修 士 論 文

仮想ノードの特性を考慮した
大規模実験リソースの管理手法

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

鍛治 祐希

2013 年 9 月

修 士 論 文

仮想ノードの特性を考慮した
大規模実験リソースの管理手法

指導教官　篠田陽一 教授

審査委員主査　篠田陽一 教授
審査委員　丹康雄 教授
審査委員　知念賢一 特任准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1010016 鍛治 祐希

提出年月: 2013 年 8 月

概 要

インターネットを始めとする実世界のネットワークは、規模が年々大きくなっている。実世界のネットワーク規模の拡大に対応したより大規模な検証が、多様なネットワークにおいて要請されている。実験者に対して実験リソースを提供する施設として、テストベッドがある。実験者は、テストベッドの提供するリソースを用いて実世界のノードを模倣する。大規模な検証を行うためには、検証環境で模倣可能なノード数を向上させる必要がある。しかし、実世界のノードの増加に対して、検証用機器の増加を追従する事は難しい。今後実世界のネットワークの規模拡張に対応した検証を行うためには、実世界のノードと比較し少数の機器を用いて、より大規模なネットワークを模倣する必要がある。そこで本研究では、テストベッドの保有する機器を活用し、より拡張性の高い実証実験を行うことを目的とする。

さまざまな仮想化の手法を用いて仮想ノードを作り出し、複数の仮想ノードを 1 台の物理ノード上に多重化するやり方は、検証の規模を拡大するために一般的に用いられてきた手法である。本研究では、これらの仮想ノードを専有する物理マシンのリソースの種類を粒度と呼び、粒度に基づいて多重化手法の分類を行った。専有するリソースの種類が小さい程、多重化の度合いは向上するが、仮想ノードの性能や機能が小さくなる。

検証目的に合わせて適切な粒度の仮想ノードを適用することで、より検証の多重度を向上することが可能である。しかし、そのためには様々な問題が存在する。まず、目的に合わせた粒度を選択するには、各粒度の仮想化ノードの機能や性能、制約といった仮想ノードの特性を把握する必要がある。また、各多重化手法によって制御インタフェースが異なるため、検証で利用する多重化手法の実装の数だけ制御インタフェースを学習する必要がある。これらを実験者が検証のために学習することは困難である。また、実験を支援するための管理システムが開発・運用されているが、物理マシンに適したきめ細やかな制御を行う。そのため、より大規模な検証を行う際には、管理システムに係るコストが非常に大きくなる。このように、仮想ノードによる大規模な検証を行うには、様々な課題がある。

今後より大規模な検証を行うためには、仮想ノードの特性に合わせた管理手法が必要である。そこで本研究では、仮想ノード群制御システム BlackSmith を設計し、仮想ノード管理の効率化を行った。BlackSmith では、複数の仮想ノードを仮想ノード群という単位で抽象化する。仮想ノード群により抽象化されるものは 2 つある。1 つは、各ノードに適用される多重化手法である。既存の環境では、個々のノードに対して、適用される各多重化手法に対応した制御命令を記述する必要があった。本研究では、仮想ノードに対する共通のインタフェースを定義し、BlackSmith で必要に応じて制御命令を変換する。これにより、様々な多重化手法を検証で容易に用いる事が可能となる。もう 1 つは、郡内のノード数である。仮想ノード群を適用した環境では、個々の仮想ノードを直接制御しない。代わりに、群に対する制御を行う事で、郡内のノードの制御を行う。群の内部では、仮想

ノードが動作する物理マシン毎に配置した管理ノードに制御を分散することで、個々の管理ノードに係る負荷を低減する。

実際に仮想ノード群制御を可能とするため、BlackSmith を実装した。BlackSmith は、仮想ノード制御インタフェースを提供するための Agent と、群制御するためのコントローラ群からなる。これらのコンポーネントを各物理マシンへ配置することで、群制御を可能とした。これまでの実証実験と比較して、検証実験への多重化手法の導入が容易となった。また、多数のノードに対する制御に関しては、個々の管理ノードに対する負荷が低減された。

目 次

第1章　はじめに

1.1　研究背景

インターネットを始めとする広域ネットワークは、規模が年々大きくなっている。スマートフォンと呼ばれる高機能携帯電話や、Internet of Things と呼ばれる高機能センサデバイス、また家電やゲーム機等あらゆる計算機が IP アドレスを保有している。そのため、これらの装置が参加する IP ネットワークの規模は、非常に大規模となっている。

新たに技術や製品を適用するには、それらが及ぼす影響を考慮する必要がある。また、Peer to Peer（P2P）技術を始めとして、計算機同士の繋がりによるネットワークが存在する。このような種類のネットワークは、ネットワークに参加する計算機の数によってそのネットワーク特性に大きな影響を与える。このように、ネットワークの複雑性・規模性が大きくなる程、細部の変化が全体に及ぼす影響は大きい。

そのため、新たに技術を開発した際には、技術を広範囲に適用することによる影響を検証する必要がある。運用後に問題が発見された場合、問題を修正するために最悪サービスの停止を伴う。広範囲に適用される技術であるほど、サービス・機能が停止した際の損失は大きい。損失を抑制するために、実際にサービスを運用する前に、技術の検証を行いたいという要求がある。

このような検証を行うための手法として、エミュレーションがある。エミュレーションは、対象となる技術や製品を実際に模倣ネットワーク上で動作させる手法である。計算により結果を予測するシミュレーションと異なり、事前に予測が難しい問題の発見が可能となる。検証を行うためには、ネットワークを模倣するための装置を用いる。装置には物理マシンや、スイッチ、ルータ等が挙げられる。実験者はこれらの装置を接続し、検証を行うための環境 (検証環境) を構築する。

しかし、大規模な検証を行うにあたり、実験者がこれらの機器を必要となる個数用意することは難しい。機器を用意するためには、様々なコストが必要である。導入の際には、機器購入及び設置の費用が発生する。また、運用の際には、機器の動作や空調に必要な電力や、運用者を雇う費用が発生する。

個々の実験者に代わり、大規模なネットワークを提供する施設として、大規模テストベッドが存在する。大規模テストベッドは、汎用的な計算機群を持つ。計算機群は必要に応じて検証者へ提供される。これにより、検証者は自前で機器を用意すること無く、大規模な検証を行うことが可能となる。

1.2 研究目的

大規模テストベッドは、大規模な検証を行うための汎用的な環境を検証者へ提供することが使命である。実世界のネットワークの規模は飛躍的に拡大し続けており、より大規模な検証が様々なネットワークにおいて要請されている。

検証環境を構成する要素として、ノードとリンクがある。大規模な検証を行うためには、検証環境で模倣可能なノードとリンクの数を向上する必要がある。しかし、テストベッドが実際のネットワークの規模拡大に追従する事は難しい。なぜならば、ネットワークは、複数のキャリアや個々のユーザが用意するノードが組み合わされる事で成立する。インターネットを例に上げると、まず各キャリアが提供する IP ネットワークを構成するルータがある。また、キャリアが提供するネットワークを利用して通信を行う各ユーザの PC や携帯電話、家電といった端末がある。更に、ユーザに対して様々なサービスを提供するサーバがある。このように、大規模なネットワークは 1 個の代表者によって運用されるわけでは無く、多様な運用者によって運用されている。それに対し、テストベッドは少数の企業もしくは国によって運営される。このことから、実世界のノードの増加に対して、検証用機器の増加を追従する事は難しい。

今後実世界のネットワークの規模拡張に対応した検証を行うためには、実世界のノードと比較し少数の機器を用いて、より大規模なネットワークを模倣する必要がある。そこで本研究では、テストベッドの保有する機器を活用し、より拡張性の高い実証実験を行うことを目的とする。

1.3 論文構成

2 章では、これまでに行われてきた大規模な検証実験と、テストベッドの現状について述べる。3 章では、テストベッドにおいて大規模ネットワーク検証を行う際の問題点を明らかにする。4 章では、問題点の中で本研究が着目する仮想ノード制御について分析を行う。また、分析した結果を元に、仮想ノード制御の問題点に対する本研究でのアプローチを述べ、アプローチに基づき仮想ノードの管理システム BlackSmith を設計する。5 章では、BlackSmith の実装を行う。6 章では、実装に対する評価と考察をする。7 章では、本研究についてまとめる。

第2章　仮想ノードを用いた検証実験の現状

2.1　大規模実証実験

本章では、これまでに行われてきた大規模な検証実験と、テストベッドにおけるノード管理の現状について述べる。

2.1.1　テストベッドにおける検証

ネットワークエミュレーション

検証を行う手法として、ネットワークエミュレーションがある。ネットワークエミュレーションは、検証の対象や周囲の環境を構成するノードを実際に動作させる検証手法である。

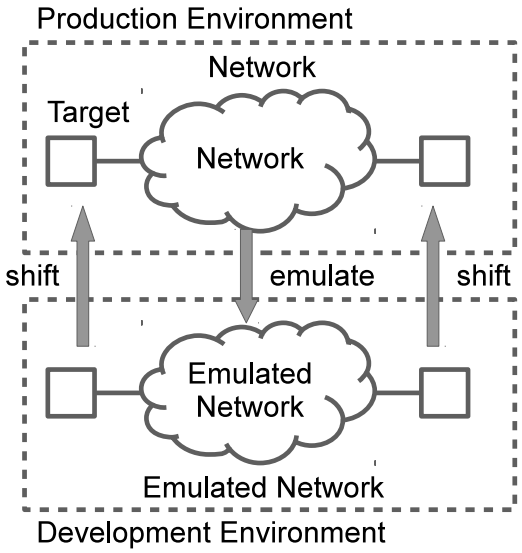

図 2.1: ネットワークエミュレーション

ネットワークエミュレーションを図 2.1 に示す。ネットワークエミュレーションでは、検証対象が実際に動作する環境と同一規模のネットワークを検証環境内で構築する。構築

した環境で検証を行う事により、本番環境で動作する際の多様な問題を事前に察知し、製品や技術の研究及び開発につなげる事が可能となる。

複数の実験者によるテストベッドの利用

テストベッドでは、より多くの検証を可能とするため、複数の実験者による並行した実験を行う。

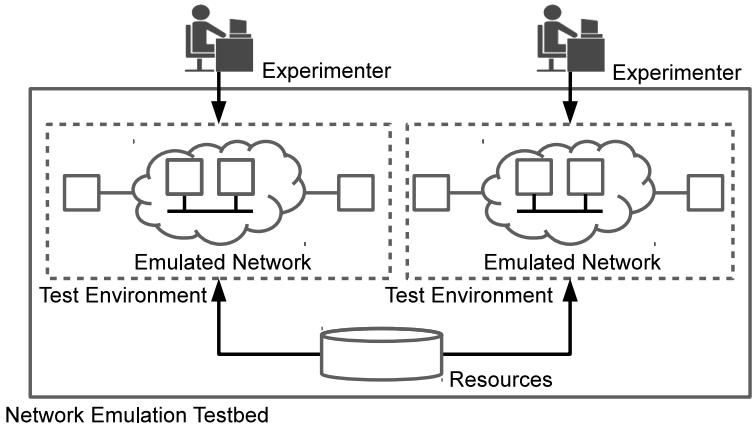

図 2.2: テストベッドの同時利用

複数の実験者によるテストベッドの同時利用の例を図 2.2 に示す。テストベッドは、保有しているリソースから、実験者が要求したリソースを必要に応じて貸し出す。それぞれの実験者は、貸し出されたリソースを用いて、それぞれの検証目的に合わせた環境を構築する。検証が終わると、リソースをテストベッドへ返却し、次の実験者がリソースを利用する。このように、保有するリソースの貸出と返却を繰り返すことで、複数の実験者によるテストベッドの利用を可能としている。

2.1.2 仮想ノードを用いた検証

1 つの物理マシンを用いて実現するノードを、本研究では物理ノードと呼ぶ。検証で利用できる物理ノード数は、物理マシンの数に依存するため、高いスケーラビリティを確保することが困難である。物理ノードに対して、1 つの物理マシンを用いて複数のノードを表現する事で、限られたリソースでより大きな検証を可能とする方法がある。本研究では、このようなノードを仮想ノードと呼ぶ。仮想ノードは物理ノードよりも性能や機能が低いが、物理マシン数を超えるノード数が実現可能なため、大規模な検証が可能となる。

仮想ノードの実現には、様々な仮想化技術や多重化技術が用いられる。以下では、各技術を用いた仮想ノードの生成方法について述べる。

トラヒック・ジェネレータ

仮想ノードを生成する方法として、トラヒック・ジェネレータと呼ばれる物理マシンを用いる方法がある。トラヒックジェネレータは、条件に基づいたトラヒックを生成する機器あるいはソフトウェアである。大規模な検証を目的として、トラヒックジェネレータを用いた機械的なトラヒックの生成が行われる。トラヒックジェネレータの特性として、非常に高速に、様々なパターンのトラヒックの生成が可能である。個々のトラヒックに対して別々のネットワーク識別子を付与することによって、各識別子を持つ仮想ノードとして利用することが可能である。一方で、トラヒック・ジェネレータによる仮想ノードは外因に基づく動作を行うことが困難である。外因の 1 つは、相互通信におけるノードの状態を元にしたものである。TCP やアプリケーションのレベルでは、ソフトウェアはコネクション等の状態を持つ。状態によっては、通信における応答の内容が異なる場合がある。もう 1 つは、動作環境によるものである。ソフトウェアが動作する環境では、実際にはオペレーティング・システムや、他のプロセスが動作する。CPU への割り込み命令や他のプロセスとの相互干渉といった影響により、通信のタイミングが異なる場合がある。予めパターン化されたトラヒックを生成するトラヒック・ジェネレータにおいて、このような外因を考慮したパターンのトラヒックを生成することは困難である。実際のネットワーク・ノードは、様々な外因を受け動作するため、複雑な振る舞いをする。この事から、単純なトラヒックの生成は実際のノードの振る舞いを模倣できるとは言えない。多様な外因の影響を考慮するためには、模倣したノードでも実際にソフトウェアを動作させトラヒックを生成する必要がある。

プロセス

プロセスは、仮想的なメモリ空間を生成することにより、複数のプログラムを並列に動作させる手法である。プロセスを用いた検証の例として、http クライアントの模倣がある。各 TCP ポートに紐付いたプロセスは、それぞれがポート番号という識別子を持って動作する。そのため、TCP レイヤの通信においては、個々のプロセスを 1 つのノードとみなす。プロセス多重は UNIX システムにおいて一般的な機能であり、容易に実現することが可能である。

仮想マシン

仮想マシンは、計算機を構成する各デバイスレベルでエミュレートすることにより、動作する OS 上に仮想的な計算機を生成する手法である。仮想マシンを用いた仮想ノードの例として、ルータの模倣がある。各仮想マシン上では OS が動作する。各 OS はルーティングテーブルを持っており、通信の際にはルーティングテーブルを参照して動作する。そのため、IP レイヤの通信においては、各仮想マシンを 1 つのノードとみなす。仮想マシンを実現するソフトウェアとしては、Xen や KVM がある。

2.2 検証規模に合わせた仮想ノードの制御粒度

仮想ノードに適用される多重化手法によって、専有可能なリソースは異なる。

表 2.1: 各粒度で専有するリソースの種類

粒度＼専有リソース	CPU	MEM	FS	ADDR	PORT
物理マシン	✓	✓	✓	✓	✓
仮想マシン	—	—	✓	✓	✓
プロセス	—	—	—	—	✓

FS: File System　ADDR: IP Address
MEM: Memory

各多重化手法と専有可能なリソースの例を表 2.1 に示す。物理マシンによるノードは、物理マシンの全ての機能を 1 つのノードが用いるため、全ての機能を専有可能である。一方、仮想マシンやプロセスといった多重化手法が適用された仮想ノードでは、リソースを共有するため、専有可能なリソースが制限される。例えば、仮想マシンは CPU 時間やメモリ量といった物理マシンのリソースを利用して生成される。そのため、個々の仮想マシンがこれらのリソースを専有する事はできない。また、プロセスは 1 つのカーネル上で動作する。そのため、カーネルが持つルーティングテーブルやファイルシステム等のリソースを専有することができない。

一方、専有するリソースが少ないほど、論理的に多重化の度合いを向上させる事ができる。例えば、仮想マシンはマシンのデバイス等の低レベルな機能を模倣する。また、仮想マシンを検証で用いるために OS が必要である。そのため、CPU やメモリといったリソースを大きく利用する。これに対しプロセスは専用のデバイスや、専用の OS を必要としない。そのため、仮想マシンに対してプロセスの方がより低いコストで多重化することが可能であり、より多重化の度合いを向上させる事が可能であるといえる。このように、仮想ノードに適用される多重化手法によって、多重化の度合いが異なる。本研究では、専有されるリソースの種類の違いを、ノードの粒度と呼ぶ。多重化に依るコストを下げるためには、仮想ノードの粒度をより小さいものにする必要がある。

2.3 仮想ノードを用いた検証の規模

限られたリソースを用いてより大規模な検証を行うために、ノードの多重化が行われている。ノードとは、ネットワークにおける端点を表現する言葉である。ノードの多重化とは、1 つの機材の中でノードを複数表現する手法である。多重化を行うにあたって、ソフトウェアレベルでノードを実現する手法が用いられる。ソフトウェアレベルで表現されるノードを、ここでは仮想ノードと呼ぶ。仮想ノードを用いた検証の例としては、宮地

らによる実験 [1] がある。この実験では、仮想マシン多重を用いて 60 台の物理マシンを用いて 1100 のノードを実現している。この検証では、物理マシン数に対して約 20 倍の仮想ノードを実現している。また、知念らによる実験では、プロセス多重を用いて 1 台の物理マシンを用いて 2048 個のノードを実現している。この検証では、物理マシン数に対して約 1000 倍の仮想ノードを実現している。このように、仮想ノードを用いる事で、より規模拡張性の高い検証実験を行うことが可能になる。

2.4 検証による多重化手法の違い

仮想ノードに用いられる仮想化手法は、検証内容により異なる。宮地らによる検証では、センサデバイスの模倣に仮想マシンが用いられた。また、知念らによる検証では、HTTP クライアントとしてプロセスが用いられた。仮想ノードの実現のために、多様な仮想化技術や多重化技術が用いられる。検証で用いられる多重化手法は、検証の目的と手法毎の制約に依存する。

制約の 1 つに、ノードの識別子がある。ノード同士が実際に通信を行うことから、ノードは互いを識別する必要がある。識別のために、ノードには様々な識別子が付与される。TCP/IP プロトコル・スタックにおいて、識別子として IP アドレスやポート番号が用いられる。また、それ以外にも MAC アドレスや、アプリケーション固有の識別子が考えられる。他の制約として、ノードの機能・性能による制約がある。例えば、プロセス多重であれば、1 つのカーネルで複数のプロセスを並列に扱う。複数の仮想ノードで 1 つのカーネルを共有する事になるため、個々の仮想ノードで異なるカーネルの設定は不可能である。そのため、ルーティングテーブル等のカーネルの機能的制約に束縛される。一方で、仮想マシン多重であれば、1 つの仮想ノードが 1 つのカーネルを持つ事になるため、個々の仮想ノードに対して個別にカーネルの設定が可能である。しかし、複数の仮想マシンで 1 つの物理マシンを共有することになるため、メモリ量や CPU 時間等の計算機の性能的制約に束縛される。これらの制約から、実験者は目的に合った多重化手法をノードに適用する。

第3章　より大規模な検証に対する問題点

3.1　既存の実験支援システムの問題点

本章では、これまで行われてきた取り組みと超大規模な環境で仮想ノードを管理する際の課題について述べる。

3.1.1　SpringOS

SpringOS は、StarBED の実験者が実験を容易に遂行する事を目的とした実験支援ソフトウェア群である。以降で示すように、SpringOS で扱う実験ノードは静的に配置された物理マシンであり、それに付随するシステムの制約により、仮想マシンを SpringOS で実験ノードとして扱うことはできない。

Resource Manager

Resource Manager[2]（RM）は、実験資源の割り当て情報を管理する機構である。StarBED では、物理マシンをノードとして実験者に貸し出しており、特定の条件に一致するノードのリストを RM から取得することができる。また、ノードの構成情報が記述されたデータベースを保持しており、物理マシンを制御するための IP アドレスや、ネットワークインタフェース等の物理マシンの構成情報を取得することができる。しかし、RM はノードの情報を静的に保持するため、動的に生成された仮想マシンを新たに実験ノードとして追加することができない。実験者が仮想マシンを実験で利用するためには、実験者が仮想マシンの生成と制御を独自の手法で行う必要がある。

Kuroyuri

Kuroyuri[3] は実験記述言語処理系である。実験シナリオと呼ばれる処理手順を記述することで、多数のノードからなる実験を容易に行うことが可能である。Kuroyuri はマスタ・スレーブモデルを採用しており、全体の実験シナリオを制御する Scenario Driver[2]（SCD）の実装である Kuroyuri Master(kuma) と、各ノード上で実験シナリオを実行する Scenario Agent[2]（SCA）の実装である Kuroyuri Slave(kusa) から成る。kuma と kusa を用いた物理ノードの制御について、図 3.1 に示す。kuma は、実験者が直接制御する管理

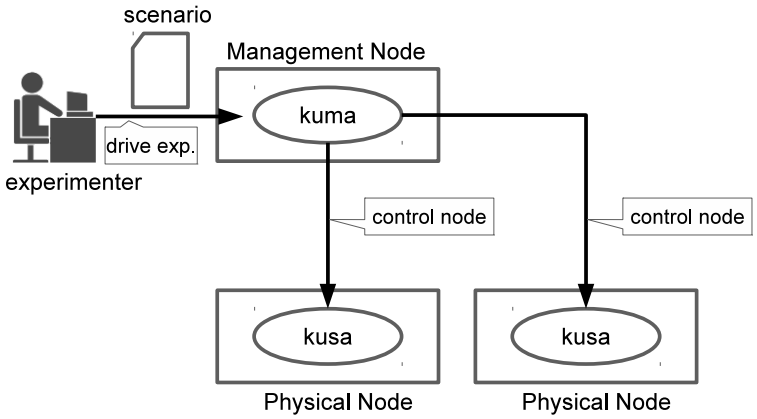

図 3.1: SpringOS による物理ノードの制御

ノードに配置する。また、kusa は制御対象の各物理マシン上に配置する。実験者から kuma に対して実験シナリオと呼ばれるスクリプトが与えられると、kuma はスクリプトに基づき、各物理ノード上で動作している kusa へ制御命令を発行する。kusa は、制御命令に基づき、自身が動作している物理ノードの制御を行う。

しかし、Kuroyuri では SCA を制御を行う実験ノード毎に管理ネットワーク上に配置する必要があるが、動的に生成された仮想マシンは施設から制御用 IP アドレスを取得することができないため、仮想マシンを実験ノードとして SCD から制御することはできない。

3.1.2 SpringOS/VM

SpringOS/VM[4] は SpringOS で仮想マシンを取り扱う拡張である。SpringOS/VM では、ノードの多重度やネットワークインタフェースの構成から仮想マシンを自動的に生成することが可能である。

SpringOS/VM での実験ノードの生成と制御について、図 3.2 に示す。RM への仮想ノードの登録や、DHCP サーバへ制御用 IP アドレスの登録を行うことで、物理マシンと同様に仮想マシンを SpringOS から利用することが可能である。しかし、施設のサーバ群の情報を動的に変更する必要があるため、本来施設が管理すべき物理マシンの情報を変更してしまう可能性がある。これにより、責任分界点が明確でなくなる危険性がある。また、SpringOS/VM では仮想ノードと物理ノードを同一に扱うため、生成された全ての実験ノードに対して、SCD から制御メッセージを送信する必要がある。そのため、実験の規模が大きくなると、管理ノードへの負荷が大きくなり、スケーラビリティを確保することが困難となる。

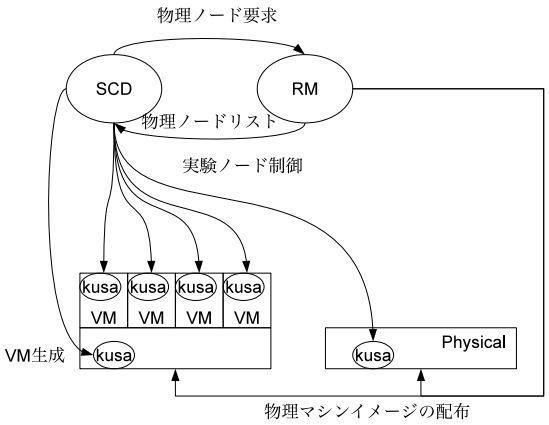

図 3.2: 既存のサポートツールを利用した多重化ノードの扱い

3.1.3 XENebula

XENebula[5] は、仮想マシン技術 Xen[5] を用いて、実験ノードによる柔軟なクラスタ環境を構築するためのツールセットである。XENebula では、実験者側で仮想マシンの生成と管理を行う。そのため、施設の管理するデータベースへ変更を加える必要が無い。しかし、Xen 以外の多重化方法については考慮されていない。また、生成したノードの制御は事前に実験者が記述した設定ファイルにより行う。そのため、実験者が高度な知識と経験を有す必要がある。

3.1.4 Network Experiment Environment

Network Experiment Environment[6]（NEE）は、入れ子型実験環境を構成する 1 つの実験単位である。NEE は、SpringOS でのノード管理とネットワーク管理についての拡張を行うことにより、実験者用の管理システムを構築することにより実現されている。これにより、実験者がノードやネットワーク等の新規リソースを追加する事が可能である。しかし NEE では、ノードの生成に関して、仮想マシンと物理マシンの概念が完全に分離しているため、利用者は物理マシンについても明示的に管理する必要がある。また制御に関しては、SpringOS/VM と同様に仮想マシンと物理マシンを同一に扱い制御するため、スケーラビリティを確保することが難しい。

3.2 仮想ノードの専有するリソースと粒度

2 章で述べたように、仮想ノードに適用される多重化手法によって、専有可能なリソースは異なる。そのため、専有するリソースを元に各粒度の分類を行った。

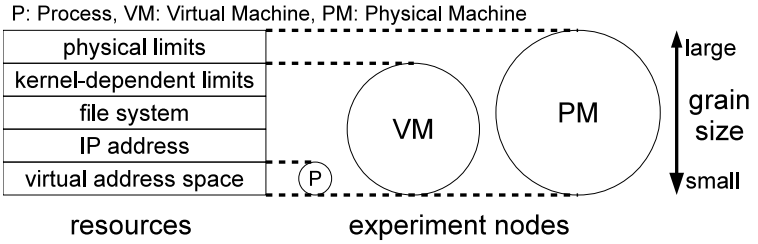

図 3.3: 専有するリソースとノードの粒度

専有するリソースと粒度の関係を図 3.3 に示す。ここでは、物理マシンの持つリソースを 5 つに分類した。物理的制限に関するリソースは、CPU 時間やメモリ量といった物理的なリソースである。このリソースは最も粒度の大きなリソースである。これらのリソースについて厳密な検証を行う場合には、多重化による仮想ノードの生成は行えない。従って、物理マシンをノードとして扱うことになる。カーネルの制限に関するリソースは、ルーティングテーブルやポート番号である。これらの厳密な検証を行うためには、各仮想ノードがそれぞれカーネルを持つ必要がある。従って、仮想マシン等の OS を含めた仮想化手法を利用する必要がある。また、ファイルシステムや IP アドレスといったリソースに関しても、仮想マシン等の仮想化手法を利用する必要がある。仮想アドレス空間は、プログラムが動作するための最も粒度の小さなリソースである。このリソースだけを専有するものには、プロセスが挙げられる。このように、専有するリソースにより、割り当てが可能な仮想ノードの粒度が異なる。

3.3 多様な多重化手法への対応

3.3.1 各多重化手法の制御インタフェース

プロセスの制御インタフェース

プロセスを制御するには、幾つかの方法がある。1 つは、シェルを用いてプロセスを制御する方法である。この方法は、プロセスを生成する際に実行するプログラムを指定する。ping のプロセスを生成する例は、次のようになる。

```
% ping www.jaist.ac.jp
```

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
int main(void)
{
    int status = 0;
    int pid;
    pid = fork();
    if (pid == 0) {
        execl("ping", "www.jaist.ac.jp")
    }
    waitpid(pid, &status, 0);
    return 0;
}
```

図 3.4: プログラム中での子プロセスの生成

もう 1 つは、プログラム中で子プロセスの生成を指定する方法である。ping のプロセスを生成する例を、図 3.4 に示す。この例では、fork 関数によって新しいプロセスを生成し、新しいプロセスでは、execl 関数によって目的のプログラムを実行する。

仮想マシンの制御インタフェース

仮想マシンの場合、実装毎に異なる制御インタフェースを持つ。ここでは、KVM と Xen を例に挙げ、仮想マシンの制御について説明する。KVM の場合は、シェルから実行する仮想マシンの構成を全て指定する。構成の種類としては、ディスクイメージや、メモリ量、割り当てる CPU の個数等がある。KVM による仮想マシンの生成の例を次に挙げる。

```
% kvm -hda vm001.img -m 512
```

この例では、vm001.img というディスクイメージを持つ、メモリ 512MB の仮想マシンを実行する。

Xen の場合は、シェルから実行する仮想マシンを指定する。仮想マシンの構成は予めファイルに指定しておくことが可能である。コマンドの例は、次のようになる。

```
% xm create /etc/xen/vm001
```

この例では、vm001 という名前の仮想マシンを実行する。

3.3.2　多重化手法に基づいた制御の必要性

仮想ノードに係るコストを抑えるためには、仮想ノードの粒度をより小さいものにする必要がある。そのためには、選択可能な粒度が十分に存在する必要がある。しかし、仮想ノードによって、制御インタフェースは異なる。計算機レベルでの仮想化では、計算機を制御するための制御インタフェースが用意される。一般的に、計算機の電源の ON・OFF や、ハードウェア構成の変更等が可能である。プロセスレベルでの仮想化では、プロセスを制御するための制御インタフェースが用意される。一般的に、プロセスの生成・破棄、開始・停止等が可能である。また、同一の仮想化手法であっても、各実装によってインタフェースが異なる。例えば、計算機レベルでの仮想化の実装である Xen と KVM では、同一の機能でも制御インタフェースは異なる。制御するためのコマンド名若しくは関数名、引数の名前・順序、与えるパラメータの形式といったものがあげられる。このように、多重化の対象や実装により様々な制御インタフェースが存在する。

検証を行うためには、これらの制御インタフェースを通して各仮想ノードを制御する必要がある。物理ノードの制御であれば、IPMI のような共通の制御インタフェースがあり、実験者は多くの物理ノードを同様に制御することが可能である。しかし、実験者がインタフェースの異なる個々の多重化手法に対応して制御を行う事は、困難である。まず、実験対象によって実験者は異なる制御を記述しなければならない。そのため、仮想ノードを制御する際には適用されている多重化技術を参照し、適当な制御インタフェースを見つけなければならない。また、多重化の対象によって制御インタフェースが異なるため、仮想ノードを制御する際には、制御対象の仮想ノードと、仮想マシンやプロセスといった多重化の 2 つの面で制御を行う必要がある。このように、物理マシンの制御と比較して、多重化手法について実験者が考慮すべき箇所が多い。

3.4　仮想ノードに対する適切な多重化手法の適用

より大規模に検証を行うためには、ノードの多重度を向上する必要がある。そのためには、仮想ノードに必要なリソースを抑える必要がある。各多重化手法により、利用するリソースの大きさが異なるため、できるだけ小さな粒度を適用する事でリソースの消費を抑える事が可能である。一方で、各仮想ノードは実験者が要求する機能や性能を満たす必要がある。つまり、実験者が要求する機能や性能を満たす仮想ノードを、適用可能な粒度の内最も小さな粒度で実現する必要がある。仮想ノードに求める特性は、様々なものがある。例えば、メモリ量、通信速度、必要とするネットワーク識別子等が挙げられる。検証の目的により、仮想ノードに求める特性は異なる。実験者は、検証の度に必要なリソースを求める必要がある。また、各仮想化手法の特性も様々なものがある。例えば、プロセスはメモリ空間やポート番号の専有は可能であるが、IP アドレスは専有できない。実験者は、様々な多重化手法について事前に調査し、このような特性を把握しなければならない。このように、仮想ノードに求める特性と、各仮想化手法の特性を理解する必要があ

り、最適な多重化手法を選択することは容易ではない。

3.5 仮想ノードの最適配置問題

計算機により、保有するリソースの量や性質は異なる。例として、メモリ量やCPU速度、CPUのアーキテクチャといった要素が挙げられる。仮想化技術により、効率的に動作するために必要なリソースは異なる。一般的に、仮想マシン多重は比較的大きなメモリ空間が必要であり、高速に動作させるためにはVTと呼ばれるCPUの機能を必要とする。これに対し、プロセスは自身が必要とするメモリ空間があれば実現できる。そのため、どの物理マシンでどの仮想ノードを動作させるかを考慮しなければならない。

3.6 管理対象の増加・複雑化

物理ノードを用いた検証では、管理対象はノード数分で良い。一方、仮想ノードを用いた検証では、仮想ノードに加えてそれらが動作する物理マシンの管理が必要である。例えば、仮想ノードを動作させるためのソフトウェアの導入や、ネットワークインタフェースの設定が挙げられる。また、仮想ノードを制御するには、対象が動作する物理マシンを経由して制御する必要がある。そのためには、仮想ノードと物理マシンの関連性を実験者が保持する必要がある。このように、実験者が管理すべき対象と情報が物理ノードを用いた検証と比較し、増加する。

3.7 管理システムへの負荷

また一方で、仮想ノードの制御のための負荷を抑える必要がある。実験支援ソフトウェアであるSpringOSでは、ノードを中央集中制御している。この制御方式では、1つの管理ノードが、検証に利用する個々のノードと直接通信を行うことにより制御を行う。現在のモデルでは、管理するノードの数に応じて、管理ノードのリソースが必要となる。超大規模な環境では、管理システムに対する負荷の増大に対して、1つの管理ノードでは十分に制御できない。管理ノードのリソースとして、ポート番号が挙げられる。管理ノードが各ノードを制御するとき、1つのノードに対して1つのポート番号を利用する。しかし、TCP及びUDPのポート番号は2の16乗であるため、同時に利用可能なポート数は最大でも約6万である。仮に1000万のノードを管理すると考え、1つのノード制御に係る秒数を1秒とすると、全ノードの制御に要する時間は約3分必要になる。応答を必要とするような制御であった場合は更に多くの時間を要する。また、別のリソースとして通信帯域が挙げられる。ノードを管理するためには、制御した結果を応答として集取しなければならない。しかし、多くのノードから一斉に応答が発生した場合、通信帯域を圧迫する。仮に1000万のノードが各1kByteの応答を行うと考えると、82Gbpsの通信が発生する。こ

の通信に対応するためには、10Gbps の NIC を 8 枚以上搭載した計算機が必要になる。このように、既存の制御方式では、超大規模な検証を行う際に管理システムへの負荷が問題となる。

第4章　仮想ノード群制御システム BlackSmithの設計

4.1　管理システムへの要求

前章で述べたように、仮想ノードをより大規模な検証において用いるためには、幾つかの問題が残っている。本研究では、多重化ノードの制御インタフェースの違いと、仮想ノード管理コストの増加に対応することで、より大規模な検証に耐えうる管理システムを設計する。

4.2　仮想ノード群による抽象的な管理

これまで、個々のノードは仮想ノードと呼ばれる単位で制御されてきた。個々の仮想ノードに対する詳細な制御を行うため、実験者が個々のノードを直接制御することで実験を行った。一方、複数のノードに対する管理を容易にするため、実験スクリプト上で複数のノードをまとめて制御する方法が開発されている。この方法により、実験者スクリプトの記述に対しては、容易に行う事が可能である。しかし、実際の管理を行う上では、1 対 1 の制御が行われるため、管理システムに対する負荷は高い。また、多重化手法の違いに対応して実験スクリプトを記述しなければならない。超大規模なノード数を管理する際には、ノード単位での制御は負荷が高い。そこで、管理の単位を個々のノードではなく、特定のノード・グループ単位で制御することで、管理コストの低減を行う。例えば、10 万からなる P2P のオーバーレイネットワークがあり、それらが 1 万 AS からなる IP ネットワークに及ぼす影響の検証があるとする。この場合、全体として仮想ノードが 11 万からなる検証を行う必要がある。しかし、P2P のオーバレイネットワークを構成している仮想ノードを 1 つのグループとして管理し、また IP ネットワークを構成する仮想ノードを 1 つのグループとして管理することで、2 つのノードグループを制御するだけで全ての仮想ノードを管理することが可能となり、管理システムに対する負荷が減少するとともに、検証環境全体の見通しを向上させる。このように、本研究では、複数の仮想ノードを仮想ノード群 (群) という単位で抽象的に扱う事で、管理システムへの負荷を減らす。群単位でのノード管理を図 4.1 に示す。群により抽象化されるものは 2 つある。1 つは、各ノードに適用される多重化手法である。既存の環境では、個々のノードに対して、適用される各多重化手法に対応した制御命令を記述する必要があった。群によって、多様な多重化手

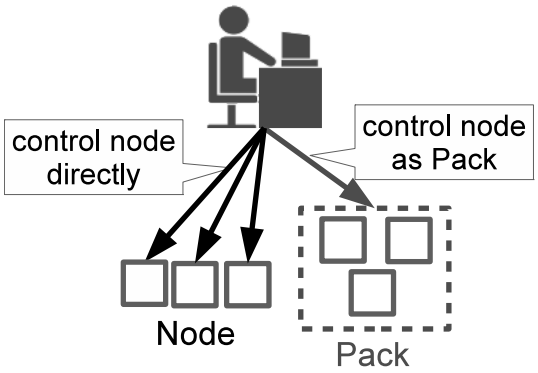

図 4.1: 仮想ノード群による検証環境の管理

法に対応したノードの制御を行う。もう 1 つは、郡内のノード数である。仮想ノード群を適用した環境では、個々の仮想ノードを直接制御しない。代わりに、群に対する制御を行う事で、郡内のノードの制御を行う。これにより、様々な多重化手法を検証で容易に用いる事が可能となる。また、多数のノードを制御するための負荷を低減することが可能となる。

4.3 共通のインタフェースによる仮想ノードの制御

仮想ノードに対して多様な多重化手法を適用可能とするため、異なる制御インタフェースを持つノードに対して、統一的な制御インタフェースを定めた。このインタフェースでは、先に述べたトラヒック・ジェネレータのように、仮想ノードを動作させるための抽象的な制御を提供する。仮想ノードの動作は、以下の用に分類される。まず、仮想ノードの生成である。定められた仕様により、動的にノードを生成する。仮想マシンであれば、必要なディスクイメージや、プログラム、メモリ量の指定による仮想マシンの起動にあたる。また、プロセスであれば、必要なプログラムの配布にあたる。次に、仮想ノードの実行である。仮想マシンであれば仮想マシンの電源投入と実験スクリプトの起動にあたる。プロセスであれば、プログラムの実行がこれにあたる。最後に、ノードの破棄である。実験終了時に、生成したノードを全て破棄する。仮想マシンであれば、生成した仮想マシンの停止と破棄にあたる。プロセスであれば、生成したプロセスの終了がこれにあたる。

4.4 動的に生成される仮想ノードの管理

多重化により新たに生成された仮想ノードは、実験中に制御を行なうために、個々の状態を管理する必要がある。テストベッドでは、施設の持つ静的なリソースを管理するためのリソース管理システムを持つ。テストベッドは複数の実験者にリソースを貸し出す形を

とるため、施設の管理者がこのリソース管理システムの管理を行なっている。実験者は、リソース管理システムへリソースの情報を問い合わせる事で、リソースの制御を行う。この仕組は複数の実験者によるテストベッドの共同利用の際に、提供する機器の貸出を明確に管理することが可能である。また、情報の登録は施設の管理者が行うため、登録された情報に関しては施設側が明確な責任を持つ。一方、実験者が動的に仮想ノードを検証で用いるためには、動的に生成されるリソースを管理しなければならない。動的に生成される仮想ノードを管理するための枠組みがこれまでに考案されている。SpringOS では、動的に生成される仮想ノードに関しても、施設のデータベースへ登録する事が可能である。また、NEE[6] では、施設から切り離された管理機構を実験環境ごとに生成し、その一部であるデータベースへ登録する。前者の方法では、施設から提供される物理マシンのように仮想ノードを扱うことが可能になるため、現在の SpringOS と親和性が高い。一方で、施設が管理するサーバへ実験者が介入する事になるため、実験者側と施設側との責任分界点が明確でなくなる可能性がある。また、大規模な仮想ノードの動的な生成に伴い、データベースへのアクセスが急激に発生する。これによりデータベースのパフォーマンスが一時的に低下すると考えられる。施設の提供するデータベースは他の利用者と共同で利用するため、パフォーマンスの低下は好ましくない。そのため、本研究の目的である大規模な仮想ノード数の実現という観点から、本研究では実験環境毎のデータベースへ生成された仮想ノード群を登録する方式を採る。

複数の物理マシン上で動作する仮想ノードのグループに対して、グループ全体を制御するためのインタフェースを持たせることで、仮想ノードを群として制御し、目的とするネットワークへの出力を得ることが可能になる。仮想ノード群に対する一様な制御を、中央管理ノードから個別の部分管理ノードへ分散することで、仮想ノード群の制御に必要な実験コストを削減することが可能である。中央管理ノードは、詳細な制御が必要な少数の仮想ノードと、ネットワーク上の通信を模倣する仮想ノード群について制御を行う。これによって、中央管理ノードが制御に必要なコストを減少する。

4.5 仮想ノード管理コストの分散

SpringOS では、マスタが個々の仮想ノードのスレイブと直接通信を行うため、個別の仮想ノードをきめ細やかに管理することが可能である (図 4.2)。このモデルでは、仮想ノードの数に応じて、それらを管理するために、管理ノードのリソースを消費する。そのため、非常に大規模な実証実験を行う時には、管理ノードの持つリソースを考慮する必要がある。1 つの IP アドレスに割り当てが可能なポート数は 65535 である。SpringOS では、1 ノードの制御に管理ノードのポートを 1 つ利用するため、同時に制御可能な仮想ノードの数は、最大でも 65535 までとなる。また、制御プログラムは制御するノードの状態を管理する必要があるため、仮想ノード数に依存して使用するメモリ量が増加する。例えば、1 つの仮想ノードの制御に必要なメモリ量が 100kB であるとするなら、100 万の仮想ノードを制御するために必要なメモリ量は約 100GB となる。

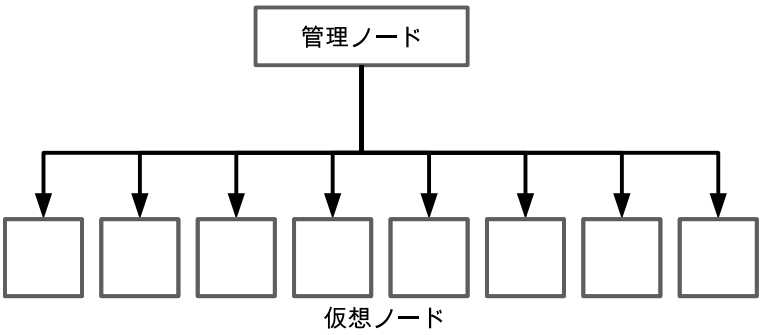

図 4.2: 単一管理ノードによる仮想ノードの管理

このような制限の中で大規模な実証実験を行うために、管理ノードが持つリソースを実験の規模に合わせて増加する方法と、制御を複数の管理ノードに分割する方法がある。前者の方法では、現状の管理ノードで制御しきれない規模の仮想ノードを制御するために、現場の管理ノードの持つリソースを増加しなければならない。しかし、計算機に取り付けられる装置の数には上限があり、上限に達した場合、既に取り付けられている装置と交換しなければならない。制御を複数の管理ノードに分割する方法は、制御が必要な仮想ノードの数に応じて複数の管理ノードを用意する事で実現する (図 4.3)。テストベッドで

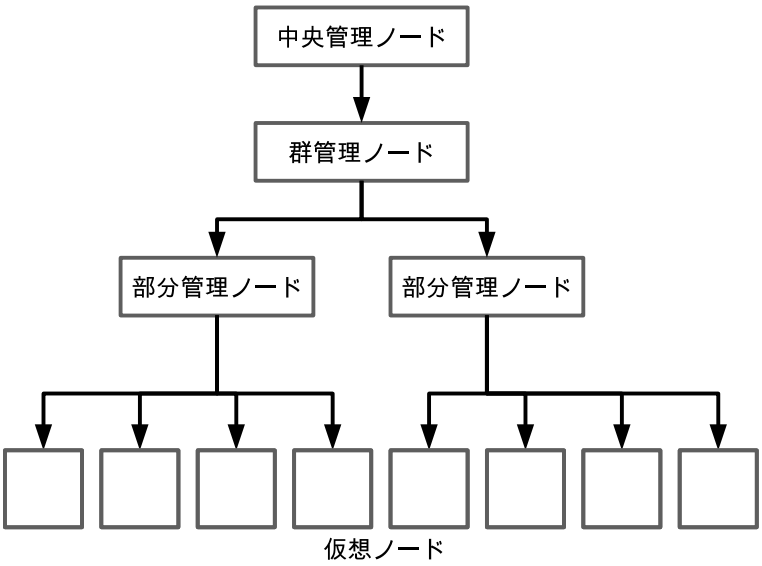

図 4.3: 複数管理ノードによる仮想ノードの分散管理

は、多数の物理マシンを実験のために用意してあるため、これらを必要に応じて管理ノードに割り当てる事が可能である。そのため、管理ノードの分割は、テストベッドでの実験に適している。

4.5.1 群としての仮想ノードの扱い

次に、仮想ノードの制御を分割する単位について考える。仮想ノードの制御が分散すると、各管理ノードに必要な管理コストは減少するが、実験環境全体の管理コストは相対的に増える。従って、スケーラビリティを確保するには、効率の良い管理の分散が必要である。

大規模なネットワーク模倣環境では、模倣を行うために各仮想ノードが通信を行う。仮想ノードは検証対象ノードと環境ノードに分類することが可能である。検証対象ノードは、検証対象となる技術が動作する対象のノードである。環境ノードは、検証対象ノードが動作するネットワーク上で通信を行い、ネットワークへの影響を及ぼすノードである。環境ノードは、検証対象ノードと異なり、個々の詳細な設定やきめ細やかな制御を要求しない。従って、これらの仮想ノードは一様な制御が可能である。本研究では、このように一様な制御が可能な仮想ノードの塊を、仮想ノード群と呼ぶ。仮想ノード群の概要を図

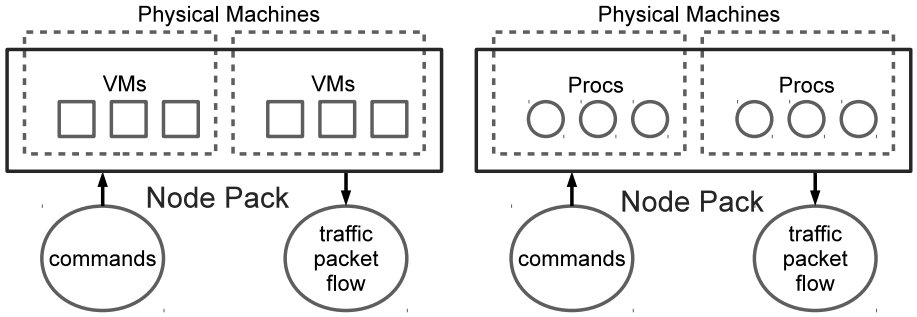

図 4.4: 仮想ノード群の概要

4.4 に示す。仮想ノード群は、個々の仮想ノードに対する個別の制御インタフェースを持たず、複数の仮想ノードに対する統合的な制御インタフェース (仮想ノード群制御インタフェース) を持つ。

図 4.5: 既存の制御インタフェースと仮想ノード群制御インタフェースの関係

次に、仮想ノード群制御インタフェースを多重化技術の観点から説明する。各多重化技術での既存の制御インタフェースと本研究で述べる仮想ノード群制御インタフェースの

関係を図 4.5 に示す。これまで、様々な多重化技術が開発されてきた。多重化技術には、KVM や Xen といったホスト型の多重化を行う多重化技術や、プロセスやスレッドといった非ホスト型の多重化技術がある。各多重化技術の生成物の制御インタフェースは多重化手法により異なる。異なるインタフェースを用いることで、複数の多重化手法を同時に利用したり、適用する多重化技術を別のものに置き換える際に、制御が煩雑になる。そこで、複数の多重化技術による生成物を統合的に扱うための統一的制御インタフェースが開発されている。この統一的制御インタフェースの例としては、libvirt が挙げられる。個々の多重化技術の制御インタフェースに対応する事に比べ、より少ない数のインタフェースへの対応で生成物の制御が可能になる。しかし、そのような統一的制御インタフェースは実証実験とは異なる各抽象化単位毎に存在する。そのため、異なる抽象化方式に属する多重化技術を同時に用いるためには、複数の統一的制御インタフェースを用いる必要がある。そこで、これらの統一的制御インタフェースを用いつつ、更にネットワーク実証実験における仮想ノードの制御を抽象化したインタフェースを用意することで、実験者に対して、汎用的な仮想ノードの制御を可能とする。

4.6 BlackSmith の構成

4.6.1 全体構成

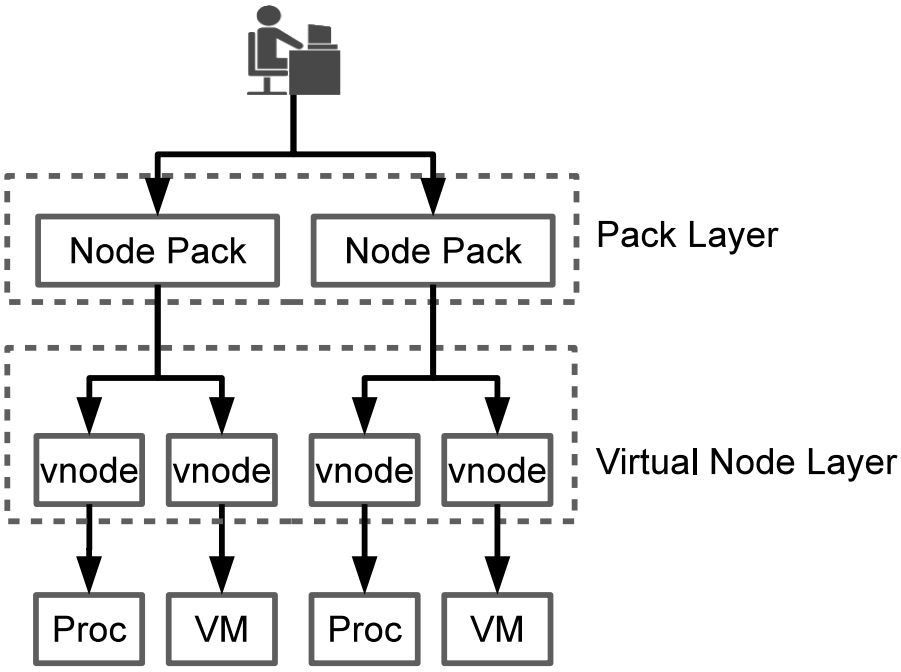

図 4.6: BlackSmith における階層構造

以上で述べたように、仮想ノードを群として制御するために、BlackSmith は 2 つのレイヤからなる階層構造を持つシステムとして設計した。BlackSmith の階層構造を図 4.6 に示す。この階層構造では、まず仮想ノードレイヤで多重化手法に対する抽象化を行う。次に、群レイヤで複数の仮想ノードに対する制御を抽象化する。最終的に、実験者に対して、多重化手法や制御するノード数に依存しない群としての制御インタフェースを提供する。

4.6.2 群レイヤ

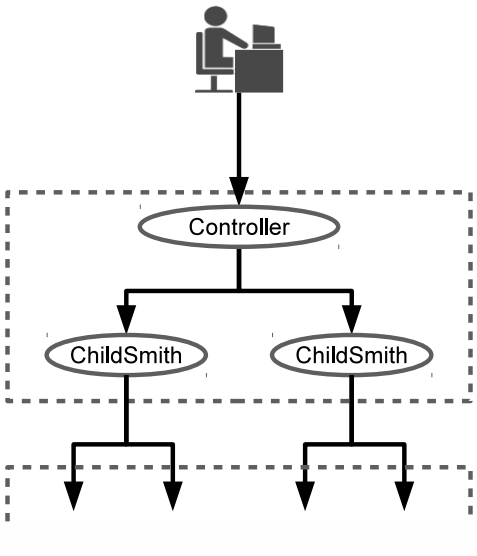

図 4.7: 群レイヤを構成するコンポーネント

群レイヤでは、複数の仮想ノードに対する制御を抽象化する。このレイヤを構成する各コンポーネントを図 4.7 に示す。このレイヤでは、群としてのノードの管理や、分散管理を実現するため、2 段階に分けて制御システムを配置する。Controller は、複数の仮想ノードに対する単一の制御インタフェースの提供を行う。複数の仮想ノードに対する制御を抽象化するためには、各仮想ノードが動作する物理マシンの情報や、各仮想ノードの状態を個別に管理する必要がある。そこで、Controller がこれらの情報を内部で保有することで、上位層に対して 1 つの群としての管理を可能とする。ChildSmith は、仮想ノードの分散管理を行う。各物理マシン内では、複数の仮想ノードが動作する。物理マシン内部での通信は、物理 NIC を経由しないため、ネットワークを経由する制御に対して、低コストで行うことが可能である。そこで、各物理マシン内に ChildSmith を配置し、物理マシン内の管理を ChildSmith に委譲することで、群の分散管理を行う。Controller が受けた管理命令を各物理マシンに配置した ChildSmith に対して依頼することで、群内のノードの分散した制御を行う。

4.6.3 仮想ノードレイヤ

このレイヤでは、多重化手法に依存しない仮想ノードの制御を実現するため、制御の抽象化を行う。このレイヤを構成する各コンポーネントを図 4.8 に示す。このレイヤでは、

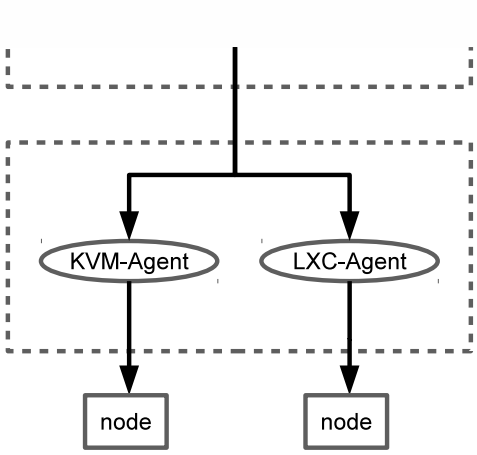

図 4.8: 仮想ノードレイヤを構成するコンポーネント

各多重化手法に対応した制御を行うため、それぞれの多重化手法に対応した Agent を配置する。Agent は、仮想ノード制御を多重化手法に関して抽象化する。各多重化手法毎に制御のインタフェースが異なる。そのため、上位レイヤからの命令に対して、各多重化手法に対応した制御命令へと投影する。

第5章　実装

本章では、仮想ノード群の制御についての考察を元に、BlackSmith の実装について述べる。

5.1　全体構成

本システムは既存の SpringOS の枠組み上での動作を目的としている。既存の枠組みへ変更を加えずに実験で本システムを利用するためには、実験毎に本システムを配布する必要がある。これを容易にするために、BlackSmith は各コンポーネントがインストールされたディスクイメージ群として実験者に提供する (図 5.1)。実験者は、物理マシンの場合と同様なディスクイメージの指定により、BlackSmith の環境を容易に構築する。BlackSmith の環境が構築されると、実験者は Kuroyuri を用いて BlackSmith へ制御命令を発行して、仮想ノード群の制御が可能になる。

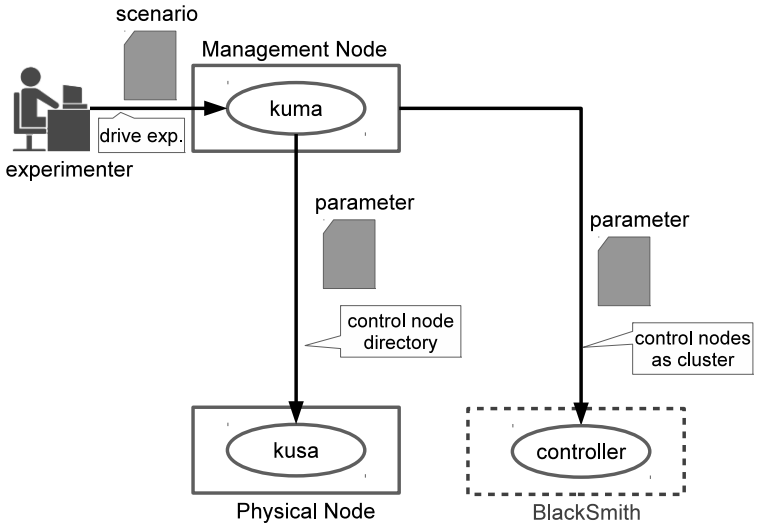

図 5.1: 既存システムへの BlackSmith の適用

本システムは、中央管理ノードとして SpringOS の Kuroyuri Master(kuma) を想定している。検証対象ノードは既存の SpringOS の枠組みにより管理する。環境ノードである仮想ノード群は BlackSmith により管理される。BlackSmith の全体構成を図 5.2 に示す。

BlackSmith は、仮想ノード群全体の制御を行う Controller と、各計算機上の仮想ノードを管理する ChildSmith、ChildSmith から制御命令を受け取り各多重化手法毎の制御命令に変換する Agent 群 (KVM-Agent、LXC-Agent) からなる。本システムの各要素は、前章で示した仮想ノード群の概要を示す図 4.3 の各要素に対応する。群管理ノードは Controller が配置された Controller node が担当する。部分管理ノードは ChildSmith が配置された各物理マシンが担当する。この構成により、仮想ノード群の管理を分散的に行う。

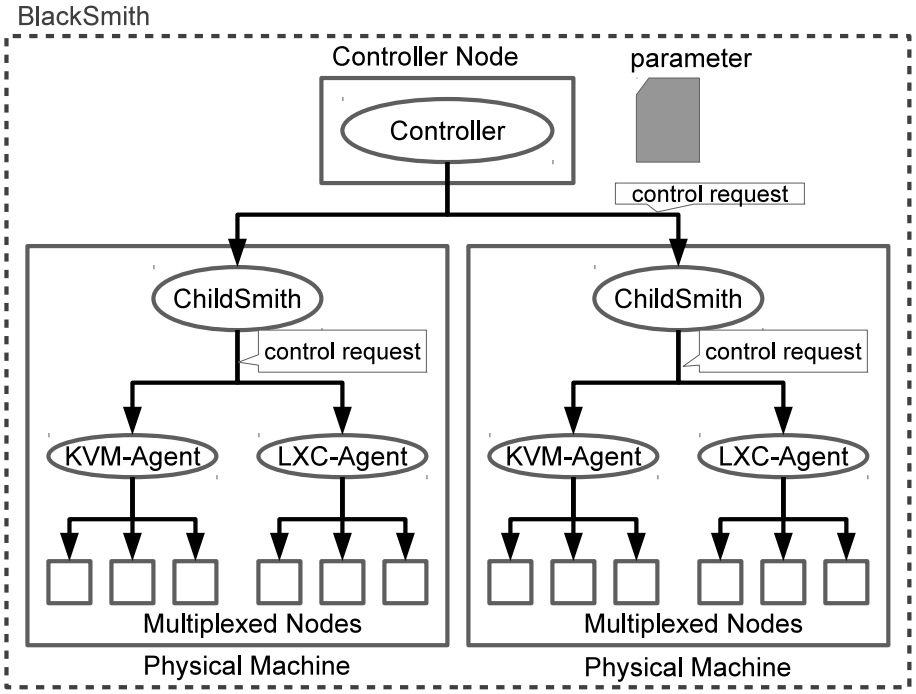

図 5.2: BlackSmith の全体構成

中央管理ノードから Controller 間の制御通信は、仮想ノード群に対する制御命令として、仮想ノードの数に関わらず行われる。これにより、仮想ノード群を構成する個別の仮想ノードへの制御通信を集約し、中央管理ノードに必要な管理コストを削減する。制御命令は、群管理ノードに配置した Controller が受け取る。Controller は実験者が指定した仮想ノードの仕様を受け取り、BlackSmith 配下の物理マシン上で動作している ChildSmith に多重化の命令を行う。個別の仮想ノードの管理を ChildSmith に任せて、Controller への管理負荷の集中を防ぐ。ChildSmith は、多重化技術毎に異なる具体的な制御命令の発行を、Agent に命令する。生成された仮想ノードを物理マシン内で管理するために、BlackSmith では、各物理マシン内に仮想ノード管理専用の内部ネットワークを持つ。Agent は、仮想ノードの制御命令を各多重化技術毎の制御命令に変換する。全てに共通のインタフェースを持たせ、BlackSmith は複数の実験粒度への対応が可能である。

BlackSmith では、実験者から直接仮想ノード要求を受け付けることで、実験者から見た実験のフローを次の様に単純化する。

- BlackSmith を用いて仮想ノードを生成
- BlackSmith を用いて仮想ノードの実行

5.2 Controller

Controller は、BlackSmith を用いた仮想ノードの管理を統率する。群を構成する全ての仮想ノードは、Controller が持つデータベースによって管理される。テストベッドが保有する物理ノードに関する情報はテストベッドが持つ静的なデータベースへ格納され、実験者が動的に生成する仮想ノードに関する情報は BlackSmith の持つ動的なデータベースへ格納される。これにより、仮想ノードに関する責任分界点を明確にする。

Controller は、仮想ノードの生成・破棄と制御、状態の管理を行う。生成・破棄のフローは次の様になる。

- 仮想ノードの生成要求を受ける
- 物理ノードに対するディスクイメージの配布を行う
- 仮想ノードに必要なファイルを物理マシンに転送する
- 各多重化手法毎に異なる仮想ノードの生成命令を物理ノードに発行する

Controller は、まず物理ノードに対して、必要なディスクイメージの配布を行い、ノードを起動する。ここで配布するディスクイメージは、多重化手法毎の、汎用的なイメージが使用される。仮想マシンによる多重化の場合、仮想化ソフトウェア及び仮想マシンを外部から操作するためのソフトウェアがインストール済みである OS を利用する。次に、各物理ノード上で仮想ノード用のデータを配布する。仮想マシンによる多重化の場合、仮想マシン用のディスクイメージや、生成する仮想マシンの構成ファイルを配布する。最後に、各物理マシン上で仮想ノードを生成する。仮想マシンによる多重化の場合は、libvrit 等の API を利用して仮想マシンを生成する。

制御のフローは次の用に行う。

- ノード制御要求を受け付ける
- 制御する仮想ノードと物理マシンの対応情報を取得する
- 物理マシンに対して、仮想ノードの制御を要求する

Controller は、まず中央管理ノードからノード制御要求を受ける。ノード制御要求は、制御対象となるノードのリストと、制御命令からなる。次に、BlackSmith が保有するデータベースから、制御する仮想ノードと物理ノードの対応情報を取得する。これにより、1 つの物理ノード上で動作する、複数の仮想ノードに対する同一の制御命令を、1 つの制御メッセージに集約することが可能となる。

情報の管理は次の様になる。

- 仮想ノード取得要求を受ける
- 仮想ノードと物理マシンの対応を決定する
- ノード生成を行う
- 生成した仮想ノードの情報をデータベースに保持する
- 必要に応じてデータベースから情報を取得する

Controller は物理マシンが必要になると、保有する物理マシンから利用する物理マシンを選択する。選択した物理マシンに対して、仮想ノードとの対応関係を決定する。この情報を元に、仮想ノードの生成を行う。仮想ノードと物理ノードの対応関係はデータベースとして保持し、仮想ノードの制御に利用する。

5.3 ChildSmith

ChildSmith は、Controller からの制御命令を受け取り、ChildSmith の動作する物理マシン配下の仮想ノードを制御する。仮想ノードへの制御を Controller が直接行う場合、仮想ノードに IP アドレスを割り当てなければならない。StarBED では、運用上管理用ネットワークの DHCP サーバの設定を変更することはできない。また、1 台の物理ノード上で動作している仮想ノードに対する同一の制御命令を、全ての仮想ノードに対して発行するため、効率的ではない。そこで、各物理マシンにおいて一旦命令を集約することにより、制御を効率的に行える。それと同時に、物理ノードの内部に制御系を生成することにより、施設の持つ DHCP サーバ等のソフトウェアに変更を加えること無く動的なリソースの制御が可能である。

本研究ではこれを実現するために、物理ノード上で制御命令を集約するための ChildSmith を設計した。ChildSmith は、Controller から受け取った制御命令を、対象の仮想ノードへ適用する。ChildSmith は仮想ノードが動作している物理ノードへ配置される。Controller から制御対象となる仮想ノードのリストと制御命令を受け取り、各仮想ノードの粒度に適した制御を行うために Agent に制御を依頼する。

ChildSmith の動作は、初期化フェイズ及び仮想ノード制御フェイズの 2 つに分かれる (図 5.3)。初期化フェイズでは、Controller から受け取った情報を元に、ChildSmith の情報を実験に合わせ初期化する。まず、Controller が指定する多重化技術に合わせた Agent を選択する。これにより、仮想ノードの多重化技術を実験環境に合わせた決定が可能である。次に、仮想ノードの制御に必要なファイル群を取得するためのファイルサーバを決定する。これにより、実験環境のファイルサーバの配置に依存しないファイルの配布が可能である。仮想ノード制御フェイズでは、ノードの構築や起動、除去を行う。初期化フェイズで決定した Agent プログラムに対して、それぞれの制御命令に対応したコマンド及びパラメータを送信する事で、仮想ノードの制御を行う。

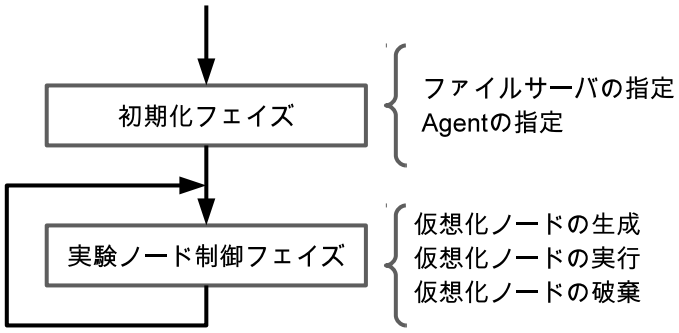

図 5.3: ChildSmith の動作フロー

5.4 Agent

Agent は、ChildSmith からの制御命令を受け取り、各多重化技術毎の制御手法に従った制御命令に変換し、実行する。様々な多重化技術への拡張を容易にするため、Agent は各多重化技術毎に独立した実装を行う。そのために、全ての Agent に共通のインタフェースを設け、同一の制御により仮想ノードの管理を可能とする。

各 Agent はノードの構築、起動、除去の制御命令を受け付けるインタフェースを持つ。また、これらの制御命令を実行する際には、仮想ノードの仕様を指示した制御パラメータを受け取る。Agent が制御命令を受け付けると、制御命令を各多重化手法毎の制御命令に変換・実行し、ノードの多重化を行う。ノードの多重化では、ディスクイメージなどのファイルコピーや、仮想ノード毎のパラメータの変更、制御用ネットワーク及び実験用ネットワークへの接続等を行う。

第6章　検証と考察

本章では、BlackSmith を用いた検証実験に対する規模拡張性を検証する。そのために、既存の手法と BlackSmith を用いた場合の仮想ノード管理について比較する。

6.1　検証方法

実際にシステムを用いた仮想ノード管理の可否に関する検証を行なった。検証では、複数の物理マシンを用いて本システムを構成し、群管理の機能について確認を行った。検証を行なうために、以下のような環境を容易した。物理リソースとして 2 台の物理マシンを用意した。仮想化手法として KVM を利用した。生成した仮想ノードは 20 である。管理コマンドとして、仮想ノードの生成と破棄及び実行を行った。これらのリソースを用いて、仮想ノードを生成・制御した。管理のために、単純な命令をコントローラに送信する中央管理コントローラを実装した。また、2 台の物理マシンそれぞれに ChildSmith 及び KVM-Agent を配置した。

6.2　検証結果

前節で述べた方法に基づき、複数の仮想ノードの管理の可否について検証を行った。その結果、2 台の物理マシンに対して仮想ノードの生成、実行、破棄を確認した。

表 6.1 に、この検証による結果と従来の仮想ノード管理の比較を示す。制御対象の仮想ノード数を N ノード、利用する多重化手法の数を M 種類とする。また、同様の制御を行う仮想ノード群の数を P 個とする。まず、発行する制御命令の数について比較する。従来手法では、個々のノードを直接制御するため、N ノードに対して N の制御命令が発行される。提案手法では、群に対して単一の制御命令を発行するため、M/P 個の制御命令が

表 6.1: 各粒度で専有するリソースの種類

制御項目＼手法	従来手法	提案手法
発行する制御命令数	N 個	N/P 個
制御命令の種類	M 種類	M/P 種類

表 6.2: 専有するリソースの種類とタイプの対応

タイプ＼専有リソース	CPU	MEM	FS	ADDR
A	✓	✓	✓	✓
B	—	—	✓	✓

FS: File System　ADDR: IP Address
MEM: Memory

発行される。従って、大規模な群が生成されるような環境においては、制御命令を大幅に減少させることが可能である。次に、利用する多重化手法の数と制御命令の種類について比較する。従来手法では、各多重化手法に関する制御命令を個別に記述する必要があるため、M 種類の多重化手法に対して M 種類の制御命令が必要である。提案手法では、各多重化手法に関する制御を隠蔽することによって、最大で M/P 種類の制御命令で実験が可能となる。従って、様々な多重化手法が利用されるような検証において、制御命令の種類を大幅に削減することが可能である。

6.3　専有するリソースを元にした実験シナリオの記述

BlackSmith によって、実験者は多様な多重化手法を同一のインタフェースで扱うことが可能となった。これまでの実験記述言語に対して専有するリソースについて記述を拡張することで、本システムと連携した動作が可能となると考えられる。例として、実験ノードを 100 万台用意し、指定したサーバに対して ping コマンドによる ICMP メッセージを送信するシナリオを考える。はじめに、実験者に対して表 6.2 に示すような実験ノードのタイプと専有可能なリソースの対応表を提示する。実験者は、専有が必要なリソースから実験ノードのタイプを選択し、ノードの要求を行う。BlackSmith を用いた環境で実験を

```
nodeclass samplenode{
  scenario{
    recv dst
    wake "/sbin/ping" "/sbin/ping" "-c" "10" dst
  }
}
nodeset nodes class samplenode num=1000000 \
 type=B os=ubuntu archi=i386
```

図 6.1: BlackSmith を用いた環境でのスクリプト記述

行う際の、kuma へ入力する実験スクリプトの一部を図 6.1 に示す。

この例では、samplenode という実験ノードの雛形を作成し、その実験ノードで実行したいシナリオを記述している。最後の行では、num=1000000 で 100 万台の samplenode の生成を指定している。type=B の記述では、ファイルシステムと CPU を他のノードと共有しても良いことを指定している。os=ubuntu、 archi=i386 は、それぞれ OS や CPU アーキテクチャの指定を行なっている。

利用者はこの例の様に、必要な実験ノードの数や粒度のみ記述すれば良く、実際にそれぞれの多重化手法でノードを生成するために必要な作業を意識する必要は無い。これによって、利用者からは透過的に実験ノードを利用することが可能となる。

6.4 リソース指向のノード多重化手法

前節では、専有するリソースから実験者に適した粒度で仮想ノードを生成する手法について述べた。これを拡張し、実験に必要なリソースの種類をより細かく分割することで、より小さな粒度でノードを多重化し、物理マシンをより効率的に利用できると考えられる。本章では、実験で利用される様々なリソースの種類を考え、最適にリソースを専有する実験ノードを実現する手法について考察する。

6.4.1 ファイル

アプリケーションの動作に必要なリソースとして、パスが固定された設定ファイルがある。複数のアプリケーションがそれぞれ別の設定による動作を行う場合は、実験ノード毎に異なるディレクトリツリーが必要である。ディレクトリツリーを変更するための方法としては、chroot がある。chroot を利用することで、1 つの物理ノード上で、同じ設定ファイルを共有するようなアプリケーションを多重化する事が可能となる。

6.4.2 IP アドレス

ネットワーク上のノードを表す代表的なパラメータとして、マシンに割り当てられた IP アドレスがある。そこで、IP アドレスを専有することができれば、IP ノードによる実証実験における実験ノードを表現することが可能である。

仮想マシンよりも多重化によるオーバーヘッドが少ない多重化方法として、ユーザ空間による多重化が挙げられる。ユーザ空間による多重化では、先に挙げた IP アドレスやポート番号を実験ノードが専有することができない。そこで、ユーザ空間による多重化ノードに、IP アドレスが専有できるような仕組みを提供することで、仮想マシンよりもオーバーヘッドの小さい多重化手法で粒度の小さい実験ノードが実現できると考えられる。

IP アドレスを専有するためには、実験ノード毎のルーティングが必要である。Unix 系 OS は、仮想ネットワークインタフェース（NIC）を利用して、複数の IP アドレスを保持

することが可能である。パケットを対象に送信する際には、宛先誘導型ルーティングを行うため、ルーティングテーブルを参照して利用する NIC が選択される。そのため、同じ対象の実験ノードに対する接続は、同じ IP アドレスを利用してパケットが送信されてしまう。実験ノードが IP アドレスを専有するためには、実験ノードが動作する物理ノード上でポリシルーティングを行う必要がある。物理ノードのようなエンドホスト上でポリシルーティングを行うソフトウェアとして、Rump[7] や PacketX[8] がある。PacketX には、Each process routing system[8] と呼ばれる、プロセス毎にルーティング制御を行う機能がある。また、Rump はネットワークスタックをユーザ空間で動作させる機構である。プログラム中に直接 IP 等の制御を記述する事で、プロセス毎に独自の制御を可能とする。また、既存のカーネルに組み込まれているプログラムを元にユーザ空間のプログラムを生成するため、カーネル空間のネットワークスタックの動作を模倣することが可能である。これらの機能を利用し、1 つのユーザ空間で実行されるプロセスに対し、1 つの経路を選択することで、個々のプロセスが IP アドレスを専有することが可能になる。これにより、IP ノードを模倣した実証実験をより低コストで行えると考えられる。

6.4.3 仮想メモリ空間

計算により単純なトラヒックを生成する場合、専有するリソースはプロセスが動作するためのメモリ空間だけで良い。仮想メモリ空間のみを専有するような実験ノードは、多重化のオーバーヘッドが非常に小さいため、より多くの実験ノードを実現することが可能である。しかし、リソースが非常に制限されるため、あまり複雑なノードを表現する事はできない。

6.4.4 ポート

サーバアプリケーションは、クライアントからの接続を待ち受けるためのポート番号が必要である。ポートは、1 つのアプリケーションに専有されると、他のアプリケーションで利用することができない。そのため、サーバに対して IP アドレスとポート番号のセットを渡すことで、1 つの物理ノード上で同じ番号のポートを利用するサーバを多重化することが可能である。

第7章　おわりに

本研究では、テストベッドにおいてより大規模なネットワークの検証を可能とする事を目的とした。そのために、これまで行われてきた大規模な検証実験について調査した。調査した結果、仮想ノードの管理をより効率的に行う必要があることが判明した。

仮想ノードの管理をより効率的に行うため、仮想ノード群制御システムを設計した。本システムでは、仮想ノードの管理を群という概念で抽象化・集約した。群とは、大規模な仮想ノードの管理を目的とした単位である。群により複数の仮想ノードに対する制御の効率化と、様々な仮想化技術への対応を行った。まず、様々な多重化技術を検証に柔軟に取り入れることで、実験リソースを効率的に扱い、また制御記述についてもより単純な命令で実験者が制御できるようにした。次に、複数の仮想ノードの制御を集約することで、管理システムへの負荷を低減し、また仮想ノードに対する複雑な制御を単純化した。

群としての制御を実際に行うために、仮想ノードを制御するためのシステム BlackSmith を実装した。BlackSmith では、コントローラ、ChildSmith、Agent の各コンポーネントからなる。各コンポーネントは独立して制御可能であり、予めインストールしたディスクイメージを用意することで、既存の実験支援システムを用いた展開が容易となった。

BlackSmith を用いて、仮想ノードを群として扱う検証を行った。検証結果、複数の物理マシンを利用した、複数仮想ノードの生成・制御・破棄等といった管理が可能であった。これにより、仮想ノードを利用した環境において、より効率的にリソースを管理し、より大規模な検証を可能とする事が可能となった。

謝辞

本研究を行うにあたり、多くの方々のご支援を頂きました。主指導教員の篠田陽一教授には、丁寧なご指導を頂きました。ここに深く感謝致します。

副指導教員の丹康雄教授、副テーマ指導教員の鈴木正人教授には、研究に関して多くのご意見を頂きました。ここに深く感謝致します。

本研究室の知念賢一特任准教授、宇多仁助教には、研究に関する示唆を頂きました。ここに深く感謝致します。

情報通信研究機構の三輪信介氏、宮地利幸氏、太田悟史氏、中井浩氏、高野祐輝氏、安田真吾氏には、テストベッドの運用に関する知見をご教授頂きました。ここに深く感謝致します。

奈良先端科学技術大学院大学の櫨山寛章助教には、テストベッドでの実験に関する知見をご教授頂きました。ここに深く感謝致します。

本研究室研究員の井上朋哉博士、本研究室博士後期課程の明石邦夫氏には、研究の方針について多くのご意見を頂きました。ここに深く感謝致します。

本研究室修了生の LATT Khin Thida 氏、松井大輔氏、川瀬拓哉氏、立花一樹氏、吉岡慎一郎氏、橋本将彦氏、田部英樹氏、村上正太郎氏には、研究に関する議論とご意見を頂きました。ここに深く感謝致します。本研究室博士後期課程の Muhammad Imran Tariq 氏、博士前期課程の大野夏希氏、向井雄一朗氏、岩橋紘司氏、加藤邦章氏、成田佳介氏、向井康貴氏、岩本裕真氏には、研究に関する活発な議論をして頂きました。ここに深く感謝致します。

最後に、これまで支えて頂いた私の家族に、心から感謝致します。

参考文献

[1] Toshiyuki Miyachi, Yoshiki Makino, Razvan Beuran, Satoshi Uda, Shinsuke Miwa, and Yasuo Tan. Fault injection on a large-scale network testbed. In *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, pp. 4–11, New York, NY, USA, 2011. ACM.

[2] 宮地利幸, 三輪信介, 知念賢一, 篠田陽一. ネットワーク実験支援ソフトウェアの汎用アーキテクチャの提案. 情報処理学会論文誌, Vol. 48, No. 4, pp. 1695–1709, 2007.

[3] Toshiyuki Miyachi, Kenichi Chinen, and Yoichi Shinoda. Starbed and springos: large-scale general purpose network testbed and supporting software. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, valuetools '06, pp. 1–7, New York, NY, USA, 2006. ACM.

[4] 宮地利幸, 知念賢一, 篠田陽一. 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 (ネットワーク). 情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム], Vol. 48, pp. 105–112, 2005.

[5] S. Miwa, M Suzuki, H Hazeyama, S Uda, T. Miyachi, Y Kadobayashi, and Y. Shinoda. Experiences in emulating 10k as topology with massive vm multiplexing. *In Proceedings of The First ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures(VISA 2009)*, 2009.

[6] 吉岡慎一郎. 大規模ネットワーク実証実験設備における実験用資源の最適利用に関する研究. Master's thesis, 北陸先端科学技術大学院大学情報科学研究科, 2011.

[7] Antti Kantee. *Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels*. Aalto University publication series DOCTORAL DISSERTATIONS, 171/2012. 2012.

[8] 井上朋哉. エンドホストポリシ指向 IP パケット制御機構の設計と実装に関する研究. Master's thesis, 北陸先端科学技術大学院大学情報科学研究科, 2006.

本研究に関する論文

- 鍛治祐希, 安田真悟, 知念賢一, 篠田陽一. SpringOS M-Extension: 超大規模実証実験環境を想定したリソース指向実験ノード管理手法. DICOMO 2012, Vol. 48, pp. 1513—9519, 2012.
- 鍛治祐希, 井上朋哉, 知念賢一, 篠田陽一. Blacksmith: 実験環境における仮想ノード 群制御システム. IEICE-IN2013-36, Vol. IEICE-113, pp. 1—6, 2013.