

# SpringOS/VM: 大規模ネットワークテストベッドにおける仮想機械運用技術

宮地 利幸<sup>†</sup> 知念 賢一<sup>†</sup> 篠田 陽一<sup>§</sup>

北陸先端科学技術大学院大学 情報科学研究科<sup>†</sup> / 情報科学センター<sup>§</sup>

## 概要

近年のインターネットへ導入される技術の対象範囲は大規模化しており、それにともない大規模なネットワーク検証環境が必要となってきた。実ノードを用い、実環境と同様の実装を用いた実験トポロジを構築できる大規模なテストベッドが提案されているが、要求される実験トポロジは日々大規模化しており対応は難しい。そこで、仮想ノード技術により実ノードを多重化し検証環境の規模を拡大する。

実ノードによる実験環境では実環境との同一性が問題になるため、汎用 OS を動作させることのできる仮想機械を用い実ノードを多重化を行う。本論文では、我々は開発している実験支援システムである SpringOS に仮想機械を制御できるよう拡張を行った SpringOS/VM について述べる。

## SpringOS/VM: A Technique for Managing Virtual Nodes on a Large-scale Network Testbed

*Toshiyuki Miyachi<sup>†</sup> Ken-ichi Chinen<sup>†</sup> Yoichi Shinoda<sup>§</sup>*

SCHOOL OF INFORMATION SCIENCE<sup>†</sup> / CENTER FOR INFORMATION SCIENCE<sup>§</sup>,  
JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY

## Abstract

The target network scale of the latest technologies for the Internet has become larger, and a large-scale network testbed is needed to evaluate these technologies. Large-scale testbeds with actual nodes to use same implementations on the Internet are proposed, however, we can not build the network topologies which request the nodes more than prepared nodes in these testbeds. So we use virtual nodes running on a physical node to build larger network topologies.

In an actual node based testbed, basically users want to build more realistic topologies, so we use virtual machines on which generic OSs run. In this paper, in order to control virtual machines, we describe extension of SpringOS, a experiment support system for large-scale actual node based testbed.

## 1 はじめに

近年、大規模なネットワークを対象とする技術が開発されるようになり、ネットワーク技術の検証に必要とされる実験トポロジも大規模化している。このような検証用ネットワークを構築する手法に、ソ

フトウェアシミュレータと実ノードによる実験環境がある。ソフトウェアシミュレータを用いて検証を行う場合は、ノード数が多ければ実験に必要な時間が長くなり現実的でない。実ノードを用いた場合は、実験シナリオに記述された時間で実験は終了するが、

大規模な実験トポロジの制御は非常に困難である。しかし大規模なネットワーク実験の結果を、現実的な時間内で得るためには、実ノードを用いた実験環境を利用する必要がある。

実ノードを用いた大規模なテストベッドとして、StarBED[1] や Netbed[2] が提案されている。このような環境では基本的に多数の計算機を集め、1 計算機を実験トポロジ上の 1 ノードとして動作させることで大規模な実験トポロジを構築する。存在する計算機では実現できない規模の実験トポロジを構築するために、その都度、新たにノードを用意することは、経済的費用、設置するための空間、そして計算機の保守費用などさまざまなコストが必要となるため現実的ではない。

この解決策として、1 台の計算機を仮想的に多重化することで、さらに大規模な実験トポロジを構築する手法がある。すでに用意されている計算機を仮想的に多重化すれば、新たな計算機を用意するためのコストは発生せず、計算機の保守費用も増加することがないため、新たな計算機を用意する場合よりも、さまざまな点でコストが小さい。

1 台の実ノードを多重化するための手法は数多く発表されており、一般的に、仮想化の度合いが高ければ 1 台の計算機上で動作できる仮想ノードの数は増加する。しかしその一方、仮想化の度合いが高いほど、実環境との同一性が損なわれることが多い。

実ノードを用いたテストベッドでの実験は、実環境用の実装を大規模な実験トポロジ上で検証することが目的である場合が多い。したがって、実験遂行者が要求する OS やアプリケーションが変更なしに動作することは非常に重要である。汎用的な OS が動作しない場合は、実験遂行者が意図するアプリケーションの動作に支障をきたす場合や、実験遂行者が変更を加えた OS を利用できないなど、問題が生じる場合がある。

そこで我々は、汎用的な OS が変更なしに動作する仮想機械を利用し、テストベッドの更なる大規模化を目指す。我々は、大規模テストベッドでの実験支援システム SpringOS の開発を行っており、本論文では SpringOS を仮想機械を取り扱えるよう拡張した SpringOS/VM について述べる。

SpringOS/VM では、ある程度高速に動作するため

多重化しやすい VMWare[3] を利用することとした。Netbed では jail[4] を利用した多重化を実現しているが、jail は FreeBSD のプロセス空間を分割するものであり汎用的でない。

## 2 実ノードによるテストベッドへの仮想機械の導入

本章では、実ノードによる大規模なネットワークテストベッドへ仮想機械を導入する際の問題点などについて述べる。

### 2.1 仮想機械の利用

前述の通り実ノードを利用したテストベッドでは、実環境との同一性は重要である。しかし仮想機械を利用した場合、ハードウェア部分をソフトウェアで模倣しているため同一性は低下する。

実験トポロジを構成するそれぞれの要素に必要な同一性の度合いは異なることが多い。一般的に主要な部分の同一性は高い必要があり、それ以外の部分は同一性が低くてもよい。実環境との同一性が高い必要性がある部分を実ノードで実現し、それ以外の部分を仮想機械で実現することにより、実験トポロジを大規模化できる。

ただし、実験遂行者は実験の性質を十分に吟味し、仮想機械を利用できるかどうか、また、利用できる場合は実験トポロジ中のどの位置に利用するかを決定する必要がある。この吟味が十分に行われない場合は、実験結果に影響をおよぼす可能性がある。

実験トポロジ中、仮想機械が利用できる部分の例を以下にあげる。

- 実験トポロジ中トラフィックを単に転送するだけのネットワークを構成するノード
- ユーザの動作を模倣し実験トラフィックを生成するノード

### 2.2 仮想機械利用の困難さ

大規模な実ノードによるテストベッドでは、実験遂行者のノード設定および実験トポロジ構築のための負荷を低減するため、支援システムが用意されていることが多い。このようなシステムを利用している実験遂行者にできるだけ仮想機械と実ノードの区別なく扱えることが望ましい。

我々が想定するテストベッドでは、多数のノードによる実験トポロジを容易に構築するために、VLAN

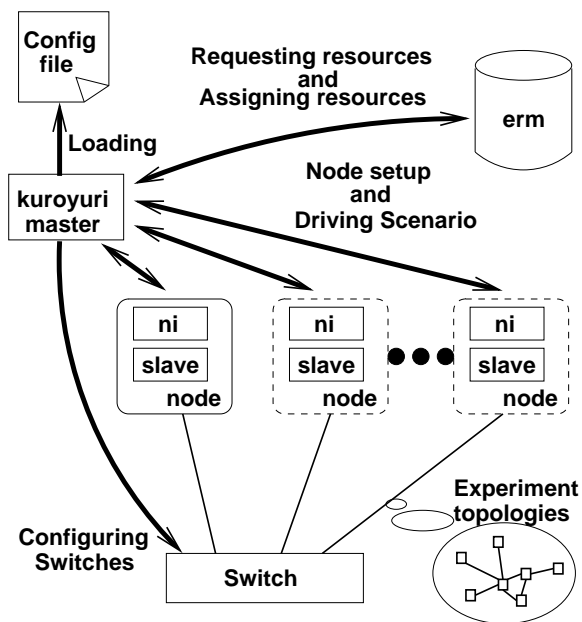


図 1: SpringOS の概要

などの仮想ネットワーク構築手法を用いて、物理接続を変更することなく実験トポロジを構築する。実ノードを利用する場合は、実ノードが接続されているノードのネットワークインターフェース (I/F) が収容されているスイッチの設定を行えば良いが、仮想機械を利用する場合は、仮想機械が動作している実ノードの I/F を検索し、その I/F が接続されているスイッチの設定を行う必要がある。

また実験支援システムでは、実験遂行者が記述した手順 (シナリオ) に従い自動的に実験を遂行する機能を持つことも多い。仮想機械を利用した場合は、仮想機械上で実行される実験駆動のためのシナリオと、仮想機械を動作させるための実ノード用のシナリオが必要となる。しかし、実験遂行者が混乱することを避けるため、仮想機械を動作させるためのシナリオは実験遂行者からは隠蔽することが望ましい。

以上のように、仮想機械を利用した場合は、仮想機械を動作させるノードの制御が必要となり、実験リソースの管理および実験手順が複雑化する。

我々は大規模なテストベッドで実ノードへのソフトウェアの導入および、実験トポロジの構築を支援するためのシステムとして、SpringOS を開発してきた。本章では、SpringOS の概要と SpringOS で仮想機械を扱うための拡張について述べる。

表 1: SpringOS の主要モジュール

要素	役割
erm	リソースの状態・属性管理 実験へのリソース割り当て
swconf	スイッチの設定
kuroyuri master	設定記述の認識 ノードへのソフトウェア導入指示 ノードへのシナリオ配布 シナリオ実行補助 他のモジュール制御
ni	ノードへのソフトウェア導入 (ディスクレス OS 上で動作)
kuroyuri slave	ノードでのシナリオ実行

### 3 SpringOS/VM

SpringOS は実験支援システム全体の名称であり、さまざまな機能を持つモジュール群により構成されている。実験遂行者は各種設定を記述した 1 つのファイルを用意するだけで、SpringOS により自動的に実験が遂行される。SpringOS の概要を図 1 に示す。

本論文では、仮想機械を取り扱えるよう拡張した SpringOS を SpringOS/VM と呼ぶ。

#### 3.1 SpringOS

SpringOS では実験の実行主体である kuroyuri master が実験遂行者による設定ファイルを読み込んだ後、設定ファイルに従い他のモジュール群と連携し実験を遂行する。SpringOS の主要モジュールを表 1 に、処理手順を以下に示す。

1. kuroyuri master による実験遂行者の設定ファイルの読み込み
2. erm によるリソースの割り当て
3. ノードの起動とノード上での ni 起動
4. ni と kuroyuri master によるノードへの OS およびアプリケーションの導入
5. ノード再起動
6. kuroyuri master から swconf にスイッチの設定リクエスト送信
7. スイッチ設定による実験トポロジ構築
8. kuroyuri master と kuroyuri slave によるシナリオ実行

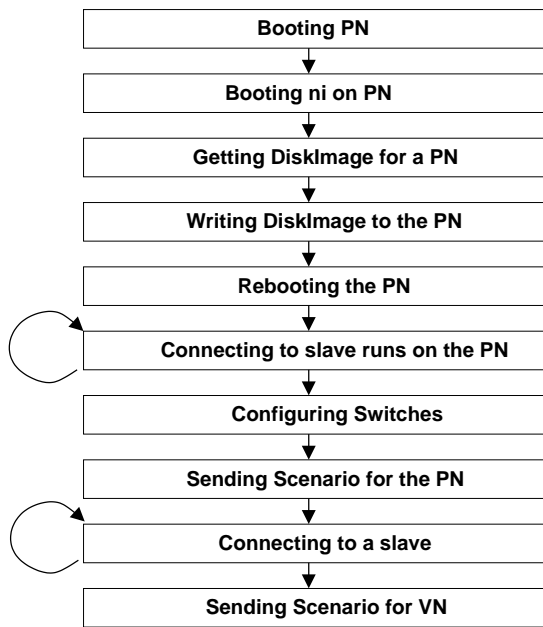


図 2: SpringOS/VM の処理手順

我々は、テストベッドに実験用と管理用のネットワークが別々に用意されていることを想定しており、SpringOS による管理用の通信は管理用ネットワークを利用して行われる。これは、管理用の通信の実験への影響を防ぐためである。管理側のネットワークの IP アドレスは DHCP により、常に同じ IP アドレスが自動的に設定される。

SpringOS の詳細については、[5] に、実験ノードへのソフトウェア導入については、[6] にまとめられている。

### 3.2 SpringOS/VM の処理手順

仮想機械のノード設定の際の、処理手順を図 2 に示す。実ノードを実験用ノードとして利用する場合は、実ノードにシナリオが送信されれば実験が開始される。しかし、仮想機械を利用する場合は、実ノードに送信されるシナリオは仮想機械の設定および起動のためのものである。したがって、仮想機械設定、起動用のシナリオを実ノードに送信した後、仮想機械上で kuroyuri slave が起動するまで待機し、slave 起動後に仮想機械用のシナリオを送信する。

### 3.3 仮想機械を利用した場合のネットワーク構成

前述の通り SpringOS はテストベッドに管理用と実験用のネットワークが別々に用意されていることを想定している。同様に仮想機械も管理用および実験用のネットワークに接続することとする。仮想機

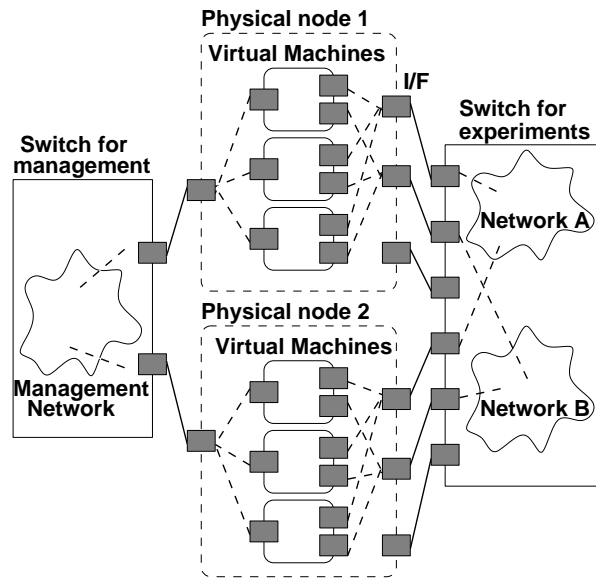


図 3: 仮想機械を用いた実験トポロジ

械を利用した簡単な実験トポロジを図 3 に示す。図中の破線で示された長方形が実ノードであり、その中の実線の長方形が仮想機械を示す。実ノードと仮想機械にはそれぞれ I/F が用意されている。I/F の物理的な接続を実線で、仮想的な接続を破線で示した。今回は VMWare の vmnet-bridge を用い、仮想機械の I/F を、実ノードの I/F が接続されているネットワークに接続されているようにみせることができる。この機能を用い、仮想機械の各 I/F を実ノードが接続されているネットワークに接続する。

### 3.4 リソース管理

erm のデータベースには実ノードの情報が 1 台ずつ記述されている。仮想機械の情報については専用の記述方法を用意し、同じデータベースで管理することとした。図 4 に erm のリソース定義ファイルの一例を示す。これは実ノード physnode001 に関する定義である。このノードには I/F が合計 7 つあり、net からはじまる行の 2 カラム目の値で、その用途が管理用または実験用かを示している。manage が管理用であり experiment が実験用、そして empty はケーブルが接続されていないことを示す。またそのほかにも I/F の種類や、MAC アドレス、そして接続されているスイッチとそのポート番号などの情報が格納されている。仮想機械を扱うため、これらの情報に加え実ノードで起動し得る仮想機械の識別子を付加した。この識別子は、実ノードの設計上動作させるこ

```

node physnode001 [
  bootdisk,SCSI
  net,manage,FastEthernet,00:00:4C:0F:77:C8,,172.16.2.1,"Linux" "eth0" "FreeBSD" "fxp0"
  net,empty,FastEthernet,00:00:4C:0F:77:C9,,,"Linux" "eth1" "FreeBSD" "fxp1"
  net,experiment,FastEthernet,00:C0:95:C4:52:70,"ethsw001,4/25",,"Linux" "eth0"
  net,experiment,FastEthernet,00:C0:95:C4:52:71,"ethsw001,5/11",,"Linux" "eth1"
  net,experiment,FastEthernet,00:C0:95:C4:52:72,"ethsw001,5/43",,"Linux" "eth2"
  net,experiment,FastEthernet,00:C0:95:C4:52:73,"ethsw001,6/29",,"Linux" "eth3"
  net,experiment,ATM,, "atmsw005,90",,"Linux" "atm0"
  vnode,physnode001vm01,physnode001vm02,physnode001vm03,physnode001vm04, \\  
    physnode001vm05,physnode001vm06,physnode001vm07,physnode001vm08, \\  
    physnode001vm09,physnode001vm10
]

```

図 4: erm の実ノードリソース定義

```

vnode physnode001vm01 [
  physical,physnode001
  net,manage,FastEthernet,00:50:56:08:01:11,,10.8.1.17,"Linux" "eth0" "FreeBSD" "fxp0"
  net,experiment,FastEthernet,00:50:56:08:01:12,,,"Linux" "eth1"
  net,experiment,FastEthernet,00:50:56:08:01:13,,,"Linux" "eth2"
  net,experiment,FastEthernet,00:50:56:08:01:14,,,"Linux" "eth3"
  net,experiment,FastEthernet,00:50:56:08:01:15,,,"Linux" "eth4"
  net,experiment,FastEthernet,00:50:56:08:01:16,,,"Linux" "eth5"
  net,experiment,FastEthernet,00:50:56:08:01:17,,,"Linux" "eth6"
  net,experiment,FastEthernet,00:50:56:08:01:18,,,"Linux" "eth7"
]

```

図 5: erm の仮想機械リソース定義

とができる最大台数分を記述する。

仮想機械のリソース定義例を図5に示す。SpringOSはIPアドレスの設定などのためにI/FのMACアドレスを必要とする。VMWareは、MACアドレスは指定された範囲内であれば自由に指定でき、VMWareの設定ファイルに記述することでこのアドレスを利用できる。この機能を利用し、前もってMACアドレスを決定しこのアドレスを記述することとした。

管理ネットワークに接続されている実ノードのI/Fには、DHCPサーバにMACアドレスとIPアドレスが登録されており、起動時にそのIPアドレスが自動設定される。仮想機械の管理ネットワークに接続されているI/Fにも同様に常に同じIPアドレスを設定するため、DHCPサーバに仮想機械で用いるMACアドレスとIPアドレスの登録を行った。

### 3.5 ノード割り当て

動作させるアプリケーションの負荷を考慮し1台の実ノード上で動作させる仮想機械の台数を変更するため、実験遂行者が実験の設定記述に、実ノード上で起動する仮想機械の数を指定することとした。ノード割り当て時には、実験遂行者が指定した多重

度と必要な仮想機械の総数から、必要な実ノード数を計算し、その値に予備ノード数を追加しermに実ノードの割り当てを要求する。ermはこの情報に基づき実ノードを検索し、実験遂行者が指定した多重度で仮想機械を起動できる実ノードが必要数存在すれば、その実ノードを割り当てる。

### 3.6 実ノード上での仮想機械のための設定

ermにより割り当てられた実ノードへのソフトウェアの導入の終了後、実ノード上では仮想機械のための設定を行うためのシナリオが実行される。このシナリオの主な役割は、vmnet-bridgeの起動、VMWareの設定ファイルの生成、VMWareの起動である。

**vmnet-bridgeの起動** 設定記述に従い、仮想機械の各I/Fと、実ノードのI/Fの対応を調査し、その結果をもとにvmnet-bridgeを起動する。

**VMWare設定ファイル生成** VMWareの設定ファイルはテキスト形式であり、簡単に編集できる。すべての仮想機械に共通した設定部分を前もって用意しておき、このファイルに各仮想機械独自の設定を追加する。各仮想機械に独立な設定には、各I/FのMACアドレスおよび接続形式、およびディ

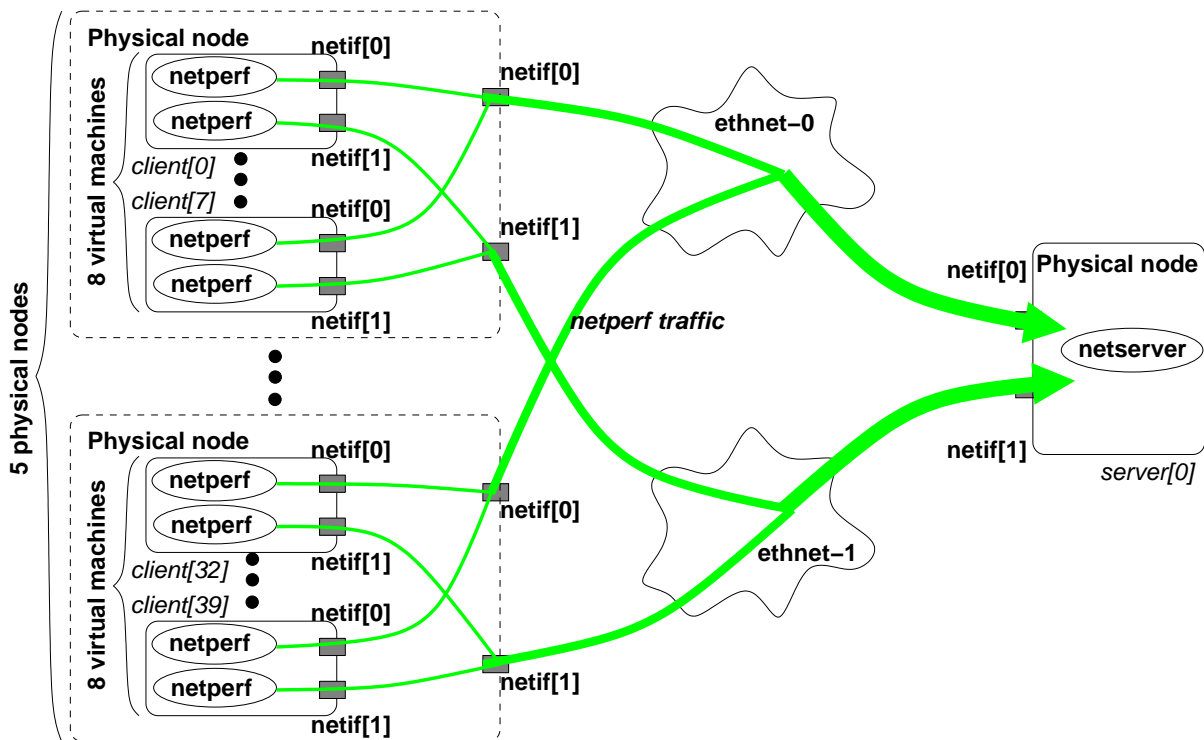


図 6: netperf 実験用トポロジ

スクイメージのファイルなどが指定される。すべての I/F は vmnet-bridge の機能を用いて、実ノードが接続されているネットワークに接続する。

また、設定ファイルに従い、VMWare のディスクイメージをダウンロードする。

**実験トポロジの構築** 実ノードの I/F が収容されているスイッチの VLAN などを設定し、実験トポロジを生成する。

**VMWare の実行** 設定ファイルで指定された数の VMWare を起動する。

### 3.7 シナリオの実行

以上の手順により実ノード上で仮想機械が起動し、実験トポロジが構築される。この仮想機械上で起動する kuroyuri slave と kuroyuri master が通信を行うことで、SpringOS と同様にシナリオを実行する。

## 4 設定記述例と動作確認

我々は SpringOS にこれまで述べた拡張を行い、StarBED 上で netperf[7] を用いた実効帯域の計測を行った。実ノード上で動作する netperf のサーバプログラムである netserv へ、仮想機械上で動作する netperf のクライアントプログラム netperf から

トラフィックを発生させるという簡単な実験である。netperf を動作させる仮想機械は、8 多重で動作させ合計 40 台用意したため、5 台の実ノードを利用することとなった。

各仮想機械の I/F は 2 つ用意し、それぞれを実ノードの別の I/F を通じ実験トポロジに接続した。各仮想機械では 2 つの netperf を起動し、それぞれ別の I/F からトラフィックを生成した。図 6 に実験トポロジを、図 7 と図 8 に設定記述の一部を示す。

図 7 は仮想機械の起動を行うための設定ファイルであり、実ノードの設定が記述されている。このクラスのインスタンスとして定義されるノードには 2 つ以上の I/F が用意されている。scenario から始まるブロックに実行されるシナリオが記述されており、I/F の設定や、VMWare の設定ファイルを生成するためのスクリプトの取得と実行、そして VMWare の起動を行う。

図 8 は仮想機械の設定である。8 多重で仮想機械を起動するよう指定されており、仮想機械を起動するための実ノード設定には図 7 で定義された vmwareC クラスを利用する。このクラスのインスタンスの仮想機械は、2 つの I/F を持ち、1 つは実ノードの 0 番

```

nodeclass vmwareC {
    method "HDD"
    disktype "IDE"
    partition 4
    ostype "Linux"
    diskimage "ftp://172.16.3.253/vmhost.gz"
    netif media fastethernet
    netif media fastethernet
    scenario {
        netifit "/usr/local/bin/ifscan"
        wakewait "/sbin/ifconfig" "/sbin/ifconfig" self.netif[0].rname\\
            (haddr(self.netif[0].ipaddr))
        wakewait "/usr/bin/wget" "/usr/bin/wget" "-q" "ftp://172.16.3.253/vmsetup.pl"
        wakewait "/bin/mv" "/bin/mv" "vmsetup.pl" "/tmp/vmsetup.pl"
        wakewait "/usr/bin/perl" "/usr/bin/perl" "/tmp/vmsetup.pl" "/tmp/vm.hint"
        for(i=0;i<_launchvnodes;i++) {
            wake "/usr/bin/vmware" "/usr/bin/vmware" "-q" "-x"\\
                ("/root/vmware/linux0"+toString(i)+"/linux.cfg")
        }
    }
}

```

図 7: VMWare 起動ノード用クラス

目の物理 I/F を通じ ethnet-0 というネットワークに接続され、もう 1 つは 1 番目の物理 I/F を通じ ethnet-1 に接続される。シナリオには I/F の設定後 kuroyuri master から netserver が起動しているノードの IP アドレスを取得し、そのアドレスに対して netperf を実行するよう記述されている。このクラスのインスタンスは nodeset ではじまる行で 40 台用意することが指定されている。また、これとは別にサーバ用のクラスも定義されており、サーバ用のインスタンスとして実ノードが 1 台用意される。

図 7 の設定部分は実験遂行者が明示的に用意する必要はなく、別ファイルに用意されているため VMWare の利用をする場合は、図 8 のように vmwareC と利用すると記述すれば良い。しかし、実験遂行者が新たにクラスの定義を行えば、実験遂行者が意図する方法で VMWare を起動することができる。また、新たなクラスを定義することで、別の仮想ノード実現手法にも対応することが可能であると考えている。シナリオ記述の詳細についても [5] にまとめられている。

これらの設定記述を用いて SpringOS/VM を実行することにより、40 台の仮想機械と 1 台の実ノードによる実験トポロジは問題なく構築され、netperf のトラフィックも発生した。これにより、SpringOS/VM

は我々が意図した通り仮想機械を起動する実ノードの設定を行い、その上で仮想機械を設定記述通りに起動できていることが確認できた。

## 5 まとめ

既存の実ノードによるテストベッドの物理的な拡張は非常にコストが高く困難である。我々は、このようなテストベッドのノードを仮想機械を用いて多重化し、更に大規模な実験トポロジを構築するため、VMWare を制御可能な実験支援システム SpringOS/VM を実装した。

また、SpringOS/VM を StarBED で動作させ、実験遂行者による設定記述通りに実ノードおよび仮想機械を制御し、実験を遂行できることを確認した。

実ノードによるテストベッドの最大の利点は実環境と同様の実装が利用できる点である。実ノードを多重化し仮想ノードを生成した場合、この利点はある程度失われる。我々は実環境との同一性を重視し、汎用 OS を利用できる計算機レベルのエミュレータである VMWare を採用した。しかしこの他にも、さまざまな抽象度で仮想ノードを実現する手法がある。このような手法には、ユーザプロセスで OS をエミュレートする User Mode Linux(UML)[8] や Cooperative Linux(coLinux)[9]、実ノードで動作している OS のプ

```

nodeclass clC {
    maxvnodes 8
    vntype vmwareC
    ostype "Linux"
    diskimage "ftp://172.16.3.253/linux.vmdk"
    netif media fastethernet via 0 net "ethnet-0"
    netif media fastethernet via 1 net "ethnet-1"
    scenario {
        netifconfig "/usr/local/bin/ifsetup/src/ifscan"
        wakewait "/sbin/ifconfig" "/sbin/ifconfig" self.netif[0].rname\\
            (haddr(self.netif[0].ipaddr))
        recv dstA
        recv dstB
        wake "/usr/local/bin/netperf" "/usr/local/bin/netperf" "-H" dstA
        wakewait "/usr/local/bin/netperf" "/usr/local/bin/netperf" "-H" dstB
        send "cdone"
    }
}
nodeset client class clC num 40
nodeset server class svC num 1

```

図 8: 仮想機械用クラス

ロセス空間を分離する jail、仮想機械モニタ Xen[10] などがある。以上であげた仮想ノード実現技術では、汎用 OS を利用できないが、より高い抽象度で仮想ノードを実現するため、一般的に仮想機械よりも高速に動作する。したがって、どの仮想化技術を実験トポロジ中どの部分に利用できるかを厳密に吟味し、さまざまな仮想ノード実現技術を併用することにより、さらに大規模な実験トポロジを構築できる。

今後は、仮想ノードを利用できる場面について検討し、それを踏まえ、SpringOS をさまざまな仮想技術に対応できるよう拡張する。

#### 参考文献

- [1] The StarBED Project. <http://www.starbed.org/>.
- [2] 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. pages 255–270, December 2002.
- [3] VMWare inc. <http://www.vmware.com/>.
- [4] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. In *2nd International System Administration and Networking Conference(SANE2000)*, May 2000.
- [5] Toshiyuki Miyachi, Ken-ichi Chinen, and Yoichi Shinoda. Automatic configuration and execution of internet experiments on an actual node-based testbed. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities(Tridentcom)*, February 2005.
- [6] 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*, pages 95–100, January 2004.
- [7] The Public Netperf Homepage. <http://www.netperf.org/>.
- [8] Jeff Dike. A user-mode port of the linux kernel. In *the Proceedings of the 4th Annual Linux Showcase and Conference*, October 2000.
- [9] Dan Aloni. Cooperative linux. In *the Proceedings of the Linux Symposium Volume One*, July 2004.
- [10] Ian Pratt. Xen and the art of open source virtualization. In *the Proceedings of the Linux Symposium Volume Two*, July 2004.