WIDE クラウド WG 2012 年度活動報告

浅井 大史[*]　上野 幸杜[†]　岡本 慶大[‡]　島 慶一[§]　関谷 勇司[*]　中村 遼[*]
Nam Dang[¶]

2012 年 12 月 30 日

1 はじめに

WIDE クラウドワーキンググループは、今後のクラウド技術の研究開発を推進するために 2010 年 1 月に設立された。複数の WIDE 組織間に渡って運用される広域連邦型クラウドシステムである WIDE クラウドシステムの運用と、それを用いた研究開発を行っている。

2012 年度の主な活動を以下に列挙する。

- クラウドデプロイメント技術 Crowbar (第 2 章)
- クラウド管理技術 VMM-MIB (第 3 章)
- 甲子園中継 (第 4 章)
- 広域 L2 延伸技術 VXLAN (第 5 章)
- 経路制御技術 LISP (第 6、7 章)
- 分散ストレージ技術検証 (第 8 章)

以降、それぞれの活動の詳細を各章にて報告する。

2 Crowbar

本節では、プライベートクラウドの構築と管理を自動化する手法に関する研究成果について報告する。具体的には、DELL 社が主導で開発しているオープンソースソフトウェアである、Crowbar [1] を用いて、WIDE クラウドのハイパーバイザー管理を自動化することを目的としたシステム構築を行う。

以下の研究計画に基づき、本研究を行う予定であり、現在第 2 タームの StarBED における大規模展開実験を行なっている。

- 第 1 ターム (2012/04 - 2012/07)
  - Crowbar に関する情報収集と調査
    - システムの理解
    - テスト環境の構築
  - WIDE クラウドへの Crowbar 適用実験
    - 実環境としての WIDE クラウドへの導入開始
- 第 2 ターム (2012/08 - 2012/11)
  - Crowbar を利用したハイパーバイザー管理手法の研究開発
    - 新規 deploy や日常の update への適応手法の検討
    - WIDE クラウドへのさらなる導入と展開
  - Crowbar を用いた大規模環境構築実験
    - StarBED を用いた大規模展開実験
- 第 3 ターム (2012/12 - 2013/03)
  - 実験成果のまとめとドキュメント作成
    - Crowbar 導入のための日本語の情報提供
    - Crowbar を用いたプライベートクラウド構築のためのホワイトペーパー
  - WIDE における Crowbar 利用紹介
    - システム論文
    - 解説記事

以下に、第 1 タームの進捗とその成果に関して報告する。

[*]東京大学
[†]慶應義塾大学
[‡]奈良先端科学技術大学院大学
[§]IIJ イノベーションインスティテュート
[¶]東京工業大学

[1] http://ja.community.dell.com/techcenter/w/wiki/185.crowbar.aspx

2.1 システムの理解

Crowbar の構造に関する勉強会を 3 回開催し、Crowbar の構成と仕組みを理解した。その結果、核となるのは Chef [2] であり、Chef における Recipe や Role の定義を行うための作法を習得する必要があるということを理解した。この Chef に関しては、Crowbar に比べれば日本語の情報がインターネット上に存在しているが、やはり一般的な日本のユーザが問題なくインストールし、設定運用を行えるレベルには情報が整備されていない。Recipe や Role の定義、Crowbar との連携による Barclamp の定義と、それらの関連性を理解することが必要となるため、これらを包括的に解説した文章と概念図が必要と考える。また、Crowbar が想定するネットワーク環境も、DELL Crowbar Deployment Guide [3] に明記されているものの、全てのユーザがこの環境をそのまま用意できるわけではない。自身の環境にあわせてどう設定変更すれば良よいか、またそれぞれのネットワークがどのように利用されるのかが明記されておらず、この文書だけでは管理者がネットワーク設計を行うことが困難である。ネットワーク構成に関しては、DELL 社の crowbar wiki サイトにある図 1 が、最も良くネットワーク構成とその役割を明記しており、視覚的にわかりやすくまとめられている。しかし、この図においてもそれぞれのネットワークの役割が明記されておらず、自身のネットワーク環境にあわせて Crowbar を構成したいユーザや、OpenStack 以外のデプロイメントに利用したいユーザにとって情報が不足している。

2.2 テスト環境の構築

第 1 タームの目標である、WIDE クラウドへの導入を行うため、テスト環境の構築を行った。まず動作検証を行うために、Build Crowbar.iso [4] にある情報に従い、Crowbar admin node インストール用の ISO イメージを構築することに取り組んだ。しかし、文章にある通りに Ubuntu 12.04 にて構築を試みたものの、正常に ISO イメージが構築されないことを確認した。原因は、livecd-creator というツールが存在しないため、最終段階において Sledgehummer イメージを作成できないためである。`livecd-creator` は CentOS にて提供されていたツールであるため、さらに CentOS 6.3 を用いて ISO イメージの構築を試みた。しかし、結果は同様に `livecd-creator` がパッケージに存在せず、ISO イメージの構築に失敗した。CentOS 6.2 では `livecd-creator` が `livecd-tools` というパッケージによって提供されていたが、CentOS 6.3 になってから提供されなくなったためであると考えられる。CentOS 6.2 は既に各地の ftp サイトから削除されており、既に CentOS 6.2 を新規にインストールすることが困難となっている。つまり、以前作成した Sledgehummer イメージを所有しているユーザは、その Sledgehummer イメージを使いまわすことで Admin node インストール用 ISO イメージを構築できるが、新規ユーザは Sledgehummer イメージを作成することが困難である。これが最新の Crowbar OpenStack 1.4 git tree で修正されているかは未確認であるが、少なくとも Web にある文章に従っただけでは構築できないことが確認された。す。

なお、検証においては、作成済みの ISO イメージ [5] から Sledgehummer イメージを取り出すことによって、Admin node インストール ISO イメージを構築することに成功した。検証のために構築した環境を、図 1 に示す。この環境において Admin node インストール用の ISO イメージを構築し、Admin node のインストールに成功した。一点、注意が必要な点として、Admin node は十分なメモリ容量が必要ということがあげられる。初期のインストール時には、Admin node 用 VM のメモリ容量を 1Gbytes しか確保していなかったため、Swap 領域へのページアウトが発生し、正しくインストールを完了することができなかった。Admin node 用として用いる VM は少なくとも 2Gbytes、可能なら 4Gbytes 以上のメモリを確保しておくべきである。正常にインストールが完了すれば、Crowbar、Chef、ganglia、nagios3 が起動され、それぞれのインタフェースにアクセスすることができた。それぞれの画面にアクセスした様子を図 2 に示す。

[2] `http://wiki.opscode.com/display/chef/Home`

[3] `https://github.com/dellcloudedge/barclamp-openstack/raw/master/crowbar\_framework/public/Opentstack\_Deployment\_Guide\_V1.3.pdf`

[4] `https://github.com/dellcloudedge/crowbar/wiki/Build-Crowbar.ISO`

[5] `http://crowbar.zehicle.com/`

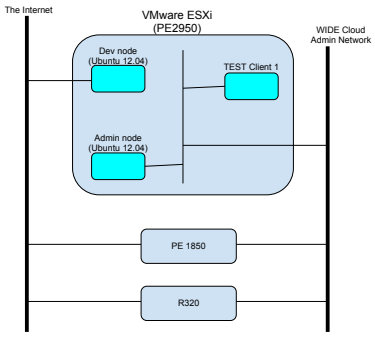


図 1: Crowbar 実証実験環境概要図

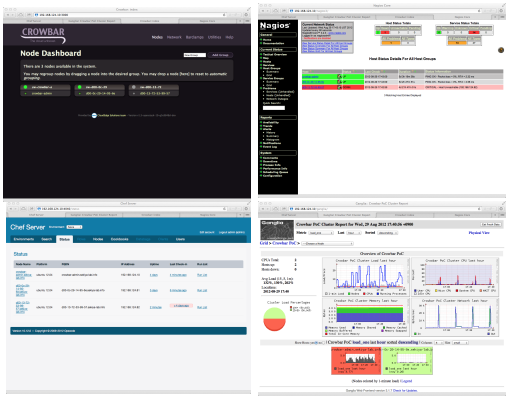


図 2: Crowbar 起動後画面例

2.3 WIDE クラウドへの導入試験

動作検証のために構築した Crowbar 環境を用いて、WIDE クラウドの HyperVisor を高速にデプロイするための導入実験を開始した。まず、kvm と libvirtd が自動的に導入されるような Barclamp の作成に取り組んだ。

以下に、KVM が起動する barclamp をつくるまでの手順を示す。

1. Barclamps ディレクトリの確認

   `/opt/dell/barclamps` に全ての Barclamp が設置される。1 つの Barclamp ディレクトリは Chef の CookBook と、その Barclamp に関する設定ファイルが設置されている。

2. 新規 Barclamp ディレクトリの作成

   `/opt/dell/bin/barclamp_create.rb` コマンドによって、Chef の基本的な設定を含んだ新規 Barclamp ディレクトリを作成する。ここでは新しい Barclamp を *TESTCLAMP* とする。

3. Chef の設定

   Barclamp では、Chef の CookBook を使って具体的な動作が決定される。Chef には、サーバの役割を定義する Role と、実際に何をさせるかを記述する Recipe があり、Role と Recipe の集合が CookBook となる。Chef は、この CookBook を作成し、ノードに Role を適用させていくことで、複数のノードを簡単にデプロイすることを可能としている。ここでは、試験的に TESTCLAMP 内の Chef に KVM を起動させる Role と Recipe を設定する。

   (a) Recipe の設定

   barclamp : TESTCLAMP の Recipe の設定は、`/opt/dell/barclamps/TESTCLAMP/chef/cookbooks/TESTCLAMP/Recipes` ディレクトリにある。とりあえず KVM を動かすには、`Recipes` ディレクトリに図 3 の内容の `default.rb` を作成する。

   これは、qemu-kvm、libvirt-bin パッケージが無ければインストールを行い、libvit-bin があれば起動する、という簡単な Recipe である。

   (b) Role への Recipe の割り当て

   Recipe を作成したら、Role に対して Recipe を割り当てる。Role のファイルは、`/opt/dell/barclamps/TESTCLAMP/chef/roles/TESTCLAMP-server.rb` (barclamp 作成時に用意される) を利用する。このファイルの中で、role “TESTCLAMP-server” に対して、recipe “default” を適用する。

```
package ''qemu-kvm'' do
 action :install
end

package ''libvirt-bin'' do
 action :install
end

service ''libvirt-bin'' do
 case node[:platform]
 when ''ubuntu''
    service_name ''libvirt-bin''
    restart_command ''serivce libvirt-bin start''
    reload_command ''service libvirt-bin reload''
    action :enable
 end
end
```

図 3: `default.rb` の内容

名前、ディスクリプションに加えて、`run_list` で複数の Recipe を Role に適用することができる。

4. 作成した Barclamp のインストール

   上記までで作成した Barclamp： TESTCLAMP を、Crowbar へインストールする。インストールは下記のコマンドで行う。

   ```
   /opt/dell/bin/barclamp_install.rb
   /opt/dell/barclamps/TESTCLAMP
   ```

   このコマンドから Barclamp : TESTCLAMP が Crowbar から見えるようになる。本来は、Crowbar の Barclamp List から TESTCLAMP を指定して Proposal を作成し、ノードに割り当てるはずだが、ここまでの設定では Proposal を作成できなかった。どうやら TESTCLAMP において、Crowbar 的に必須である設定が不足していると考えられる。この点に関しては、さらなる調査が必要となるため、今後解決する。

5. Chef からノードに Role を適用

   Crowbar から Role を適用することができないため、Chef の Web インターフェースから Role を適用した。Chef の Role リストから先ほど作成した TESTCLAMP-server の Role を探し、ノードに割り当てる。そして、そのノードを再起動すると、boot 後に Chef のクライアントが実行され、上記の Role TESTCLAMP-server にひもづけられた default recipe が実行される。これによって、そのノードで KVM が動作することを確認した。

   上記導入試験はまだ Barclamp 定義が未完成であり、Crowbar からの適用ができていない。今後さらなる調査を行い、まずは WIDE クラウドの一般的な Hyper Visor 定義を実現することを目指す。

2.4 成果の公開に関して

現在、第 1 タームがほぼ完了し、第 2 タームの実証実験を開始している。この段階においても、様々な問題に直面し、その度に文章や事例といった情報量の不足を実感してきた。そのため、現在までの経験を日本

語の文章において公開することも、日本の Crowbar ユーザにとって大きな貢献になると考える。また、日本においては Crowbar や Chef 自体の知名度も高くないため、このようなツールによって OpenStack や Cloudera が簡単に導入できるということからアピールすることが必要と考える。そのため、現時点での経験に基づく情報を、なんらかの媒体を通じて公開することが有意義であると考える。具体的には、次のような媒体が考えられる。

- CMS や Wiki を利用した、Crowbar 情報サイトの立ち上げ
- WIDE クラウドへの Crowbar 導入ホワイトペーパー

2.5 今後の予定

第 2 ターム、第 3 タームの予定に従い、共同研究を遂行していく予定である。現在のところ、大きな問題点は発生していないが、英文ページにて情報が古い部分、間違っている部分、足りない部分のフィードバックを、随時 Crowbar ML にて行なっていきたいと考えている。また、現在 Crowbar 2.0 のデザインが議論されている段階であり、現在の Crowbar 利用の経験を通じた Crowbar 2.0 へのフィードバックも行なっていければと考える。また、NICT 北陸 StarBED 技術センター (`http://starbed.nict.go.jp/`) での Crowbar 導入実験も行いたいと考える。StarBED では、多くのノードがベアメタルとしてユーザに提供され、ユーザは数百台単位のベアメタルサーバや、その上で動く 1000 台規模の VM に対して自由に OS をインストールし、実験環境を構築している。この際、現在は SpringOS [6] という、NICT が独自に作成したツールを用いて OS のインストール作業を行なっているが、ここに Crowbar を利用した OS 展開システムを導入することで、よりユーザにとって視覚的に使いやすい実験環境構築システムを提供できる可能性があると考える。この点に関しては、北陸 StarBED 技術センターの研究員とも話し合いを行なっており、まずは導入実験を行うことを計画している。共同研究の当初の目的である、WIDE クラウドへの展開を通じたプライベートクラウド展開へのモデル作成と事例作成に関しては、Crowbar 自体の改造を行わずとも、Barclamp を多種作成することによって実現できる見通しが立った。しかし、日本のユーザの規模にあわせた、中小規模企業ユーザや個人ユーザにとって使いやすくなるような改造や、日々の OS アップデートといった運用管理により役立つ機能が必要であると考える。そのため、Crowbar 自体の改良を行うことも検討したい。

3 仮想計算機モニター MIB

サーバ仮想化技術の発展により、VMware[1] vSphere Hypervisor (ESXi)[7]や Xen[2]、KVM[3] などのハイパーバイザーソフトウェアを用いることで、1 台の物理マシン上で複数の仮想マシンを稼働させることが可能となった。これにより、従来は多数の物理マシン上で稼働していたシステムを仮想マシンとして少数の物理マシンに集約することができ、計算機資源の効率的な利用が実現できる。また、計算機資源の抽象化により、仮想マシンへの計算機資源の動的割り当てや、ライブマイグレーション技術による仮想マシンの無停止でのハードウェア移行などの柔軟な運用が実現でき、仮想マシン上で稼働するサービスの可用性・保守性・拡張性が向上している。しかし、物理計算機資源を仮想マシンに割り当てるため、従来のホスト単位での計算機資源管理手法を用いることができない。例えば、ハイパーバイザー上で従来のホスト単位での計算機資源管理手法を用いた場合、総合的な物理計算機資源は管理できるが、各仮想マシンに割り当てられた詳細な計算機資源は管理できない。また、ハイパーバイザー上の仮想マシンで従来のホスト単位での計算機資源管理手法を用いた場合、物理計算機資源の管理はハイパーバイザーにより行われているため、仮想マシンからは割り当てられた仮想的な計算機資源の情報しか扱うことができない。一方、ハイパーバイザーにおける物理計算機資源の管理および仮想マシンへの割り当て管理はハイパーバイザーの運用において重要であるため、一般に各ハイパーバイザーソフトウェアによりこれらの情報を管理する API が提供されている。しかし、各ハイパーバイザーソフトウェア間で API が共通化されていないため、管理ソフトウェアもハイパーバイザー

[6] `http://www.starbed.org/documents/index-j.html`

[7] `http://www.vmware.com/products/vsphere-hypervisor/overview.html`

ソフトウェアに大きく依存してしまっており、現状では計算機資源管理が容易ではない。そこで、ハイパーバイザーおよび仮想マシンの計算機資源管理を簡便に行える方法が必要とされている。

我々は、従来からネットワーク機器における資源管理に広く用いられている SNMP (Simple Network Management Protocol) を用いたハイパーバイザー上の計算機資源および仮想マシンに割り当てられた計算機資源の管理を実現するために、これらの資源に対する MIB (Management Information Base) オブジェクトを定義し、“Management Information Base for the Virtual Machine Monitoring” (`draft-asai-vmm-mib`) として IETF (Internet Engineering Task Force) に Internet-Draft を提出した [4]。具体的な MIB オブジェクト一覧は省略するが、この Internet-Draft ではハイパーバイザーの運用に必要な計算機資源情報として、以下のオブジェクトを定義している。

- ハイパーバイザーに関するオブジェクト
  - ハイパーバイザーソフトウェアの情報
  - 仮想マシン一覧
- 各仮想マシンに関するオブジェクト
  - 仮想マシンのメタ情報・状態
  - 仮想 CPU（CPU 時間・アフィニティ）
  - 仮想メモリ（割り当て量）
  - 仮想ディスク（割り当て量・利用量）
  - 仮想ネットワーク（IF-MIB で定義されるオブジェクト）

また、我々は、提出した Internet-Draft で定義した MIB オブジェクトに対してアクセスする SNMP エージェントである *virtsnmp* [8]を実装した。なお、実装は 2012 年 12 月現在の最新バージョンである `draft-asai-vmm-mib-01` ではなく、前バージョンの `draft-asai-vmm-mib-00` に基づいている。図 4 に *virtsnmp* の実装の概要図を示す。*virtsnmp* は Net-SNMP[9]のサブエージェントとして実装されており、libvirt[10] API を通じてハイパーバイザーソフトウェアにより管理されている仮想マシンおよび仮想マシンに割り当てられた計算機資源にアクセスする。物理計算機資源については、ハイパーバイザー上でシステムコールを発行することでアクセスするようになっている。本実装の利点は、libvirt API を用いたことで KVM や Xen、VirtualBox など、幅広いハイパーバイザーソフトウェアを同時にサポートしていることである。しかし、ハイパーバイザーソフトウェア（特に KVM および Xen 以外のソフトウェア）によっては libvirt API からアクセスできないオブジェクトが多数あるため、現状では KVM および Xen で取得できるオブジェクトに着目して、実装の改善および Internet-Draft 提案への反映を行っている。

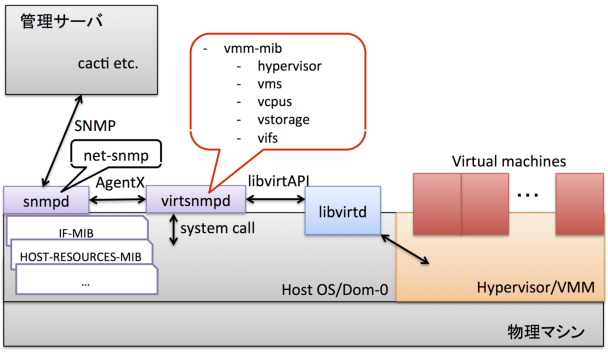


図 4: virtsnmp の実装

現在は同様の MIB オブジェクトを提案している [5] の著者である Schoenw らと連携して共同提案を策定中である。

4 甲子園インターネット中継

全国高校野球選手権大会 (夏の甲子園) のインターネット中継は、1998 年から奈良先端大と朝日放送、NTT スマートコネクトとの共同プロジェクトとして実施されている。大規模な実トラフィックを利用した Web 配信システムの実験が可能となっている。

2010 年度、2011 年度に引き続き、WIDE クラウドを利用した配信実験を行った。2012 年度は VM 数でクラウドの配信リソースを調整するシステムを構築し、実験を行った。

[8] `http://hg.scyphus.co.jp/virtsnmp/`
[9] `http://www.net-snmp.org/`
[10] `http://libvirt.org/`

4.1 甲子園システムの概要

甲子園システムは、Web コンテンツ全体を提供するサーバ群と、TV 放送映像をパラパラアニメ状の画像にして配信する Sentry システムから成る。Sentry システムの詳細とメカニズムについては、WIDE クラウドワーキンググループの 2011 年度報告書、および [6] を参照のこと。

2011 年度の実験では、WIDE クラウド内部セグメントに置かれた配信用 VM のサービスアドレスを東阪の map646 サーバから広告し、コンテンツを配信しながらのライブマイグレーションを可能とした。しかし、VM へのトラフィック振り分けの単位が VM 1 台の配信キャパシティを超える場合はサービスを継続できない、配信する VM の数だけグローバル IP アドレスが必要になるといったの問題があった。そこで、2012 年度はリバースプロキシを利用してこれらの問題を解決しようと試みた。

4.2 実験トポロジ

2012 年度は 2 種類の配信ネットワーク構成を利用した。それぞれのネットワーク概要図を図 5 および図 6 に示す。

パターン 1 では、リバースプロキシの nginx[7] VM と、画像を配信する sentryc VM の両方を WIDE クラウド内に構築し、インターネットとの接続は東阪に設置済の map646 サーバを利用した。甲子園 Web サイトユーザからのアクセスは東阪の map646 サーバのいずれかを通じて WIDE クラウドの内部に入り、nginx VM へ届く。nginx VM はさらにリクエストをバックエンドに配置した sentryc VM へ送る。sentryc からユーザまではこの逆の経路を辿る。この方式では 2011 年度の構成と同様、東阪での配信冗長性を確保したまま VM 数で配信能力を調整できるというメリットがあるが、nginx VM へのリソース割当が少ない場合にボトルネックになるというデメリットもある。これは、IaaS クラウドのユーザ側でリバースプロキシを構築するような環境を想定している。

パターン 2 では、リバースプロキシを堂島 DC に設置した物理サーバ上に構築した。これにより、パターン 1 における nginx VM のボトルネックを解消できるが、物理サーバが sentryc VM とインターネットの接続における単一障害点となる。これは、IaaS クラウドのプロバイダ側でリバースプロキシを提供しているような環境を想定している。

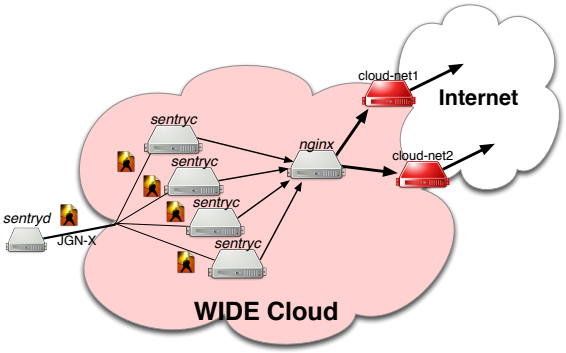


図 5: 2012 年度甲子園システムネットワーク概要図 パターン 1

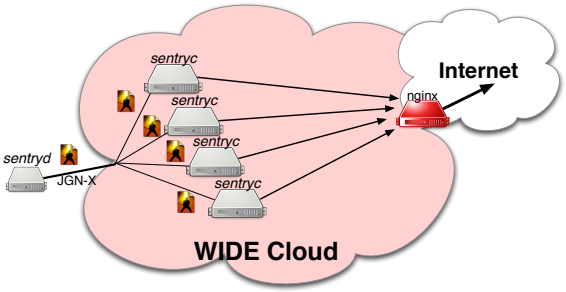


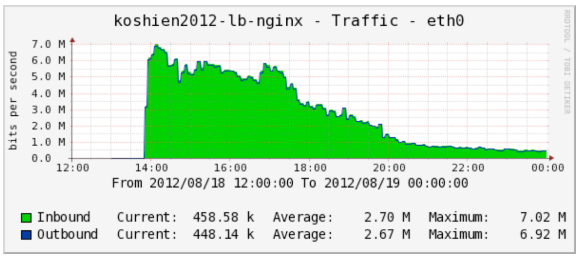
図 6: 2012 年度甲子園システムネットワーク概要図 パターン 2

画像コンテンツ転送に関しては昨年度同様、JGN-X を利用して WIDE クラウドの内部セグメント (VLAN 6) と ABC の内部セグメントを接続し、sentryd サーバから sentryc VM へコンテンツ転送可能とした。また、従来から運用しているオンプレミスのシステムと WIDE クラウドのシステムでの配信振り分けは、こちらも昨年度同様、JavaScript を利用したクライアントサイドサーバ選択を利用している。詳細については [6] を参照のこと。

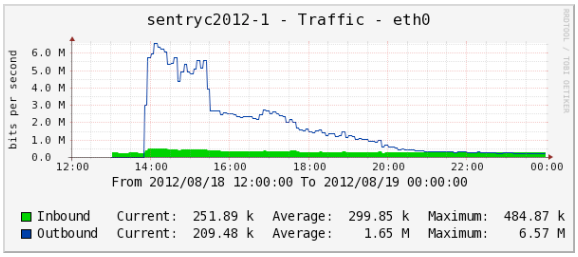
4.3 運用報告

本システムの実戦投入は選手権後半の 8 月 18 日から行った。まず始めに、パターン 1 のネットワーク構成において、1 台の nginx VM を経由して 1 台の sentryc

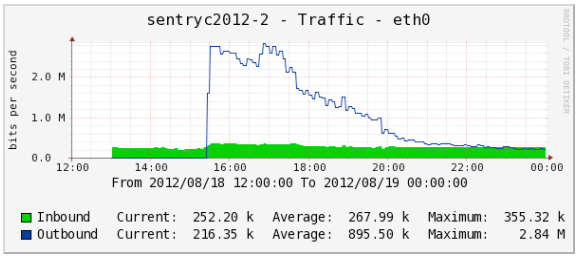
VM (VM1, sentryc2012-1) へプロキシする構成とした。また、nginx は 2 台目の sentryc VM が起動した際に自動的にトラフィックを分散するよう設定した。各 VM へのリソース割当は、sentryc VM へ VCPU 1、VMEM 2GB、nginx VM へ VCPU4、VMEM 2GB とした。運用開始日のトラフィックを図 7 に示す。12:50 頃に nginx VM へのトラフィック振り分けを始め、15:30 頃に 2 台目の VM (VM2, sentryc2012-2) の apache httpd を起動した。図 (c) の 15:30 頃、VM2 のトラフィックグラフが急激に上昇すると同時に、図 (b) において VM1 のトラフィックが半減している。これより、nginx の設定を変更すること無く、VM を起動することによって配信リソースを調整可能であることが実証された。nginx は、VM2 の httpd 起動後数秒以内にトラフィックの分散を始めるなど、優秀な死活監視機能を搭載していることも分かった。



(a) nginx VM



(b) VM1



(c) VM2

図 7: パターン 1 構成でのトラフィック

しかし翌 8 月 19 日のトラフィック集中時に一部クライアントへの配信遅延が発生した。図 8 にその際のトラフィックを示す。この原因を究明したところ、map646 のリソース枯渇によるものと分かった。その後物理サーバを利用した map646 への切り替え作業なども行ったが、設定ミスなどにより十分なパフォーマンスを出すことができなかった。また、sentryc の VM 数に関しては事前検証を行ったが、3 台以上では nginx VM がボトルネックとなってしまうことが分かったため 2 台までしか利用しなかった。

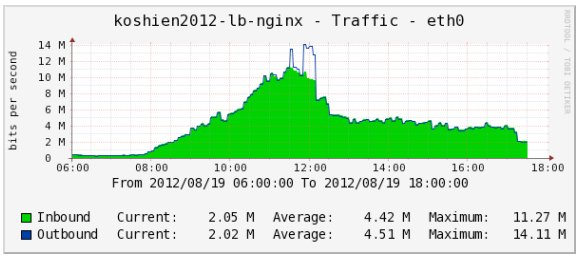


図 8: map646 トラブル発生時のトラフィック

8 月 23 日の決勝戦ではパターン 2 の構成にて配信を行った。1 台の nginx 物理サーバを経由して 4 台の sentryc VM へプロキシする構成とした。決勝戦でのトラフィックを図 9 に示す。配信した試合自体も注目のカードとなったこともあり、5 分間の平均トラフィックで過去最大となる 350Mbps を記録した。オンプレミスのシステムでも 800Mbps を記録し、合計トラフィックは 1.15Gbps となり、こちらも過去最高を記録している。

4.4 考察と今後の課題

2012 年度の配信実験ではリバースプロキシを利用し、配信リソースの柔軟な調整やグローバル IP アドレスの節約を試みた。VM 数による配信リソーススケーリングはうまく動作し、また実験中に同時に利用したグローバル IPv4 アドレスはリバースプロキシ用の 1 つのみである。最終的には過去最大の配信トラフィックを受け持つことができた。しかし、今回構築したシステム以外の部分によるシステムトラブルが多発し、配信中のマイグレーションなどに関しては十分な実験が行えなかった。配信に直接関係する部分だけではなく、基盤部分のベンチマークや調整、チューニングも含めて事前に行う必要がある。

nginx は、パフォーマンス面、設定面、死活監視面のどれを取ってもソフトウェアベースのロードバランサとしては非常に優秀であることが分かった。今回は VM と物理サーバ両方での実験を行ったが、VM では割り当てたリソースに対して十分なパフォーマンスが発揮できないため、物理サーバやアプライアンスを配置し、IaaS プロバイダの 1 つのサービスとして運用する形が望ましいと思われる。今後はこれらリバースプロキシの強化や冗長化なども含めた、より強固な配信基盤の構築に取り組みたいと考えている。

4.5 付録

2012 年度の実験で利用した nginx の設定 (抜粋) を図 10 に示す。

5 VXLAN

サーバ仮想化技術の発展によって、1 台の物理マシン上で複数の仮想マシン (VM) を稼働させることが可能になった。このサーバ仮想化技術を用いて、VM をユーザに対して提供する Infrastrucre as as Service(IaaS) モデルのサービスは、現在のコンピューティング環境において不可欠な構成要素になっている。ユーザは、IaaS 上の VM などの仮想資源を利用することによって、ハードウェアを用意することなく IaaS クラウド上のコンピュータ資源を利用することができ、また動的な資源の追加や削除などが可能になる。しかし、IaaS 環境上で多くの VM が動作可能になった結果、その VM に接続されるネットワークの構成が問題になっている。

IaaS 環境におけるネットワーク構成の大きな問題のひとつがマルチテナンシーである。IaaS 環境において、VM はその OS からアプリケーションまで全てユーザによってコントロールされる。VM がどのようなパケットを送信するかは予想できないため、VM が接続されるネットワークはユーザごとに分ける構成をとる必要がある。また、IaaS 環境上でユーザが要件に応じて VM 毎にセグメントを分けるなど、柔軟なネットワークを構築できる必要がある。そのためには、1 つの IaaS クラウド上で、ユーザ毎、さらにはユーザ毎に複数のセグメントと、数多くのセグメントを分ける必要がある。しかし、既存のセグメントを分ける手法である VLAN は VLAN ID が 12bit であるため、4096 個のセグメントしか分けることができなかった。また、VLAN を HV 間にまたがって通す場合、HV 間の L2 スイッチ群にまで設定変更が必要であり、運用コストが高い。そこで 2011 年、Virtual eXtensible LAN(VXLAN) が提案された [8]。

VXLAN は、イーサフレームを IP でカプセル化するオーバーレイの L2 延伸技術である。VXLAN のカプセル化の概要を図 11 に示す。VXLAN では、L2 のマルチキャスト、ブロードキャスト、未知の宛先のユニキャストを L3 マルチキャストに、L2 ユニキャストを L3 ユニキャストにカプセル化する。カプセル化されたパケットを受信した VTEP(VXLAN Tunnel End Point) は、カプセル化されたイーサフレームの送信元 MAC アドレスが、外側の IP ヘッダの送信元 IP アドレスの VTEP の配下にいると学習し、この MAC アドレスと VTEP の IP アドレスのセットをエントリとして Forwarding Database (FDB) を構築する。これによって、VXLAN は各 VTEP 配下のセグメントをレイヤー 3 オーバーレイを用いた L2 延伸を実現する。

また、VXLAN はカプセル化する際に VXLAN ヘッダ内に Virtual Network Ideintifier(VNI) という 24bit の識別子を付与する。VTEP はこの VNI 毎に FDB を構築することによって、約 1677 万以上の論理セグメントを構築することができる。また、VTEP 間をレイヤー 3 オーバーレイで結ぶため、VTEP が HV 内にソフトウェア的に実装されることによって、HV 間のバックボーンネットワークに変更を加えることなく、エッジである HV 内のみの変更でネットワークの構成を変更することができる。これらの機能によって、VXLAN は IaaS クラウドにおけるネットワークの問題を解決する。この VXLAN は、2012 年現在、VMware、Arista、Brocade など、いくつかのベンダーによって実装が行われている。

クラウド WG では、2011 年にこの VXLAN を実装し、オープンソースソフトウェアとして公開している [9]。この実装は、Linux のユーザランドで動作する VXLAN ソフトウェアである。デーモンプログラムである vxland がパケットの転送や FDB の管理を行い、vxlanctl コマンドを用いて VNI 毎に tap インターフェースの作成、削除を行う。また、このソフトウェアを Vyatta[10] の拡張とした実装も公開している

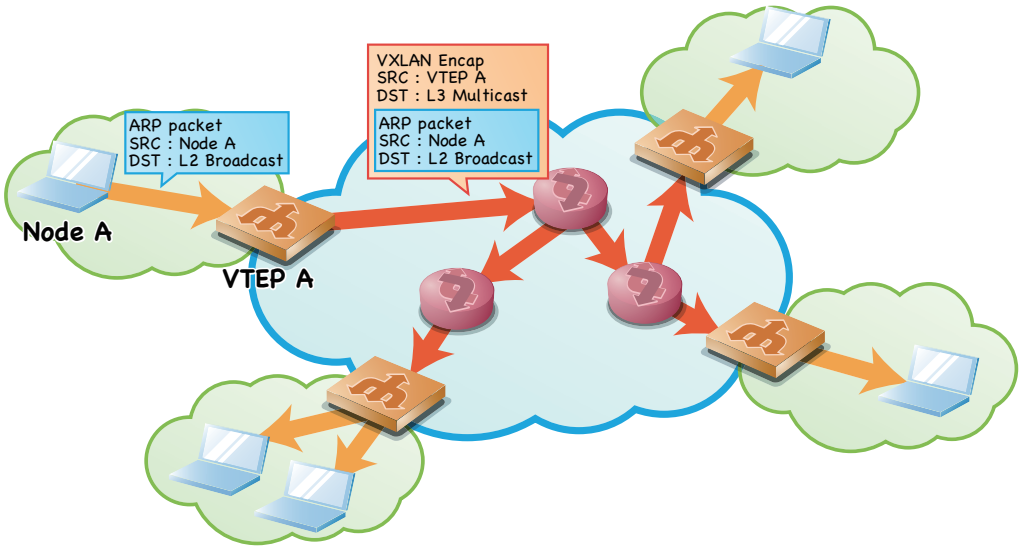


図 11: VXLAN の概要

[11]。Vyatta はソフトウェアルータであり、Linux 上の zebra[12] や iptables といったネットワークソフトウェアを統合的に扱うためのインターフェースを提供する。この VXLAN の Vyatta モジュールは、Linux 上で動作する本実装と、それを設定するための Vyatta CLI 拡張を提供する。この VXLAN 実装を用いて実際に WIDE クラウド上にて two WG のサーバの運用を行っている。詳細は 2012 年度 WIDE 報告書の「Two WG の活動報告」章をご確認頂きたい。

VXLAN によって、ネットワークを簡単に論理的に分割することが可能になった。しかし一方で、いくつかの問題も存在する。まず、VXLAN はカプセル化に IP マルチキャストを用いる。しかし現在のネットワークでは L3 ドメインを超えて IP マルチキャストを動作させることが難しい。そのため、VXLAN のオーバーレイドメインも同様に L3 ドメインを超えることが難しくなっている。また、VXLAN の VTEP 機能をゲートウェイとして HV の外に設置した場合、1 つのエッジセグメントに対して複数の VXLAN ゲートウェイが設置される場合を VXLAN はプロトコルとして考慮しない。そのため、VXLAN ゲートウェイ自体の冗長化が難しい。このように、VXLAN には実際に運用する上で問題が存在する。こういった問題に留意した上で、我々は、VXLAN を実装、運用した知見を活かしてクラウド環境におけるより柔軟なネットワークの研究を今後も継続していく予定である。

6 LISP Beta Network への参加と運用

近年、インターネットのルーティングアーキテクチャは、マルチホーミング、 トラフィックエンジニアリング等の要求に対して、スケーラビリティの確保が困難になってきている。これらの問題を解決するべく設計されたプロトコルとして、Locator/ID Separation Protocol(LISP)[13] が注目されている。LISP を導入することで、他 AS へ広告する経路を変更することなく任意のプレフィックスをインターネット上の任意の位置に配置する、マルチホームを PI アドレスやパンチングホールルーティングをせずに実現する、アンダーレイネットワークの IP スタックに関わらず任意の IP スタックを使用することなどが可能になる。これら LISP の持つ特長は柔軟で可用性の高い IaaS 環境の実現に応用できるため、クラウドワーキンググループにおいても研究及び運用を行っている。LISP の詳細なプロトコル仕様については、IETF にて公開されている Internet-Draft[13] を参照されたい。

LISP は 2007 年 7 月に Internet-Draft が提案された比較的新しいプロトコルであるため、その実験・検証を目的として LISP Beta Network[14] が運用されてい

る。クラウド WG では、2011 年 7 月より LISP の実験・検証を目的として LISP Beta Network に参加しており、藤沢 NOC 内に LISP Beta Network との接続ルータを設置している。クラウド WG では、LISP Beta Network より 153.16.68.0/24 及び 2610:D0:320B::/48 の割り当てを受けており、このプレフィックスを実験・検証のために使用している。LISP Beta Network を実験・検証に用いた例として、クラウドワーキンググループが作成した LISP XTR 実装 [15] が挙げられる。図 12 に LISP Beta Network の概略を示す。

図 12 で、detr.fujisawa は LISP Beta Network からの LISP パケットを終端するための ETR である。detr.fujisawa と pxtr.fujisawa は P2P で直接接続されており、detr.fujisawa に到達した LISP パケットはカプセル化を解除した上ですべて pxtr.fujisawa に転送される。pxtr.fujisawa では、detr.fujisawa より転送されてきたパケットをクラウドワーキンググループが独自に設置した map-server の持つ経路に従い、実験・検証用の ETR 等にカプセル化した上で転送する。このように、一旦 detr.fujisawa で LISP Beta Network からのすべてのパケットを終端し、pxtr.fujisawa から先は独自の経路制御とすることで、LISP Beta Network より割り当てを受けたプレフィックスを細分化して利用している。

クラウドワーキンググループ WG は、今後も LISP Beta Network を LISP に関連する実験・検証に使用する方針である。

7 LISP XTR の実装

LISP は既にいくつかの実装が作成されているが、IETF 76 にて相互運用性試験 [16] が行われた段階では、デュアルスタックで control-plane の機能を持ったオープンソース実装は存在していなかった。[16] の報告によると、オープンソースの LISP 実装である Open LISP、 Aless、 LISP-Click は、control-plane の機能がない。また、デュアルスタックに対応するのは Open LISP のみである。Cisco IOS、 Zlisp は control-plane の機能を持ちデュアルスタックに対応しているが、オープンソースではない。

そこでクラウドワーキンググループでは、新たにオープンソースで control-plane の機能を持ち、デュアルスタックに対応した LISP XTR の実装である *lix* を作成した。実装は [15] で公開している。lix の実装は、Linux kernel 2.6.32 及び C 言語 (gcc version 4.4.5) を使用して行った。lix では、data-plane に関連する機能はローダブルカーネルモジュール (LKM) として動作する。また control-plane に関連する機能はユーザーランドでデーモンとして動作する。lix の ITR 実装の概要を図 13 に、ETR 実装の概要を図 14 に示す。

図 13 に関して、ITR としての動作をするためには、lix のカーネルモジュールが転送するパケットをフックする必要がある。lix のカーネルモジュールはカーネル API の一種である netfilter に対してハンドラを登録し、Linux のフォワーディングエンジン内で FORWARD チェインに入ったパケットをフックする。また、ポジティブキャッシュとネガティブキャッシュの管理及びロンゲストマッチは Radix-Tree 方式で実装した。フックしたパケットの宛先アドレスがキャッシュにマッチした場合は、ヘッダ付加後、カーネル API を利用して OS 内のルーティングテーブルに基づいてパケットを送信する。キャッシュにマッチしない宛先アドレスをもつパケットをフックした場合は、その宛先アドレスを temporary cache として map-cache table に登録した後、ユーザーランド空間の control-plane を通じて map-cache の解決を試みる。このとき、カーネル空間とユーザーランドの通信には netlink インターフェースを用いる。control-plane は、map-server から結果が得られた場合、再び netlink インターフェースを通じてカーネル空間の data-plane に結果を送信する。結果が得られなかった場合は、一定時間経過後に temporary cache を消去するよう data-plane に要求する。

一方、図 14 に関して、ETR としての動作をするためには、lix のカーネルモジュールが自ホストの UDP ポート 4341 番宛のパケットをフックする必要がある。これも ITR の時と同様に netfilter に対してハンドラを登録し、Linux のルーティングエンジン内で INPUT チェインに入ったパケットをフックし、UDP ポート 4341 番宛のものを選別することで実現した。カプセル化解除後の元のパケットは、モジュールをロードした際に作成される仮想インターフェースに着信させることによって Linux のフォワーディングエンジンに再入力し、ARP または ND の解決を行ったのち EID 側のホストに到達する。

lix は LISP XTR として基本的な機能を備えている。

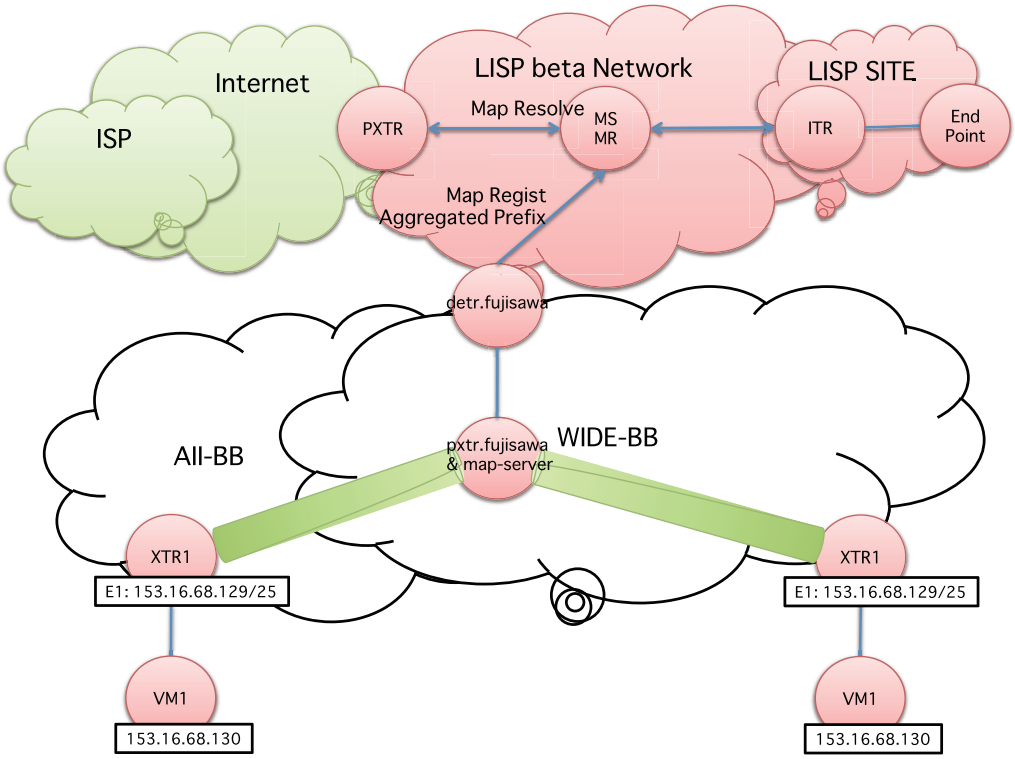


図 12: LISP Beta Network の概略

しかし、マルチキャストや複数の map-server への対応など、仕様書で定義されているが未実装の機能があるため、今後も改良を続けるとともに、クラウドワーキンググループが参加する LISP Beta Network 上で実験・検証を進める。

8 仮想環境における分散ファイル/ストレージシステムの評価指標の研究

8.1 Background

The hosting service is one of the legacy Internet services which has been provided by many Internet service providers for long time. The service was originally using physical computers located in a datacenter to host services of their customers, however, the virtualization technology changed the hosting infrastructure drastically. Many hosting services, except some mission critical services or services which require high performance, are now providing their services using virtual host computers.

The important point of virtualization in service operation is that the efficiency in multiplexing computing resources. We want to put as many virtual machines as possible as long as we can keep the service level agreement. A virtual machine migration technology contributes this purpose. If we have multiple virtual machines which does not consume a lot of resources in multiple physical machines, we can aggregate those virtual machines to fewer number of physical machines using a live migration technology without stopping them.

Usually, such a migration operation is utilized within a single datacenter only, however, considering the total efficiency of service provider operations, we need global resource mobility strategy among multiple datacenters ([17, 18, 19]). One of the essential technology to achieve this goal is a storage sys-

tem for virtual machines. When a virtual machine moves from one physical computer to another computer, both physical computer must keep providing the same virtual disk image to the virtual machine. We are typically using NFS or iSCSI technologies at this moment, however it is difficult to design a redundant and robust storage infrastructure in a wide area service operation. Recent research papers show that distributed storage management systems ([20, 21, 22, 23]) are becoming mature. We are now at the stage to start considering other choices when designing a virtual storage infrastructure.

When considering distributed storage systems, it is not enough just focusing on the I/O performance if we use them as backend storage systems for virtual machines. We need evaluation indexes suitable to the virtualization environment.

In this paper, we investigate the possibility and potential of recent distributed file/storage systems as virtual machine backend storage systems and evaluate their I/O throughput in a realistic large scale storage infrastructure which consists of 88 distributed storage nodes. We then define two evaluation indexes for the virtualization environment; *ParallelismImpactRatio* and *LatencyImpactRatio*. We evaluate impact to the throughput which may be affected by the number of virtual machines running in parallel and by the network latency assuming that virtual machines are operated in multiple datacenters geographically distributed.

8.2 Target Systems

8.2.1 Virtualization

The virtual machine hosting environment we used in this measurement was *QEMU* ([24, 25]) and *KVM* ([26, 3]). The version of the QEMU/KVM software was 1.0 (qemu-kvm-1.0). The host operating system was Linux (Ubuntu 12.04.1 LTS).

8.2.2 File/Storage Systems

We selected the following four different stores in this experiment, with the first three belonging to the hash-based distribution family, and the last one belongingto the metadata-based family. The reason of the choice is that these four systems are well-known and the implementation of these file/storage systems are publicly available. We are focusing on the performance of real implementations in this paper.

- Ceph
  Ceph ([20, 27]) is a distributed object-based storage system designed to be POSIX-compatible, and highly distributed without a single point of failure. On March 2010, Ceph client was merged into Linux kernel 2.6.34, enable easy integration with popular Linux distributions, although the software is still under development. Ceph provides multiple interfaces to access the data, including a file system, *rbd*[11], *RADOS* [28], and C/C++ binding. In this experiment, instead of the file system interface, we utilized the rbd interface, which is also supported by QEMU. This rbd interface relies on *CRUSH* [29], a Replication Under Scalable Hashing [30] variant with the support of uniform data distribution according to device weights.

- Sheepdog
  Sheepdog ([21, 31]) is a distributed object-based storage system specifically designed for QEMU. Sheepdog utilizes a cluster management framework such as *Corosync* [32] or *Apache ZooKeeper* [33] for easy maintenance of storage node grouping. Data distribution is based on the consistent hashing algorithm. QEMU supports sheepdog's object interface natively similarly to Ceph's rbd interface.

- GlusterFS
  GlusterFS ([22, 34]) is a distributed filesystem based on a stackable user space design. It relies on consistent hashing to distribute the data

[11] `http://ceph.com/wiki/QEMU-RBD/`

based on a file name, and data is stored on disk using native formats of a backend storage node. One of the interesting features of GlusterFS is that metadata management is fully distributed, therefore there is no special nodes for metadata management. GlusterFS provides POSIX semantics to its client nodes.

- XtreemFS
  XtreemFS ([23, 35]) is an open source object-based, distributed file system for wide area network. The file system replicate objects for fault tolerance and caches metadata and data to improve performance over high-latency links. XtreemFS relies on a centralized point for metadata management and for data access. The file system provided by XtreemFS is guaranteed to have POSIX semantics even in the presence of concurrent access.

TABLE 1 summarizes the properties of target systems.

8.3 Testbed Environment

We utilized *StarBED*[12] [36] which is operated by National Institute of Information and Communications Technology Japan[13]. StarBED provides more than 1000 server nodes interconnected via 1Gbps switches which can be reconfigured to topologies suitable for our experiment plans. More information is available from the web page.

In this experiment, we used 89 server nodes to build distributed file/storage system infrastructure. The server specification is shown in TABLE 2.

The upstream switch of the servers was Brocade MLXe-32.

表 2: The specification of server nodes

Model	Cisco UCS C200 M2
CPUs	Intel(R) Xeon(R) CPU X5670 @ 2.93GHz × 2
Cores	24 (with Hyper-threading)
Memory	48GB
HDD	SATA 500G × 2
NIC 0, 1, 2, 3	Broadcom BCM5709 Gigabit Ether
NIC 4, 5	Intel 82576 Gigabit Ether

8.4 Layout of Service Entities

8.4.1 Measurement Management

Fig. 15 shows the layout of service entities of each distributed file/storage system. The measurement topology consisted of 89 nodes.

The k011 node was used as a controller node from where we sent measurement operation commands. k011 was also used as a NFS server to provide shared space to record measurement result taken on each server node. All the traffic to control measurement scenarios and all the NFS traffic were exchanged using the *Management Network* (the left hand side network of Fig. 15) to avoid any affect to the measurement.

8.4.2 Distributed File/Storage Systems

All the nodes were attached to the *Experiment Network* using a BCM5709 network interface port (the right hand side network of Fig. 15). The file/storage systems consisted of 88 nodes, using from k012 to k100 except k020. Most of the nodes worked as a simple storage node entity, except a few nodes which had some special roles as described below.

- Ceph
  Ceph requires two kinds of special nodes for its operation. One is a *Metadata Server*, the other is

[12] http://www.starbed.org/
[13] http://www.nict.go.jp/en/index.html

表 1: Summary of target file/storage systems

	Ceph	Sheepdog	GlusterFS	XtreemFS
Metadata management	Distributed metadata server	n/a	Distributed	Centrilized
Data management	CRUSH	Consistent hasing	Consistent hashing	Key-Value
Storage style	Object-based	Object-based	File-based	Object-based
File system	Yes	No	Yes	Yes
QEMU native support	rbd	sheepdog	n/a	n/a
Data access entry point	Monitor node	any Sheepdog node	any GlusterFS server node	Metadata server
WAN awareness	No	No	No	Yes

a *Monitor*. In the measurement topology, k012, k014, k016, and k018 acted as metadata servers. k013, k015, k017, k019 acted as monitors. These nodes also acted as storage devices (it is called as *Object Storage Node, OSD*). The rest of the nodes acted as OSDs.

Ceph keeps multiple copies of data object for redundancy. The number of the copies kept in the system is configurable. In the experiment, we chose 2 as the replication factor in the system.

- Sheepdog
  Sheepdog requires a group communication infrastructure to monitor nodes joining and leaving the storage entity. In the experiment, Apache ZooKeeper was used for that purpose. k012 to k019 were used for the ZooKeeper service. All the nodes including k012 to k019 acted as participating nodes of Sheepdog.

  The Sheepdog storage system was formatted to use a replication factor of 3 in this experiment.

- GlusterFS
  The metadata management in GlusterFS is fully distributed, so there is no special nodes like a metadata server. GlusterFS provides the replication and striping function for redundancy and performance enhancement. In the experiment, the replication factor was set to 3 and the striping factor was set to 2.

  Since GlusterFS requires the total number of nodes to be a multiple of the value of replicas × stripes, the number of participating nodes must be divisible by 6 in this case. Therefore, k040, k060, k080, and k100 were not used as a participating node in GlusterFS. The remaining 84 nodes were used to construct the GlusterFS system.

- XtreemFS
  XtreemFS requires two kinds of special nodes. One is a *Directory Server*, the other is a *Metadata and Replica Catalog, MRC*. k013 was used as a directory server, and k012 was used as a MRC.

  As similar to other systems, XtreemFS can have multiple copies of data object. In the experiment, 3 copies of data were kept in the system.

8.4.3 Virtual Machines

All the nodes except k011 acted as QEMU hypervisors. Fig. 16 shows the topology for virtual machines. Each hypervisor has one virtual machine. The network provided at each hypervisor for virtual machines is a NATed network built by a network bridging function and NAT software. We used the *bridge-utils* package and the *dnsmasq-base* package provided for the Ubuntu 12.04.1 system.

Virtual machine disk images were served by the QEMU hypervisor. Depending on the underlying distributed file/storage systems, the following configuration parameters were used.

- Ceph
  QEMU has a built-in Ceph OSD disk image support as the rbd method. A virtual machine disk image was divided into units of the Ceph OSD object, and distributed through the OSD system.

- Sheepdog
  QEMU has a built-in Sheepdog disk image support as the *sheepdog* method[14]. Similar to Ceph, a virtual machine disk image was divided into units of the Sheepdog object and distributed among the storage entities.

- GlusterFS
  At this time of writing, QEMU does not have any direct support of GlusterFS. Since GlusterFS provides an ordinal filesystem interface, we simply mounted the GlusterFS using *FUSE* [37] on each hypervisor and put all the virtual machine disk images into the mounted directory. Each disk image was divided into two pieces (since we used the striping factor of 2), and 3 copies of each piece were distributed among the storage entities.

- XtreemFS
  There is no XtreemFS direct support in QEMU either. Similar to the GlusterFS operation, we mounted the XtreemFS volume on each hypervisor using FUSE and put all the virtual machine disk images in the directory. Each disk image was divided into units of the XtreemFS object and distributed to the storage entities.

8.5 Measurement Procedures

The main goal of this measurement is to evaluate the maximum performance we can achieve when we use distributed file/storage systems as virtual machine disk image storage systems, and to evaluate their robustness against the number of running virtual machines and network latency. The measurement tool used in the experiment was the *Bonnie++* benchmark software [38].

To evaluate the impact of the scale of the virtual machine infrastructure, we conducted four types of tests:

- Parallel-1
  In the Parallel-1 case, only one virtual machine executes the measurement program. That means the virtual machine can occupy the entire distributed file/storage systems in this case. We run the measurement program on ten different virtual machines sequentially.

- Parallel-10
  In the Parallel-10 case, 10 different virtual machines run the measurement command simultaneously. Each virtual machine is located in different hypervisors.

- Parallel-20
  Same as the Parallel-10 case, except the number of virtual machines is 20.

- Parallel-44
  Same as the Parallel-10 case, except the number of virtual machines is 44.

We also performed the same measurement in a network environment with different latency aiming to emulate wide area network as shown in Fig. 17. We divided 88 storage nodes into 4 groups, each of which has 22 nodes. Nodes in the same group can communicate without any delay, however, nodes in different group have 10ms delay in each direction. This configuration roughly emulates four datacenters located in four different locations.

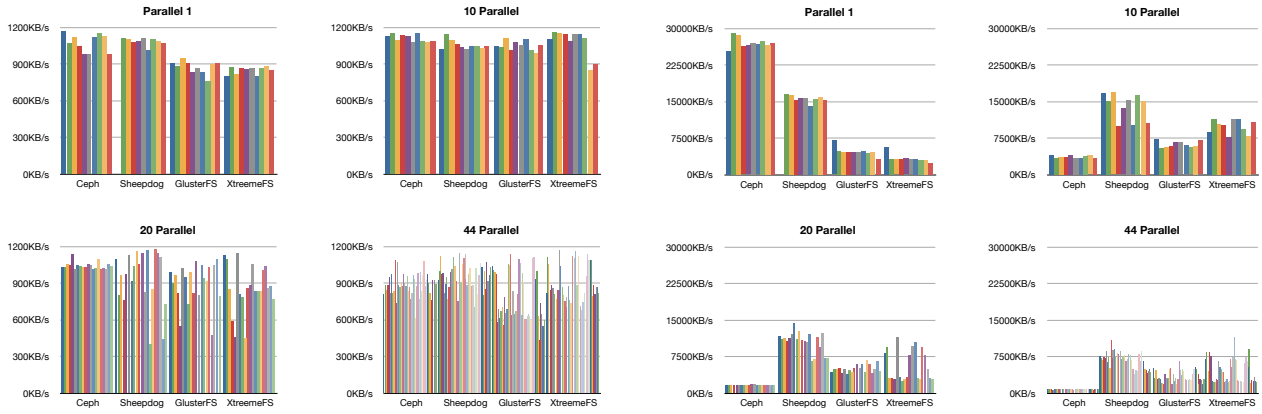
8.6 Results

8.6.1 Write Throughput

Fig. 18 shows the result of the throughput measurement of write operations. Each bar in the figure corresponds to each virtual machine.

In character write operations (Fig. 18(a)), we can observe Ceph and Sheepdog show slightly better performance than GlusterFS and XtreemFS in the Parallel-1, Parallel-20, and Parallel-44 cases. When the number of parallel operation increases, fluctuation of the throughput is observed especially in

[14] `http://github.com/collie/sheepdog/wiki/Getting-Started`



(a) Character write

(b) Block write

図 18: Write throughput

Sheepdog, GlusterFS, and XtreemFS cases. Ceph is more stable than others.

In block write operations (Fig. 18(b)), Ceph shows twice as much throughput as that of Sheepdog and five times as much throughput as those of GlusterFS and XtreemFS cases, as long as the number of running virtual machine is one. When we run multiple virtual machines in parallel, Sheepdog gives better performance than others. Sheepdog, GlusterFS, and XtreemFS show some fluctuation in their throughput results. Ceph works more stable than others, however, its throughput is worse especially when the number of parallel virtual machines increases.

8.6.2 Read Throughput

Fig. 19 shows the result of the throughput measurement of read operations.

In character read operations (Fig. 19(a)), GlusterFS and XtreemFS perform slightly better than Ceph and Sheepdog. We can observe many missing bars in the result of GlusterFS and XtreemFS. This is expected behavior of Bonnie++. When a test operation completes in too short time, Bonnie++ does not provide any result because it may give inaccurate result. For precise measurement, we need to increase the amount of read operation in the measurement procedure so that we can get reasonable operation time to calculate trustable value of read throughput.

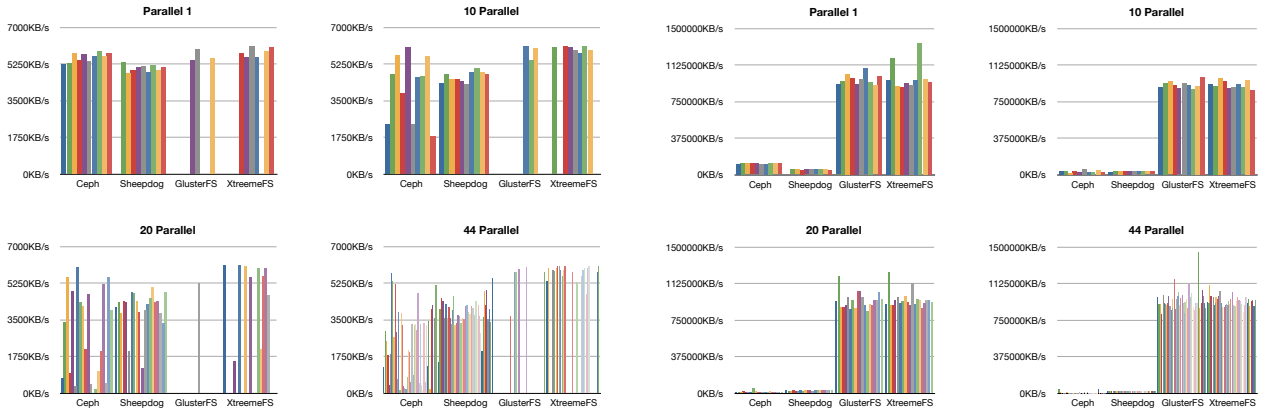
In block read operations (Fig. 19(b)), GlusterFS and XtreemFS give almost similar performance and they are quite better than Ceph and Sheepdog. In the Parallel-1 case, GlusterFS performs around 8 times and 18 times better than Ceph and Sheepdog respectively. In the Parallel-44 case, GlusterFS performs around 165 times and 48 times better than Ceph and Sheepdog respectively.

We can also observe the number of running virtual machines in parallel affects the performance in Ceph and Sheepdog cases. GlusterFS and XtreemFS seem to be more robust in parallel operations.

8.6.3 Throughput with Latency

Fig. 20 shows the impact of network latency. In each chart in Fig. 20, the left hand side chart shows the result with no latency, and the right hand side chart (with the gray background) shows the result with 20ms latency as described in section 8.5.

Different from the previous figures, the bars in Fig. 20 show the average throughput of multiple virtual machines. The blue, green, yellow, and red bars indicate the average throughput of the Parallel-1, Parallel-10, Parallel-20, and Parallel-44 cases.



(a) Character read

(b) Block read

図 19: Read throughput

We can observe that in both read and write operation cases, network latency affects the performance of the operations significantly, except for the read operations of GlusterFS and XtreemFS. Their read throughput did not show big degradation even though there was 20ms network latency.

8.6.4 Scalability

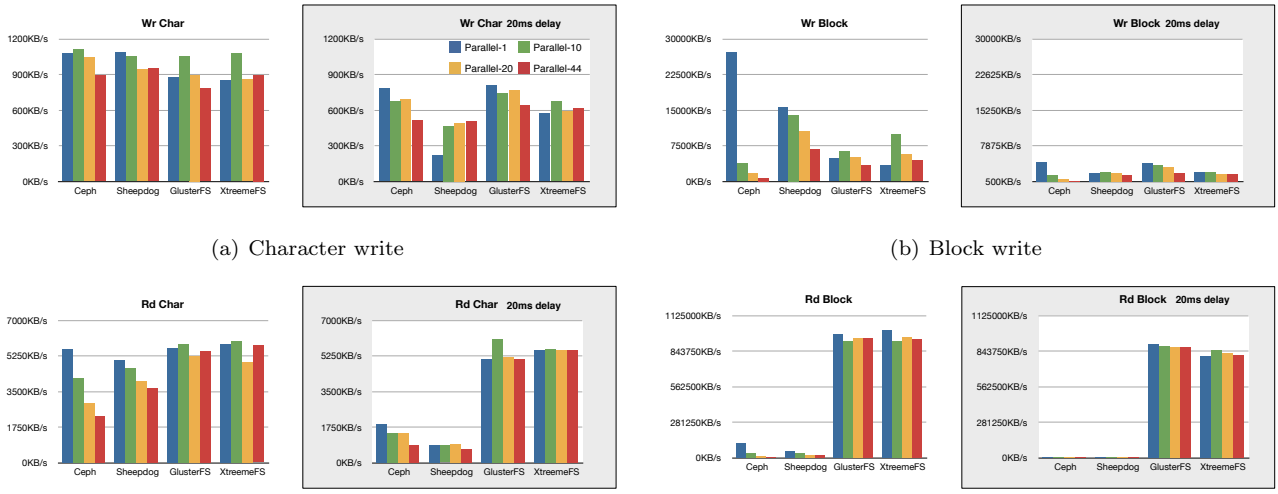
We can also observe that scalability of GlusterFS and XtreemFS is better than that of Ceph and Sheepdog from Fig. 20. When the number of virtual machines running in parallel increases, Ceph and Sheepdog tend to show performance degradation, while GlusterFS and XtreemFS can keep almost same performance regardless of the number of virtual machines.

8.7 Discussion

We observed Ceph and Sheepdog provided good performance in character write operations. Although the difference was not so big actually. The performance of Ceph was around 122% of that of GlusterFS. Sheepdog was almost the same.

Interestingly, the performance of block write operations of Ceph was better than others when we ran only one virtual machine in the entire system, however, when we ran multiple virtual machines, the performance went worst. This result means that the concurrent access to Ceph OSDs have some bottleneck. Technically, the mechanism to provide virtual disk in Ceph is similar to that of Sheepdog. We were initially expecting Ceph and Sheepdog would show similar performance trend, but they were different. In this measurement experiment, we used the rbd method provided by QEMU to serve virtual disks to virtual machines. That method is simply using Ceph OSDs as distributed object stores, and is not using the filesystem part of Ceph. If we used Ceph as a filesystem as similar to GlusterFS and XtreemFS, the result might be different.

For read operations, GlusterFS and XtreemFS provided better performance in both character read and block read operations. Especially in block read operations of the Parallel-44 case, GlusterFS was 164 times better than Ceph, and 48 times better than Sheepdog. XtreemFS performed similarly. The difference between these two groups was that we used QEMU direct support functions for Ceph and Sheepdog (the rbd method as described before, and the sheepdog method respectively) to provide virtual disks to virtual machines, while we used GlusterFS and XtreemFS through hypervisor's filesystem. We think the caching strategy of hypervisor's filesystem contributed the better read performance. As noted



(a) Character write

(b) Block write

(c) Character read

(d) Block read

図 20: Impact of latency and impact of the number of virtual machines

before, Ceph can also be used as a filesystem. The performance measurement when we use Ceph as a filesystem is one of our future works.

Since we are targeting a wide are operation of virtual machines for geographically distributed datacenters, the impact of latency, and the impact of the number of virtual machines in operation in the system are important evaluation factors.

Fig. 21 shows the parallelism impact ratio calculated from equation 1.

$$ParallelismImpactRatio = \frac{Throughput_{Parallel-X}}{Throughput_{Parallel-1}} \quad (1)$$

Where $Throughput_{Parallel-1}$ is average throughput of I/O operations when we run only one virtual machine at one time. $Throughput_{Parallel-X}$ is average throughput of I/O operations of the parallel case $Parallel - X(X = 10, 20, or 44)$.

In write operations, we can observe the trend that the performance becomes worse as the number of virtual machines increases. In read operations, Ceph and Sheepdog have the similar trend, however, GlusterFS and XtreemFS does not have any impact of parallelism.

Fig. 22 shows the latency impact ratio calculated from equation 2.

$$LatencyImpactRatio = \frac{Throughput_{Delay20}}{Throughput_{Delay0}} \quad (2)$$

Where $Throughput_{Delay0}$ is average throughput measured in the base topology without any latency configuration, $Throughput_{Delay20}$ is average throughput measured in the topology with the latency configuration as shown in Fig. 17.

We can observe that GlusterFS is the most robust system against network latency than others. XtreemFS is also good, however, we can see notable throughput degradation in block write operations performed by multiple virtual machines in parallel.

The performance degradation of Sheepdog is most significant when there is a latency in a network.

In block read/write operations of Ceph and Sheepdog, the ratio goes better when the number of parallel virtual machines increases. This means that the impact of network latency becomes smaller and smaller when we have more number of virtual machines. It seems that XtreemFS is also showing similar trend, however, we think we need more data to say so.

For character read/write operations, we cannot see such trend as that of block operations. It seems that the impact of latency is constant in most cases. Only

the character read operations of Sheepdog is showing increasing throughput as the number of virtual machine goes up, however, we think we need more data to be confident.

One another important evaluation factor which is not investigated this time is the data transfer efficiency. When we perform a certain amount of read or write operations, we need to measure how much amount of background traffic is required on each distributed file/storage system. The smaller background traffic is better. This is our future task.

8.8 Conclusion

The virtualization technology enabled us to operate more number of computers to host real services. One of the merit of virtualization is that it makes it easy to move virtual resources among physical machines. To achieve more efficient operation of datacenters, we need to aggregate more virtual resources to fewer physical machines. To do that, distributed file/storage system which can be operated in a wide area network is required.

In this paper, we picked Ceph, Sheepdog, GlusterFS, and XtreemFS and evaluated them as a backend storage system for virtual machines. The evaluation result shows that GlusterFS and XtreemFS provide far better read performance (we observed 164 times better performance at maximum). Sheepdog provides better block write performance.

We defined two new storage evaluation indexes in the virtualization environment; *ParallelismImpactRatio* and *LatencyImpactRatio*. From these indexes, we observed that the number of virtual machines running in parallel does matter for all the systems. When the number grows, the throughput goes down, however, for block read operations, GlusterFS and XtreemFS are not affected by this factor. The latency impact exists, however, we observed the trend that when we increase the number of virtual machines, the impact is going to small.

From what we have achieved from the experiment, our current suggestion of the distributed file/storage system for virtual machine image storage is GlusterFS or XtreemFS. We still need to evaluate more factors, such as data transfer efficiency, which is important especially in a wide area operation. We will keep investigating the distributed file/storage systems to build better wide area virtualization environment.

Acknowledgments

We would like to thank Toshiyuki Miyachi and Hiroshi Nakai for their support at StarBED and many useful suggestions and comments. We also thank all the staffs at StarBED for helping our experiment.

9 まとめ

今年度はWIDEクラウド基盤の運用拡張に加え、クラウド基盤自体の導入管理を支援する技術、クラウド基盤運用管理を補佐するためのモニタリング機能の研究活動を通じ、より高度なクラウド基盤運用に向けた活動を行った。また広域クラウドの実現のための、広域経路制御技術、広域ストレージ技術などの研究に取り組んだ。WIDEクラウドワーキンググループでは、今後も規模性にすぐれた広域分散環境でのクラウド運用を可能とする技術の研究開発を継続していく予定である。

参考文献

[1] VMware, Inc., March 2010. http://www.vmware.com/.

[2] Citrix Systems, Inc., March 2010. http://www.xen.org/.

[3] R.A. Harper, A.N. Aliguori, and M.D. Day. KVM: Kernel-based Virtual Machine. http://www.linux-kvm.org/.

[4] Hirochika Asai, Yuji Sekiya, Keiichi Shima, and Hiroshi Esaki. *Management Information Base for the Virtual Machine Monitoring*. IETF, October 2012. draft-asai-vmm-mib-01.

[5] Michael MacFaden, Juergen Schoenwaelder, Tina Tsou, and Cathy Zhou. *Definition of Managed Objects for Virtual Machines Controlled by a Hypervisor*. IETF, July 2012. draft-schoenw-opsawg-vm-mib-01.

[6] Yoshihiro Okamoto, Satoru Noguchi, Satoshi Matsuura, Atsuo Inomata, and Kazutoshi Fujikawa. Koshien-Cloud: Operations of Distributed Cloud as A Large Scale Web Contents Distribution Platform. In *In Proceedings of The 3rd Workshop on Company, Campus and Community Networking - Technology, Management and Ethics (C3NET 2012). IEEE*, pp. 363–368, July 2012.

[7] Igor Sysoev. nginx [engine x]. http://nginx.org/.

[8] Mallik Mahalingam, Dinesh G. Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T. Sridhar, Mike Bursell, and Chris Wright. *VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. IETF, August 2011. draft-mahalingam-dutt-dcops-vxlan-00.

[9] Ryo Nakamura. userland vxlan implementation. https://github.com/upa/vxlan.

[10] Vyatta Inc. vyatta. http://www.vyatta.com/.

[11] Ryo Nakamura. userland vxlan module for vyatta. https://github.com/upa/vxlan-vyatta.

[12] Zebra Project. GNU Zebra. http://www.zebra.org/.

[13] Dino Farinacci, Vince Fuller, Dave Meyer, and Darrel Lewis. *Locator/ID Separation Protocol (LISP)*. IETF, November 2012. draft-ietf-lisp-24.

[14] . LISP Beta Network. http://www.lisp4.net/.

[15] Yukito Ueno. LISP XTR implementation. https://github.com/edenden/lix.

[16] IETF 76. LISP Interoperability Testing. http://tools.ietf.org/agenda/76/slides/lisp-4.ppt.

[17] Eric Harney, Sebastien Goasguen, Jim Martin, Mike Murphy, and Mike Westall. The efficacy of live virtual migrations over the internet. In *Proceedings of the 2nd international workshop on Virtualization technology in distributed computing (VTDC'07)*, 2007.

[18] Cisco Systems, Inc., VMware, Inc. Virtual Machine Mobility with Vmware VMotion and Cisco Data Center Interconnect Technologies. Technical report, Cisco Systems, Inc., VMware, Inc., 2009.

[19] Takahiro Hirofuchi, Hirotaka Ogawa, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. A Live Storage Migration Mechanism over WAN for Relocatable Virtual Machine Services on Clouds. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, pp. 460–465, 2009.

[20] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI'06)*, pp. 307–320. USENIX, 2006.

[21] Kazutaka Morita. Sheepdog: Distributed Storage System for QEMU/KVM. Linux.conf.au 2010, January 2010.

[22] Gluster Inc. Gluster File System Architecture. Technical report, Gluster Inc., 2010.

[23] Eugenio Cesario, Toni Cortes, Erich Focht, Matthias Hess, Felix Hupfeld, Björn Kolbeck, Jesús Malo, Jonathan Martí, and Jan Stender. The XtreemFS Architecture. In *Linux Tag*, 2007.

[24] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC'05)*, pp. 41–41, April 2005.

[25] Fabrice Bellard. QEMU: Open Source Processor Emulator. http://www.qemu.org/.

[26] R.A. Harper, A.N. Aliguori, and M.D. Day. KVM: The Linux Virtual Machine Monitor. In *Proceedings of the Linux Symposium*, pp. 225–230, 2007.

[27] Inktank Storage, Inc. Ceph. http://ceph.com/.

[28] Sage A. Weil, Andrew W. Leung, Scott A. Brandt, and Carlos Maltzahn. RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters. In *Proceedings of the 2th international Petascale Data Storage Workshop (PDSW'07)*, pp. 35–44, November 2007.

[29] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC'06)*, November 2006.

[30] R.J. Honicky and Ethan L. Miller. Replication under scalable hashing: A family of algorithm for scalable decentralized data distribution. In *Proceedings of the 18th International Parallel & Distributed Processing Symposium (IPDPS 2004)*, April 2004.

[31] Kazutaka Morita. Sheepdog. http://www.osrg.net/sheepdog/.

[32] Steven Dake, et al. Corosync. http://www.corosync.org/.

[33] The Apache Software Foundation. Apache ZooKeeper. http://zookeeper.apache.org/.

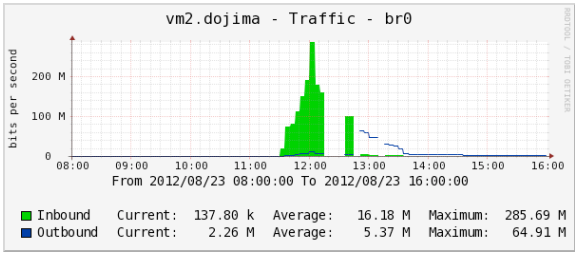
[34] Gluster Inc. Gluster. http://www.gluster.org/.

[35] The Contrail E.U. project, The MoSGrid project, and The First We Take Berlin. XtreemFS. http://www.xtreemfs.org/.

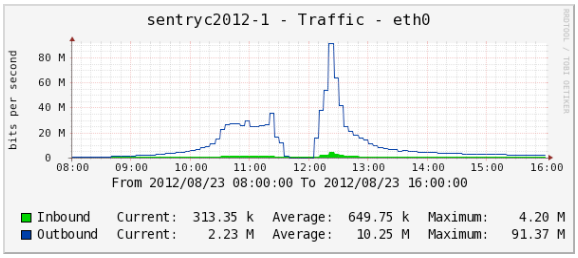
[36] Toshiyuki Miyachi, Kenichi Chinen, and Yoichi Shinoda. StarBED and SpringOS: large-scale general purpose network testbed and supporting software. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools (valuetools'06)*, October 2006.

[37] Miklos Szeredi. FUSE: Filesystem in Userspace. http://fuse.sourceforge.net/.

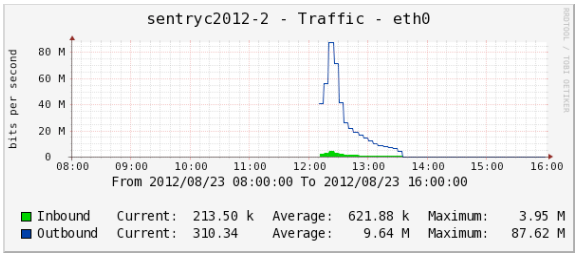
[38] Russell Coker. Bonie++. http://www.coker.com.au/bonnie++/.



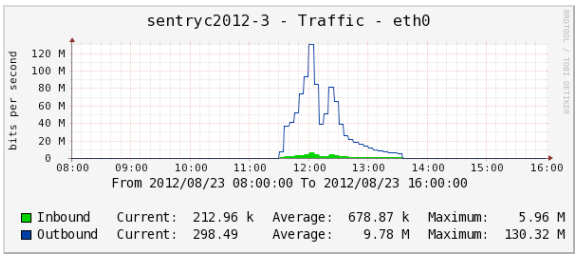
(a) nginx



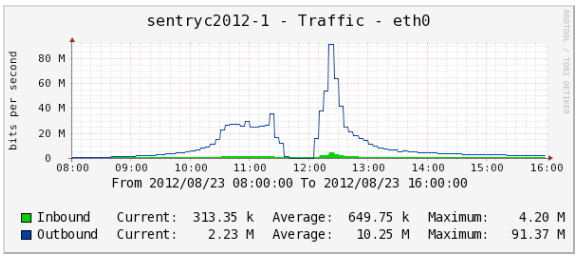
(b) VM1



(c) VM2



(d) VM3



(e) VM4

図 9: パターン 2 構成でのトラフィック

```
worker_processes 4;
error_log /koshien/logs/error.log notice;
pid /tmp/nginx.pid

events {
    worker_connections 65536;
}

http {
    include mime.types;
    default_type application/octet-stream;

    access_log /koshien/logs/access.log combined;

    sendfile on;
    keepalive_timeout 10;

    upstream sentryc {
        server 10.10.21.61:80;
        server 10.10.21.62:80;
        keepalive 8192;
    }

    server {
        listen    [::]:80 default ipv6only=on;
        listen    80;
        server_name nginx.cloud.wide.ad.jp;
        server_name 203.178.131.232;
        server_name 2001:200:dff:fff1:216:3eff:fe00:beef;

        location / {
            proxy_http_version 1.1;
            proxy_set_header Connection \";
            proxy_pass http://sentryc;
            }
    }
}
```

図 10: nginx config (抜粋)

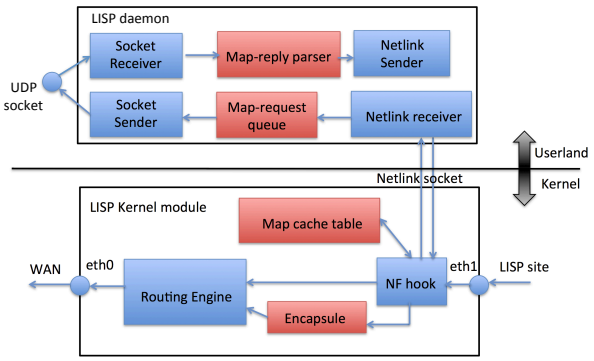


図 13: ITR 実装の概要

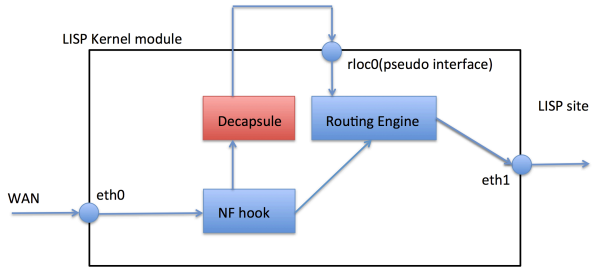


図 14: ETR 実装の概要

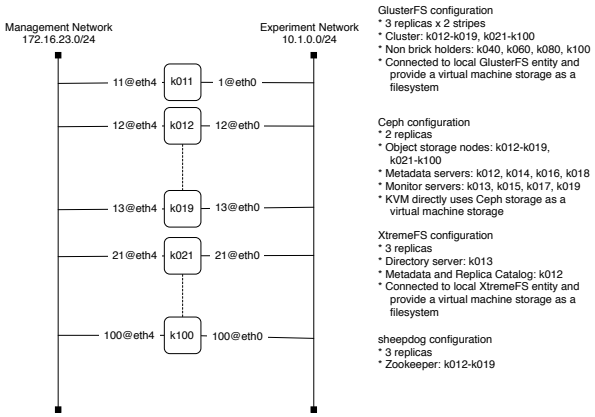


図 15: Node layout of distributed file/storage systems

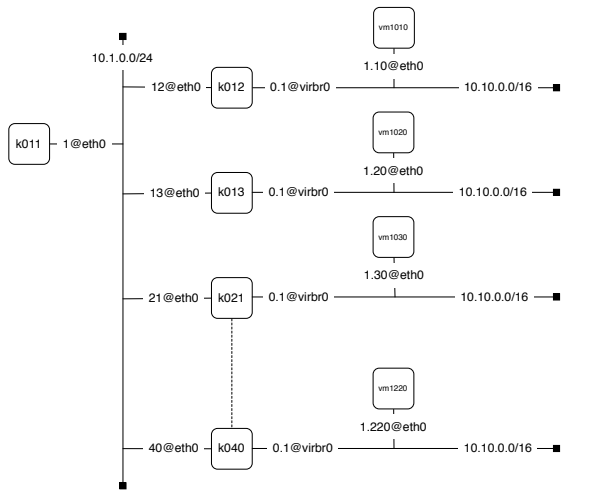


図 16: The layout of virtual machines

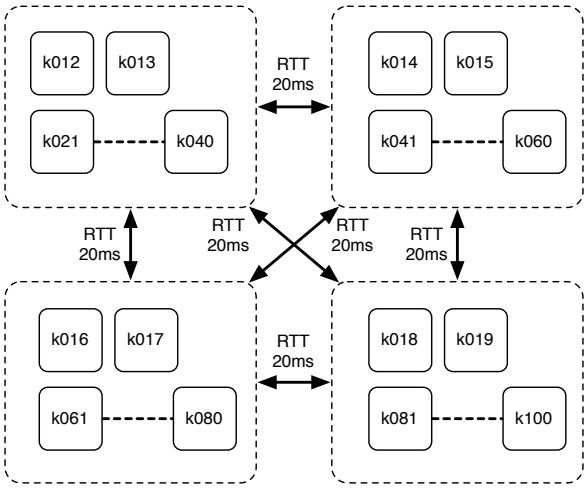


図 17: The latency parameter configuration

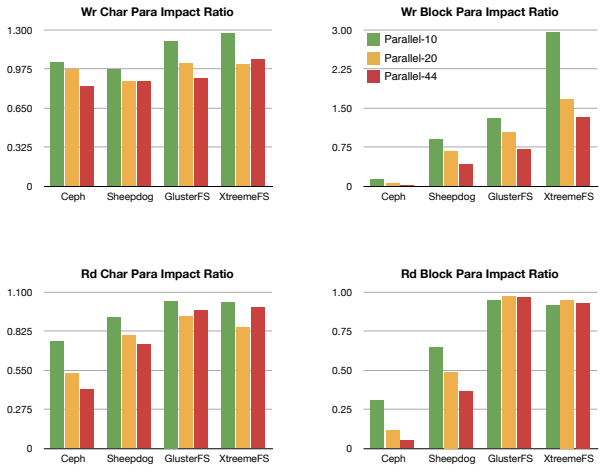


図 21: Parallelism impact ratio

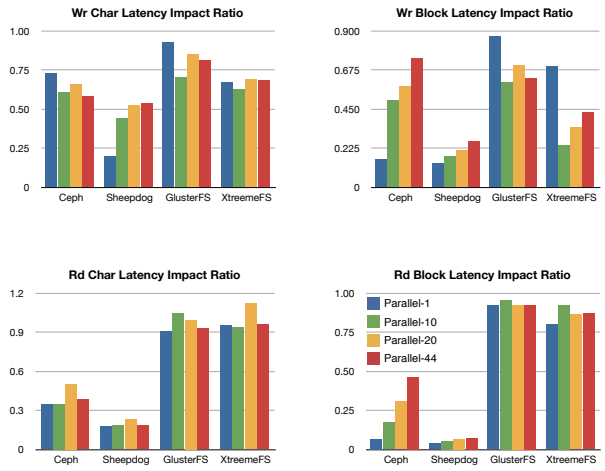


図 22: Latency impact ratio