戻る。一部のハードウェア構成では、OS が停止して電源が入っている状態（halt）を経ることがある。本研究の目的は、このサイクルを形作る要素技術群をインタラクティブに操作することである。

先に述べた StarBED とその上で実施される実験の現状から、要件（requirement）を挙げる。

1) 電源投入・切断の自動化
2) OS 切替の自動化
3) 様々な OS への対応、ディスクレスへの対応
4) ノードを制御するインタラクティブなユーザ・イ

表 1: SpringOS と類似ソフトウェア群との比較

機能		実験支援スイーツ			参考システム	
		SpringOS	emulab	PlanetLab	バックアップ	シン・クライアント
電源管理	電源投入	◎	×	×	×	×
	電源切断	◎	×	×	○	○
OS 切替え	起動切替	◎	◎	×	×	◎
	対象 OS	◎非特定	○限定	×固定	◎	○限定
	ディスクレス	◎	×	×	×	×
	スワップ *	◎	◎	×	◎	◎
ネットワーク実験	バッチ	◎	◎	×	—	—
	インタラクティブ	◎	×	◎	—	—

* 要件 (必須) ではないが、望ましい機能。

ンターフェイス

また、実験の再開を容易にするため、もう一つの望ましい機能 (recommend) を挙げる。

5) ディスク内容の退避 (スワップ)

このスワップの有用性は後述する。

2.4 本研究の位置付け

先に挙げた要件を中心に、本研究と主要なネットワーク実験支援ソフトウェア・スイーツや参考システムとの機能比較を表 1 にまとめる。実験支援スイーツは emulab [5, 6] と PlanetLab [7] を挙げた。我々の研究は OS を限定せず、ディスクレス OS まで対象としていることが特徴である。

実験支援スイーツ以外ではバックアップ・システムとシン・クライアント（Thin-client）を比較対象とした。バックアップ・システムはディスク操作に特化し、シン・クライアントは複数の OS 起動を扱えるなど、いくつかの点が我々の要件に合うが、全てを網羅しているわけではない。

3 前提とする要素技術

3.1 ブートローダ

ブートローダとは OS を起動するプログラムの総称で、一般にディスクのブートセクタに格納されている。ブートセクタはディスク先頭（特に MBR と呼ばれる）、あるいは各パーティションの先頭に配置されている（図 3）。

ブートセクタは 512 バイトで、そのうちプログラム領域は 446 バイトである。このサイズで記述できるプログラムは限られているため、多くの OS では段階的に起動する（図 4）。慣習として各段階は**ステージ**と呼ばれる。

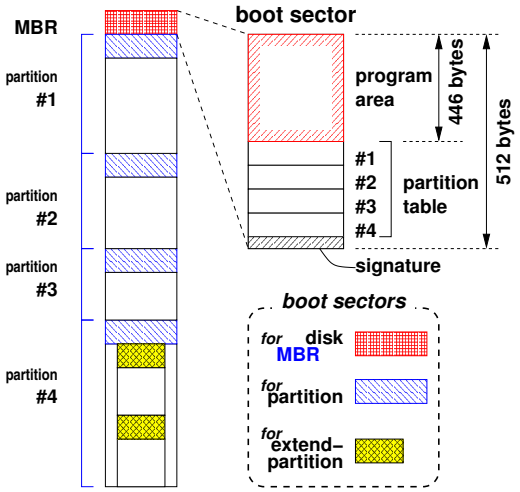

図 3: ディスクレイアウト

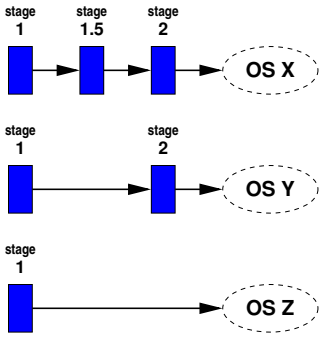

図 4: 様々なブートローダ

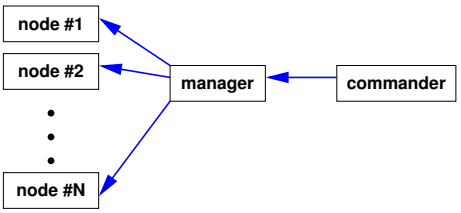

図 5: マネージャ・モデル

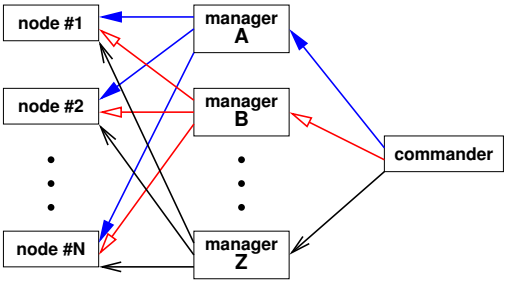

図 6: 複数マネージャ

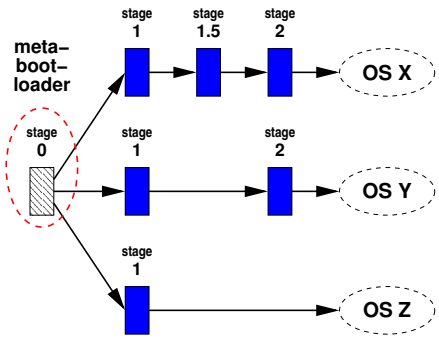

図 7: メタ・ブートローダ

3.2 PXE

近年の PC ではネットワークからの起動手段として PXE [8] が普及している。PXE はブートローダをネットワークから取得する機構で、DHCP [9] や TFTP [10] を用いる。取得したブートローダを実行することで、所定の OS を起動する。ディスクを使わずに起動できることが大きな利点である。

4 設 計 と 実 装

多数のノードを管理することを踏まえて、統括して管理するマネージャを設置する（図 5）。図中 commander はユーザが操作し、ノードへの指令を発行するプログラムである。先に述べたように、本研究では複数の要素技術をまとめる。したがって、結果的に複数のマネージャを扱うことになる（図 6）。

```
0:7C00h から 0:0600h へコピー
        // バッファからプログラム領域へコピー
0:0600h へジャンプ
        // プログラム領域へジャンプ
LBA アクセスが可能か確認
if(LBA アクセス不能) {
  エラーメッセージを出して停止
}
while(指定パーティションまで繰り返す) {
  ブートセクタ開始ブロックの位置を取り出す
  ブートセクタをバッファに読み込む
}
0:7C00h へジャンプ
        // バッファ先頭へのジャンプ
        // 次のブートローダへ処理を委譲
```

図 8: メタ・ブートローダのアルゴリズム

4.1 メタ・ブートローダ

StarBED では全てのノードはネットワーク起動（PXE）としてある。ノードのハードウェアは起動後にブートローダをネットワークで取得し、そのブートローダから OS を起動する。ディスクから OS を起動させるため、ディスクとパーティションを指定してディスク上のブートローダを読み込むブートローダを与える。すなわち、ディスクのブートローダを読み込むブートローダをネットワークで配布する。ここでは、このような機構を**メタ・ブートローダ**と呼ぶ（図 7）。

このようなメタ・ブートローダを設けることで、ディスク数とパーティションレイアウトの組み合わせに応じて、複雑な起動が可能となる。PC の元々の仕様では、ディスクあたりのアクティブ（起動）パーティションは一つに制限されているが、このアプローチではその制限から解放され、多数のパーティションから任意のパーティションを選んで起動可能となる。

このようなブートローダの連鎖は MBR に導入する多くのブートローダに見られる。すなわち、MBR から起動後、各パーティションのブートローダを呼び出す際に連鎖が行われる。ただし、各パーティションのブートローダ呼び出しをネットワークから提供する点がこのアプローチの特徴である。

4.1.1 coil

我々のメタ・ブートローダの実装である coil のアルゴリズムを疑似コードで図 8 に示す。ブートローダの実行はバッファ上で始まる。後続のブートローダを読み込むため、バッファを空ける必要がある。最初に別に設けたプログラム領域へ自身をコピーする（図 9）。その後、各パーティションのブートセクタ場所とその

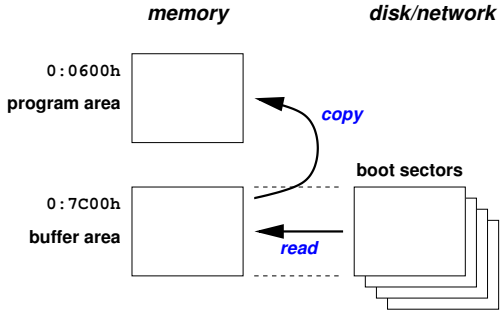

図 9: メモリ配置

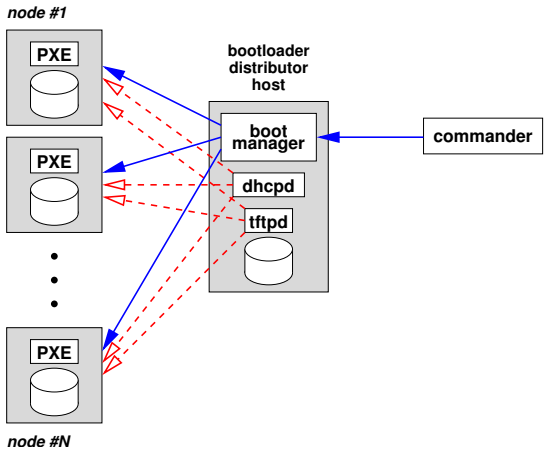

図 10: ブートマネージャ

内容を指定場所まで読み込み、そのブートローダに処理を委譲する。

このような機構により、一般的な OS の多くを起動可能とした。以下に起動を確認した OS をあげる。これ以外にも多くの OS が起動可能だと思われる。新規インストールについては後述する（4.3.1 節）。

- Windows Server 2000
- Windows XP
- FreeBSD 4.x
- TurboLinux
- FedoraCore 5

4.2 ブートマネージャ

OS 起動を統合的に管理するため、専用のプログラム、ブートマネージャを設ける（図 10）。DHCP と TFTP を制御することと相互のデータコピーの削減のため、dhcpd と tftpd を同じホストで稼働させる。このホストを**ブートローダ配布ホスト**と呼ぶ。このブート関連ファイルを格納しているディレクトリにはファイルが多く、ファイル操作の負荷や管理の手間を軽減するために、ファイル数を極力抑える必要がある。

表 2: pxelinux 向けディレクトリ構成

ブートローダ	/tftpboot/**ホスト名**.pxe 実体は pxelinux.0
設定	/tftpboot/pxelinux.cfg/→ **16 進 IP アドレス**
ユーザ設定	/tftpboot/**ユーザ名**/pxelinux.cfg/→ **ホスト名**

```
DEFAULT  linux
IPAPPEND 1
label linux
    kernel chinen/vmlinuz-2.6.13-starbed-ide
    append initrd=chinen/piperf_ide080324
```

図 12: pxelinux 設定例

4.2.1 シンボリック・リンクによるブート切替

前述のような各種ブートローダを切替えるため、ブートローダ配布ホスト上に多数のブートローダを蓄え、必要に応じて配布するブートローダを切替える。その際、シンボリック・リンクで配布するブートローダを関連づける。たとえば、ノード sintcla001 を第一パーティションから起動する際には、そのブートローダ sintcla001.pxe を第一パーティション向けブートローダ p1.bin にシンボリック・リンクで関連づける。第二パーティションから起動する際には、同様に p2.bin にシンボリック・リンクで関連づける。

4.2.2 pxelinux ディレクトリ構成

ここでは、メタ・ブートローダを用いず直接ネットワークから起動する場合を紹介する。

PXE 起動の OS の一つに pxelinux [11] を用いた Linux がある。pxelinux を使う場合は、ブートローダは共通だが各種設定やその設定で用いるカーネルや initrd 等のファイル群が必要となる。パーティションの分だけ用意すればよいメタ・ブートローダと異なり、そのバリエーションは多岐に渡るため、管理が繁雑になる。そこで、システムの設定ファイルとユーザのファイルを格納するディレクトリを設け（表 2）、必要に応じてシンボリック・リンクで割り当てる。

たとえば、ノード sintclb019 へユーザ chinen の設定 sintclb019.cfg を割り当てる場合を説明する。pxelinux が要求するパスは、sintclb019 の IP アドレス 172.16.1.19 に基づく pxelinux.cfg/AC100113 である。したがって、tftpd がアクセスするパスは/tftp-

1	与えたい設定	/tftpboot/chinen/pxelinux.cfg/sintclb019.cfg
2	pxelinux の要求	pxelinux.cfg/AC100113
3	tftpd のアクセス	/tftpboot/pxelinux.cfg/AC100113

3 → 1 へのシンボリック・リンクを設ける。
ただし、TFTP クライアントから / は見えないので絶対パスは使えない。
そこで、相対パスで ../chinen/pxelinux.cfg/sintclb019.cfg とする。
※ AC100113 は sintclb019 の IP アドレス 172.16.1.19 から生成された名前

図 11: pxelinux.cfg リンク例

表 3: FreeBSD 向けディレクトリ構成

ブートローダ	/tftpboot/**ホスト名**.pxe 実体は pxeboot
設定	/tftpboot/boot/loader.rc
カーネル	/tftpboot/kernel_**IP アドレス**
ディスクイメージ	/tftpboot/diskimage_**IP アドレス**

表 4: DMP コマンド群

コマンド	摘要
STAT	ファイルの属性表示
SYMLINK	シンボリックリンク作成
RMSYMLINK	シンボリックリンク削除

boot/pxelinux.cfg/AC100113 となる。これを../chinen/pxelinux.cfg/sintclb019.cfg へリンクする。図 11 にパス一覧を示す。図 12 に設定例を示す。

4.2.3 FreeBSD ディレクトリ構成

StarBED では FreeBSD の PXE 起動は pxeboot 向けに設定しており、表 3 のようなディレクトリ構成になっている。このカーネルやディスクイメージのパスパターンは設定ファイル loader.rc に記述している。

4.2.4 ディレクトリ操作サーバ DMAN

前述のような各種ブートローダ切替を数百台規模で行う場合、手作業は現実的ではなく自動化が必要である。また、ブートローダ配布ホスト上で多数のユーザが無秩序に作業すると混乱を招く。これらの作業は競合を防ぎ排他的に処理する形態が望ましい。そこで、我々はブートマネージャの実装例として、ディレクトリ操作サーバ DMAN を開発した。

DMAN はネットワークを介したクライアントのリクエストに応じて、前述のブートローダに関わるシンボリック・リンクを管理（作成・削除）する。サーバとして実現したため、設備管理者とユーザの責任分界が明らかになり、ディレクトリ操作のインターフェイスを統一する効果を持つ。

ブートローダを操作する性質上、DMAN はブートローダ配布ホスト上で稼働せねばならない（図 13）。DMAN は独自の DMP（Directory Manipulation Protocol）を用いる。DMP はシンボリックリンクを操作する簡単な行指向のプロトコルである。表 4 にそのコマンドをまとめた。

4.3 ディスクイメージ操作

OS 切り替えの一環として、ディスクイメージの操作を行う。これは起動ディスクのディスクイメージを入れ替えれば、OS の切り替えとなるからである。図 14 で示すように、実験時にとりうるディスクイメージに関する処理の流れは、a) インストール済 OS を使用、b) 新規 OS を使用、c) 再開、の 3 通りである。以下、新規インストールと入れ替えのためのスワップ処理を説明する。

4.3.1 新規インストール

前述のようにメタ・ブートローダはディスク上のブートセクタ内のブートローダを起動する。多くの OS のインストール・プログラムは、ディスク上のブートセクタ内にブートローダを書き込むため、この手法で対応できる。すなわち、多くの OS では特別の操作が不要で、単純に当該 OS の手順に従ってインストールするだけで良い。この手法で対応できないのは、複数のディスクやパーティションを用いる場合である。これは特殊な場合で一般化は困難なことから例外的に対処することになる。

後述のように一括インストールを利用するなら、CD 等を用いる繁雑なインストール作業は最初の一台だけでよい（4.3.3 節）。

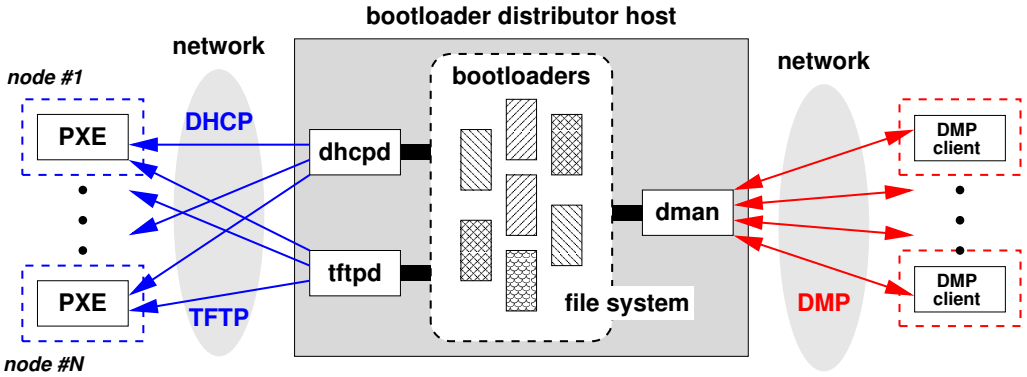

図 13: ブートローダ配布ホスト上の DMAN 配置

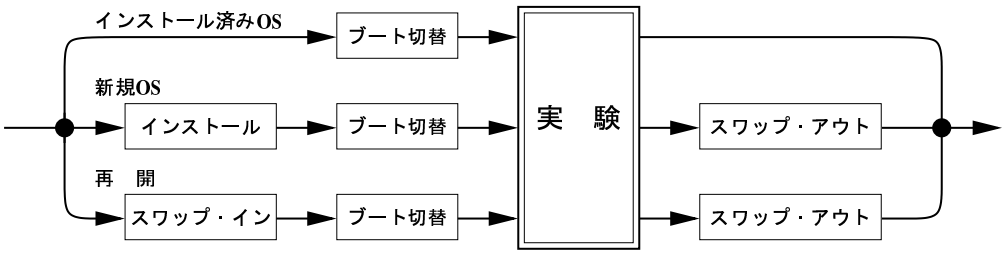

図 14: ディスクイメージ操作の流れ

4.3.2　スワップ処理

一般的に、実験では与えるパラメータを変更して何度も手順を繰り返す場面が多い。また、一旦行った実験を後日もう一度実施することもある。そこで、実験で用いたノードのディスイメージを保存し、後日復元できれば実験再開が容易になる。ここで、保存を**スワップ・アウト**（swap-out）、復元を**スワップ・イン**（swap-in）と呼ぶ。

スワップ・アウトはディスクおよびパーティション内容を読み込み、ネットワーク上のサーバに転送する。逆に、スワップ・インはサーバに保存されているディスク内容をディスクおよびパーティションに書き込む。これは PXE 起動のディスク操作専用に最適化したディスクレス OS とディスク操作プログラムによって実現している。既に文献 [12] として発表済なので、ここでは説明を省く。ノード活動サイクルにスワップ処理を含めて描くと、図 15 となる。

4.3.3　一括インストール

スワップ・イン、スワップ・アウトと対比して記述すると、ノード毎の処理を想像するが、あるノードのスワップ・イン結果を複数のノードへスワップ・アウトするとノードの複製となり、多数ノードへの一括インストールとなる。この手法を用いると、ユーザの手作業によるインストールは最初の一台だけで済む。たとえば、FreeBSD の 200 台一括インストールが 22806 秒（平均 114 秒/台; 1 分 54 秒/台）等の実績がある [13]。

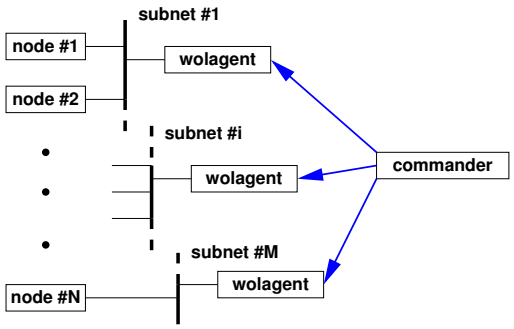

図 16: WoL 関連配置

同様に各種バックアップシステムが一括インストールの機構として流用可能である。これが、前述のようにバックアップシステムを本研究の比較対象として扱った理由である。

4.4　電源投入・切断

StarBED では Wake on LAN(WoL) [14] によって、ネットワークを介してノードの電源を投入する。WoL はデータリンク層の技術のため、指令する PC と指令を受けて起動する PC が同じサブネットワークに接続している必要がある。数百台規模の実験設備では、その全てを一つのサブネットワークに接続することは難しい。同様に、ユーザとノードが直接同じサブネットワー

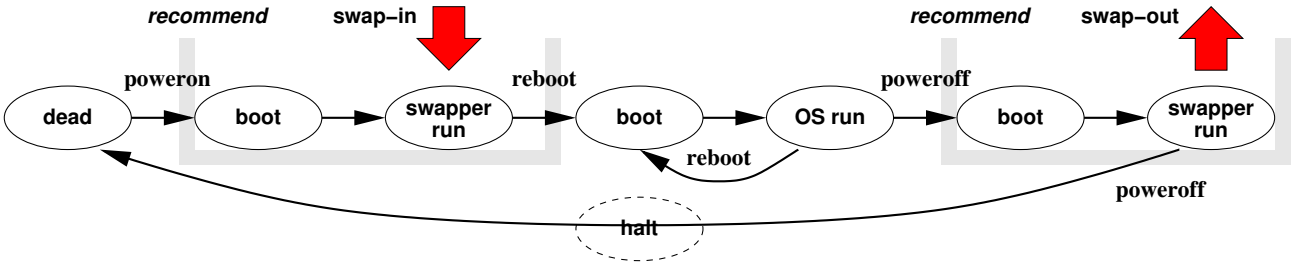

図 15: スワップを含むノード活動サイクル

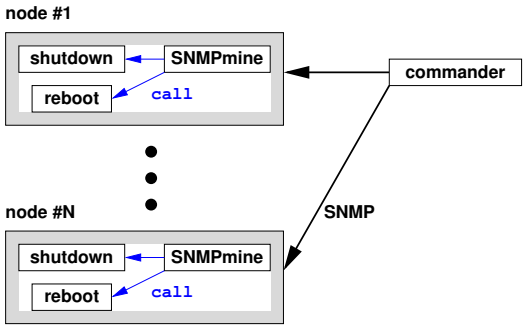

図 17: SNMP 関連配置

クに接続することも難しい。そこで、WoL を中継するプログラム wolagent を作成した。図 16 にその配置を示す。この wolagent もマネージャの一つである。

StarBED の大部分のノードは電源を切断するための管理ハードウェアを持たない。そこで、ソフトウェアで電源切断を行う。各ノードで SNMP 受信プログラムを設置し、特定メッセージを受信した際に shutdown プログラムを呼び出す。StarBED のノードはソフトウェアから電源を切断できるため、図 2 に登場した halt 状態に陥らず、OS 実行中から電源切断に至る。同様に reboot プログラムの呼び出しによって、再起動も実現した。図 17 にその配置を示す。現状では、これらの SNMP メッセージは NEC MIB を利用している。実装は NEC 製の snmpd プラグインや、別途開発した SNMPmine を用いる。

4.5 ユーザ・インターフェイス

アイコンをマウスでクリックするような単純な GUI は、マウス操作が作業のボトルネックとなってしまう。つまり、作業時間がマウス操作に比例してしまう。数百台制御する際には数百回、数千回マウス操作が必要となり現実的ではない。そこで、今回は CUI を採用した。また、CUI であれば容易にプログラマブルになる。

4.5.1 シェル sbpsh

本章で紹介したマネージャ群と通信するプログラム sbpsh を作成した。sbpsh を用いることで、ユーザはノード制御として、起動、OS 切替、電源投入・切断、リブートを指示できる。たとえば、ノード 100 台を第二パーティションで起動する際には次の 2 行を入力する。

```
> setdiskboot sintclb1-100 2
> poweron sintclb1-100
```

ハイフンは連続したノード群を表現する。このコマンド列は、DMAN へ 100 台分のブートローダ切替え、wolagent へ 100 台分の起動を発行する。

また、コンマによって隔たりがあるノード群を指定できる。以下の例は、sintclb8 を除いた電源オフの指定で、SNMP による shutdown を 99 台分発行している。

```
> poweroff sintclb1-7,sintclb9-100
```

4.6 障 害 対 策

今回のシェル sbpsh ではユーザの命令を逐一実行するため、特に障害回復はせず、ユーザに任せている。インタラクティブなノード制御はほぼプリミティブに近いため、障害対策をする場合はより上位に設けるべきである。前述のように sbpsh のプログラマブルな側面を活用した場合（sbpsh を呼び出して命令シーケンスを構成する等）、障害についての注意が必要である。

ちなみに、スクリプト言語処理系 kuroyuri では、ノード確保時に予備ノードを用意して、応答のないノードは実験に採用しない等、いくつかの対策を施している。ネットワークの障害は Spanning Tree Protocol （STP）[15] 等で回避することを想定している。

4.7 お さ ら い

ノード電源投入は WoL を用い、その管理を wolagent で行う。OS 起動は PXE とし、そのブートローダは DHCP や TFTP で配布、その管理を DMAN で行う。ディスク上の OS 起動はメタ・ブートローダ coil を用いる。また、Linux や FreeBSD のディスクレス起動も可能である。ノード停止や再起動は SNMP のメッセージを送り、SNMPmine でノード上の shutdown や reboot を実行する。このそれぞれの技術を呼び出すユーザ・イ

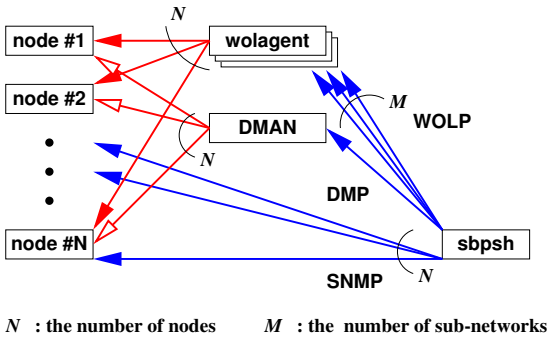

図 18: システム構成

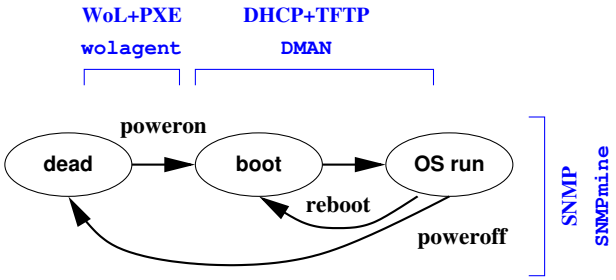

図 19: ノード活動サイクル

ンターフェイスとしてシェル sbpsh を作成した。図 18 に全体のシステム構成を、図 19 に技術や実装を加えたノード活動サイクルを示す。

5 今後の予定

従来、管理ハードウェアは各社が独自規格で提供していたが、近年では IPMI [16] に統合されつつあり、普及が進んでいる。そこで、本研究でも IPMI への対応を計画している。ハードウェアを導入することにより、ソフトウェア暴走時にもノードの管理が可能となる。

前述のように数百台規模に GUI を導入するのは容易ではないが、直感に訴える操作性を考えると GUI も捨てがたい。今後、多数のノードを容易に扱える GUI の導入を検討する。

6 おわりに

多数の PC を扱う設備では、管理維持のために作業の自動化が重要である。これまで自動化に向けて多くの技術が開発されてきた。加えて StarBED のようなネットワーク設備では、頻繁に OS を切替えるため、起動 OS 切替の自動化も必要である。本研究では、StarBED の従来のスクリプトによるバッチ的なノード制御以外に、インタラクティブなシェルを設けてノードを制御する。特にノードの活動サイクルに注目して設計した。ユーザはユーザ・インターフェイスから制御を指示でき、ノードの電源投入・切断、そして OS 切替をシームレスに扱える。この結果、設備維持や実験準備の作業が容易になり、実験を迅速に進行させる。

本研究で扱った技術のうち、起動 OS の切替えは特に重要である。多くのネットワーク実験設備や支援ソフトウェア・スイーツが、固定あるいは限られた数個の OS しか対応していない状況で、この技術を用いることで StarBED は対応 OS が増えた。また、ディスクレス対応によりディスクのオーバヘッドを大幅に軽減することも可能にした。

謝辞

本論文で紹介したノード管理の枠組には、我々の実験に関する要件だけではなく、ユーザの要請も反映されている。本研究の推進にはユーザが不可欠であった。ユーザの方々に深く感謝する。

設備設置当初は日本電気株式会社に技術的支援を受けた。

参考文献

[1] Toshiyuki Miyachi, Kenichi Chinen, and Yoichi Shinoda. StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software. In *International Conference on Performance Evaluation Methodlogies and Tools (Valuetools) 2006*, October 2006.

[2] Ken-ichi Chinen and Toshiyuki Miyachi and Yoichi Shinoda. A Rendezvous in Network Experiment — Case Study of Kuroyuri. In *International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities(Tridentcom)*, March 2006.

[3] The StarBED Project. http://www.starbed.org/.

[4] 独立行政法人 情報通信研究機構北陸リサーチセンター. http://starbed.nict.go.jp/.

[5] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. pp. 255–270, Boston, MA, December 2002. USENIXASSOC.

[6] Eric Eide, Leigh Stoller, and Jay Lepreau. An experimentation workbench for replayable networking research. In *Proceedings of the Fourth Symposium on Networked Systems Design and Implementation (NSDI 2007)*, pp. 215–228, April 2007.

[7] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *HotNets-I '02*, October 2002.

[8] Intel Corporation. Preboot Execution Environment（PXE） Specifiction Version 2.1, sep 1990.

[9] R. Droms. Dynamic Host Configuration Protocol, RFC2131. March 1997.

[10] K. Sollins. The TFTP Protocol (Revision 2), RFC1350. July 1992.

[11] H.Peter Anvin. PXELINUX - SYSLINUX for network boot. http://syslinux.zytor.com/pxe.php.

[12] 三角真, 宮地利幸, 知念賢一, 篠田陽一. 実ノードを利用したネットワークシミュレーションにおけるノードへの OS の導入及びパラメータ設定機構の開発. 情報処理学会研究報告書 DPS-116, pp. 95–100, January 2004.

[13] 宮地利幸. 大規模実証環境の実現と実験支援によるネットワークサービスの検証技術. 北陸先端科学技術大学院大学博士論文, 2007.

[14] Advanced Micro Devices, Inc. *Magic Packet Technology*, November 1995.

[15] IEEE Computer Society. 802.1D: IEEE Standard for Local and Metropolitan Area Networks Media Access Control (MAC) Bridges. 2004.

[16] Intel Corporation. *IPMI v2.0 specifications Document Revision 1.0*, February 2004.

[17] StarBED Project. できる StarBED — SpringOS で簡単大規模実験 — version 0.1, 2008. http://www.starbed.org/documents/sb_usr_intro-0.1.pdf.

[18] StarBED Project. できる StarBED — 仮想機械で簡単 SpringOS 体験 — version 0.2, 2008. http://www.starbed.org/documents/vmstarbed-0.2.pdf.

A その他の情報

SpringOS は http://www.starbed.org/ で公開している。SpringOS や StarBED の使い方は文献 [17, 18] が詳しい。