Vol.2013-OS-124 No.5
2013/2/28

インタークラウドにおける仮想インフラ構築システムの提案

高野 了成[1,a)]　中田 秀基[1]　竹房 あつ子[1]　柳田 誠也[1]　工藤 知宏[1]

概要：クラウド間で柔軟に資源を融通したり、複数クラウドの資源を組み合わせてサービスを提供するための仮想インフラ構築技術の開発が求められている。本論文では、そのような事例の一つとして、インタークラウド環境において、IaaS 事業者に対して、計算機、ネットワーク、ストレージ資源を提供する HaaS サービスモデルおよび HaaS 資源管理システム Iris (Inter-cloud Resource Integration System) を提案する。Iris はデータセンタ内部の各種資源の管理および仮想化と、データセンタ間ネットワークの制御を行う。Iris のプロトタイプ実装を用いた実証実験では、Apache CloudStack で管理された IaaS システムに変更を加えることなく、HaaS の資源を透過的に提供できることを示す。実験結果から、HaaS システム上へのユーザ VM のデプロイ時間は、IaaS システム上と遜色ないことが分かった。さらに、各要素技術に対する機能または性能上の課題について議論する。

1. はじめに

クラウド技術が広く浸透し始めている。クラウドを中心としたエコシステムの構築、企業内 IT システムのクラウド化、そしてパブリッククラウドとのシームレスな連携などを目指し、ハイブリッドクラウドやインタークラウドへの注目が高まっている。このような背景から、クラウド間で柔軟に資源を融通したり、複数クラウドの資源を組み合わせてサービスを提供するための仮想インフラ構築技術の開発が求められている。

我々はインタークラウド環境において統合的に資源管理を行うミドルウェア GridARS [1], [2] の開発を行っている。GridARS は、地理的に分散された計算機群やそれらを結ぶネットワークなどの異種の資源を同時予約し、確保した資源の利用状況をモニタリングできる。アプリケーションの実行環境としては、分散環境上にユーザ専用の仮想的なクラスタ計算機を構築し、予約時間にアプリケーションを自動実行する技術を開発してきた [3]。さらに、アプリケーションからネットワークパスを制御する Web サービスインタフェースのフォーラム標準化 [4]、研究教育ネットワークへの導入を進めている。

本論文では、上記の開発で得られた知見を活かし、IaaS (Infrastructure as a Service) 事業者からの要求に対して、ネットワーク越しにハードウェア資源を貸し出す HaaS (Hardware as a Service) サービスモデルを提案する。IaaS サービスの需要は大きく変動するので [5]、ピーク需要に対する設備投資はコストが高い。そこでピーク時は HaaS 事業者から資源を借りて IaaS 環境を拡張し、自らのサービスとして提供することで、負荷をオフロードすることが本サービスの狙いである。HaaS サービスを実現するため、資源管理システム Iris (Inter-cloud Resource Integration System) を設計し、その妥当性を検証する。本論文では、プロトタイプ実装を用いて、既存の CloudStack システムに対して、要求に応じた資源提供が可能なことを示す。

以下、2 節で代表的な IaaS 管理ソフトウェアである Apache CloudStack の概要を述べる。3 節で HaaS システムへの要求をまとめ、HaaS サービスモデルを提案する。4 節で HaaS 資源管理システム Iris を提案する。そのプロトタイプ実装の動作の検証と基本的な性能評価を 5 節で行う。実験結果から得られた知見を 6 節に示す。7 節で関連研究に言及し、最後に 8 節でまとめを行う。

2. Apache CloudStack

我々が目指す HaaS サービスへの要求を抽出するために、IaaS クラウド構築・管理ソフトウェアの実例として、Apache CloudStack [6] の全体構成を俯瞰する。CloudStack は、元々 Cloud.com 社（旧 VMOps 社）にて商用製品として開発が始まり、現在は Apache プロジェクトの一つとしてオープンソースソフトウェアとして開発が継続されている。データセンタ事業者によるクラウドサービス、研究機関によるアカデミッククラウドにおける国内外の利用実績も多い。開発言語は Java であり、対応している仮

[1] 独立行政法人 産業技術総合研究所 情報技術研究部門
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST)
[a)] takano-ryousei@aist.go.jp

ⓒ 2013 Information Processing Society of Japan

想計算機モニタ（以下、VMM と記す）には Xen、KVM、VMWare がある。

CloudStack の全体構成を**図 1** に示す。管理サーバ（Management Server）は、ユーザインタフェースを提供し、仮想マシンインスタンス（以下、ユーザ VM と記す）のプロビジョニング、資源管理を行う。計算ノード（Computing Node）は、ユーザ VM が動作する計算機である。プライマリストレージ（Primary Storage）は、ユーザ VM 用のストレージ領域であり、セカンダリストレージ（Secondary Storage）は、ユーザ VM のテンプレートイメージ、ISO イメージ、スナップショット用のストレージ領域である。クラスタ（Cluster）は複数の計算ノードから構成され、プライマリストレージを共有する。ポッドは（Pod）一つ以上のクラスタから構成され、単一の L2 ネットワークを共有する。複数種類の VMM を利用するなどの運用を除き、ポッド内のクラスタは一つとすることが一般的である。ゾーン（Zone）は、管理サーバと一つ以上のポッド、セカンダリストレージから構成される。

CloudStack が作成する仮想マシンには、ユーザの要求によって作成されるユーザ VM の他に、ルータ、DHCP、ファイアウォールなどを提供する仮想ルータ VM (VRVM)、ユーザに仮想マシンのコンソール (VNC) アクセスを提供するコンソールプロキシ VM (CPVM)、セカンダリストレージを提供するセカンダリストレージ VM (SSVM) と呼ばれるシステム VM が存在する。システム VM は、必要に応じて作成されるので、その数は動的に増減する。

ユーザ VM は複数の論理ネットワークに属する。主なものは、ユーザ VM 間の通信に用いるゲストネットワーク、管理サーバとの通信用の管理ネットワーク、ストレージアクセス（NFS など）用のストレージネットワーク、外部ネットワークアクセス用のパブリックネットワークである。外部ネットワークアクセス時の NAT 機能やゲストネットワークにおける DHCP 機能は前述の VRVM が提供する。さらにゾーンには基本ゾーンと拡張ゾーンの 2 種類が存在し、利用できるネットワークモードが異なる。両者とも単一の L2 ネットワークをテナント（IaaS 利用者）間で共有する共有ネットワークは利用できるが、VLAN を用いる隔離ネットワークを利用できるのは拡張ゾーンのみである。共有ネットワークでは、セキュリティグループと呼ばれる L3 レベルのフィルタリングによる隔離が可能である。

3. HaaS サービスモデル

CloudStack などを用いて運用されている IaaS 事業者に対して、要求に応じて、ネットワーク越しにハードウェア資源を貸し出す HaaS サービスモデルを提案する。一般的に HaaS と IaaS は同義として言及されることが多いが、本論文では、HaaS は IaaS に資源を提供する、より下位層のクラウドサービスであると定義する。

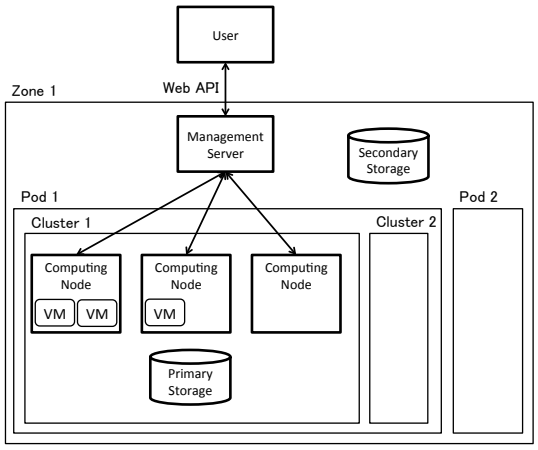


図 1 CloudStack の構成

3.1 要求分析

IaaS 事業者から HaaS システムに対する要求をまとめる。

既存 IaaS システムとの親和性 IaaS システムから HaaS の資源を透過的にアクセスしたい。利用できる VLAN 範囲等に制約がない。また、特定の IaaS システムに依存せずに利用したい。

設定作業の最小化 IaaS システムを改変したり、設定を変更したくない。IaaS データセンタ内に HaaS と接続するためにゲートウェイを設置する程度は許容できる。

資源提供の単位の自由度 計算ノード、ポッド、ゾーンなど、IaaS 事業者が必要とする単位で資源を授受したい。

安定したデータセンタ間通信 データセンタ間に安定した通信路を確立し、データセンタを跨がることによる性能低下を回避したい。

続いて、HaaS 事業者から HaaS システムに対する要求をまとめる。

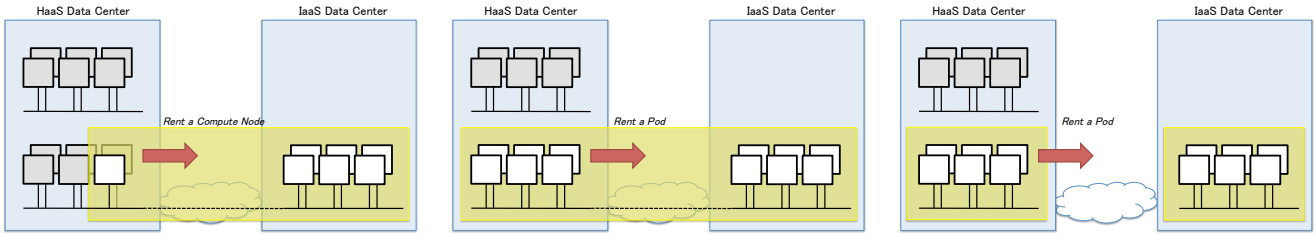
制御ネットワークの隠蔽 IaaS の立場では借りる計算ノードは物理計算機に近いほど望ましいが、HaaS 事業者の立場では計算ノードから HaaS の制御ネットワークへのアクセスは禁止したい。

複数テナントへの対応 複数の IaaS 事業者 (テナント) に対して資源を提供できる。この際、データセンタ内のネットワークでは通信性能とセキュリティを適切に隔離したい。

3.2 HaaS サービスモデルとその分類

計算資源を貸し出す単位とネットワーク到達性の観点から、HaaS サービスモデルを**図 2** に示した 3 つに分類する。図中の黄色の矩形は L2 ネットワークの境界を示している。IaaS 事業者に貸し出す資源の単位としては、計算ノード単体とポッドの二つを考える。ポッドは計算ノードの集合であり、L2 ネットワークを共有する。ただし、他の HaaS 資源とはネットワークが隔離されている。ネットワーク到達性としては、IaaS 事業者の L2 ネットワーク内に資源を

ⓒ 2013 Information Processing Society of Japan



(a) 計算ノード単位 (L2)　(b) ポッド単位 (L2)　(c) ポッド単位 (L3)

図 2　HaaS サービスモデル

配置する方法と、別の L2 ネットワークとして分離する方法を考える。

図のモデル (a) とモデル (b) では、IaaS データセンタと HaaS データセンタ間をトンネルで接続することで、L2 ネットワークを拡張する。一方、モデル (c) では、明示的にルータ経由で HaaS データセンタの資源を利用する。IaaS システムでは L2 ネットワークとしての到達性を前提としている機能があるので、その場合はモデル (a) またはモデル (b) を選ぶ必要がある。これらの場合は、IaaS 利用者に対してデータセンタの違いを明らかにしないが、ユーザに対して遅延の存在を明らかにしたい場合はモデル (c) を選べばよい。

4. Iris: HaaS 資源管理システム

4.1 概要

提案モデルを実現するための HaaS 資源管理システム Iris (Inter-cloud Resource Integration System) の概要を**図 3** に示す。IaaS システムを管理しているクラウド管理者は、資源不足によりサービルレベルを維持するのが困難になる兆候を捉えると、Iris の資源コーディネータを介して計算ノードやネットワークを確保し、IaaS システムを増強する。

本システムは、資源コーディネータ、各種資源管理マネージャとデータセンタ間を接続するためのゲートウェイから構成される。資源コーディネータと各資源管理マネージャは Web サービスインタフェースを介して通信する。資源管理マネージャとして、計算ノードを管理、制御する CRM (Compute Resource Manager) と、ゲートウェイおよびデータセンタ間のネットワークを制御する NRM (Network Resource Manager)、ストレージを制御する SRM (Storage Resource Manager) が存在する。以下、各コンポーネントの詳細について述べる。

4.2 Web サービスインタフェース

IaaS 事業者から HaaS システムに対して、各種資源を予約、プロビジョニング、モニタリングする Web サービスインタフェースは、G-lambda プロジェクト [7] で策定している GNS-WSI (Grid Network Service-Web Services Interface)

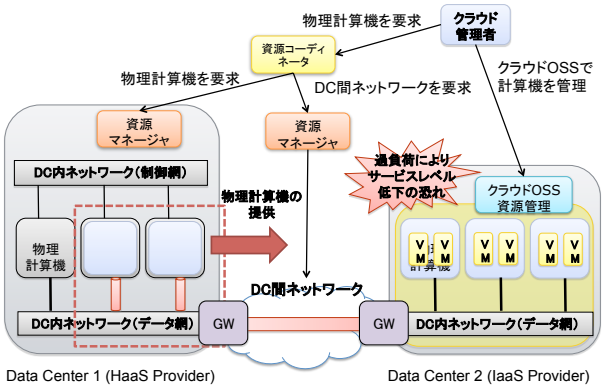


図 3　HaaS 資源管理システム Iris の概要

バージョン 3 を基に拡張する。Iris は、GNS-WSI3 を介して、複数ドメインで動作する計算機、ネットワーク、ストレージ資源マネージャと連係し、必要な資源を同時予約して提供できる。GNS-WSI では各資源を予約するときに計算機の台数やネットワークの帯域などといった要求仕様をプロパティとして指定することができる。Iris では、HaaS サービスの用途に応じて、プロパティを追加定義する。

4.3 計算資源管理

既存のクラウドは、仮想計算機や VLAN を用いて、物理資源上に 1 段階の仮想化を行っているが、HaaS システムでは 2 段階の仮想化が必要である。例えば、IaaS システムから HaaS の制御ネットワークを隠蔽するために、提供する計算ノードから特定のネットワークインタフェースを取り除く必要がある。その上で、CloudStack からユーザ VM などをデプロイするには、計算ノードの実行環境内で KVM を動作させなければならない。

以下、物理計算機を L_0、n 段目の仮想計算機を L_n と呼ぶ。したがって、**図 4** に示す通り、HaaS システムは IaaS システムに対して L_1 計算機を提供し、IaaS システムは、IaaS 利用者に対して L_2 計算機を提供することになる。

L_1 VM の実装技術として、仮想マシンと OS コンテナを考える。Linux ではそれぞれ Nested KVM [8] と Linux Containers (LXC) [9] が利用可能である。Nested KVM は、Intel VT や AMD-V などの CPU の仮想化支援機構を、KVM のゲスト OS から利用できる仕組みであり、KVM を入れ子状に実行可能となる。性能オーバヘッドは大きい

ⓒ 2013 Information Processing Society of Japan

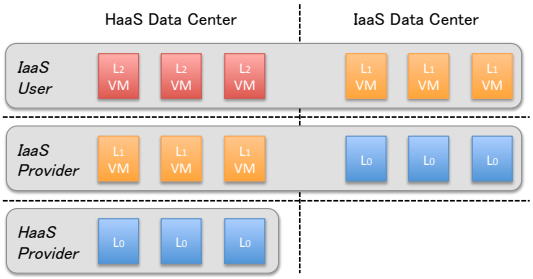


図 4 HaaS システムと仮想化階層

が、ユーザ VM 間およびホスト OS との隔離は強い。Linux カーネル 3.1 以上で対応している。LXC は OS レベルの仮想化技術である。コンテナ間でカーネルは共有されるので、性能オーバヘッドは小さいが、隔離は弱い。

4.4 データセンタ間ネットワーク資源管理

データセンタを跨がることによる性能低下を極力回避するために、十分な帯域と帯域の保証が必要である。近年、研究教育ネットワークでは、SINET Bandwidth on Demand や Internet2 ION など、必要なときに必要な帯域を必要な地点間に確保する帯域オンデマンドサービスを提供し始めている。さらに複数のネットワークプロバイダに跨がるネットワークパスが必要なこともある。我々はそのような場合を想定して OGF NSI [4] インタフェースの標準化とその参照実装の開発を進めている。

上記のような L1 および L2 の帯域保証サービスを利用できない商用インターネット網を利用せざるを得ない場合、データセンタ間の通信はベストエフォートにならざるを得ない。さらに、サービスモデル (a) およびモデル (b) では、データセンタ間トラフィックを L3 パケットにカプセル化するために、両データセンタのゲートウェイ間でトンネルを設定する必要がある。この際、VLAN タグも両データセンタ間で疎通させる必要があるので、L2 フレームを丸ごとカプセル化する。L2 フレームをカプセル化する L2 トンネリングプロトコルとして、STT、VXLAN、NRGRE など様々なプロトコルが提案されているが、ここでは Linux が標準で提供している GRE-tap を用いる。通常の GRE (Generic Routing Encapsulation) では、L3 部以降を IP パケットでカプセル化するが、GRE-tap は L2 部も含めてカプセル化する。以下、GRE-tap を単に GRE と記す。

4.5 データセンタ内ネットワーク資源管理

複数テナントに対応するためには、利用する VLAN ID や IP アドレスが重複しないように、データセンタ内ネットワークの仮想化が必要である。さらにテナント毎に通信性能の影響を隔離するために、帯域制御が必要になる。

ネットワーク仮想化については、菊池ら [10] が提案している GRE トンネリング方式を基にする。そして IaaS 事業者に提供するすべての計算ノードとゲートウェイにおいて、フルメッシュでトンネルの設定を行う。

テナント毎の帯域制御については、データセンタ間帯域の共有またはデータセンタ内帯域の共有が考えられる。前者はデータセンタ間の予約帯域もしくは利用可能帯域に合わせて、データセンタ間トラフィックを制限する。後者はモデル (a) のように単一の物理ネットワークを共有する場合に必要となる。そこで、高精度帯域制御ソフトウェア PSPacer [11] を Open vSwitch から操作できるように拡張し、Iris から各計算ノードの出力帯域をユーザ VM 単位で制御できるようにする。その詳細は論文 [12] で述べる。

4.6 プロトタイプ実装

HaaS サービスモデル (a) への対応に焦点を当て、Iris のプロトタイプ実装を開発した。計算ノードの仮想化技術として、Nested KVM と LXC の両者に対応した。データセンタ間ネットワーク制御を実現するためにゲートウェイを実装した。現時点では、Web サービスインタフェースとデータセンタ内ネットワーク制御は未実装である。そのため HaaS システムの制御は Web サービス経由ではなく、コマンドラインからスクリプトを実行して制御した。また、データセンタ内ネットワークの隔離ができないので、複数の IaaS 事業者に資源を提供できないという制限がある。

ゲートウェイは、Open vSwitch バージョン 1.4.0 [13] と GRE プロトコルを用いて、IaaS データセンタと HaaS データセンタ間を L2 で接続する。Open vSwitch は、データパスなど一部はカーネルモジュールとして実装されているが、基本的にはユーザランド上に実装されたソフトウェアスイッチである。GRE の実装には Linux カーネルの実装と Open vSwitch に含まれるユーザレベル実装があり、どちらも Open vSwitch から利用できる。プロトタイプ実装では、設定の簡便性から、ユーザレベル実装を利用した。

IaaS 側のゲートウェイのネットワーク設定例を**図 5** に示す。1〜5 行目でブリッジ `br0` を作成し、`eth0` に設定されていたアドレスを付け替えている。6〜7 行目でブリッジのポートに `eth0` と `gre0` を接続する。8 行目が GRE の設定である。HaaS 側のゲートウェイも同様の設定をすればよい。ここまでで IaaS と HaaS のゲートウェイ間でトンネルが設定できる。12 行目以降の設定は通常は不要であるが、CloudStack で用いる NFS サーバや NAT をゲートウェイと同じ計算機上で実行する場合に必要となる必要である。ここでは管理ネットワークとストレージネットワーク用に VLAN 101、パブリックネットワーク用に VLAN 102 を用いており、それぞれに IP アドレスを設定している。

5. 実験

Iris のプロトタイプ実装を用いて、HaaS サービスモデル (a) の実現性を実証するために実験を行った。具体的

ⓒ 2013 Information Processing Society of Japan

表 1 ユーザ VM の諸元

	HaaS UVM (L)				HaaS UVM (K)			
	CPU	mem	disk	NIC	CPU	mem	disk	NIC
L_2	1	512 MB	5 GB (qcow2)	virtio_net	1	512 MB	5 GB (qcow2)	virtio_net
L_1	4	nolimit	chroot	veth	2	6 GB	32 GB (qcow2)	virtio_net
L_0	4	8 GB	640 GB	e1000e	4	8 GB	640 GB	e1000e

```
ip addr del 10.1.1.1/32 dev eth0
ip link set eth0 promisc on
ovs-vsctl add-br br0
ip addr add 10.1.1.1/16 dev br0
ip link set br0 up
ovs-vsctl add-port br0 eth0
ovs-vsctl add-port br0 gre0
ovs-vsctl set interface gre0 type=gre \
  options:local_ip=<Self public IP address> \
  options:remote_ip=<Peer public IP address>

# for management/storage network
ovs-vsctl add-port br0 eth0.101 tag=101 \
  -- set interface eth0.101 type=internal
ip addr add 192.168.101.201/24 dev eth0.101
ip link set eth0.101 up

# for public network
ovs-vsctl add-port br0 eth0.102 tag=102 \
  -- set interface eth0.102 type=internal
ip addr add 192.168.102.201/24 dev eth0.102
ip link set eth0.102 up
```

図 5 IaaS 側のゲートウェイの設定例

には、IaaS 事業者は CloudStack を運用し、Iris を用いて同一ゾーン内のクラスタに計算ノードを追加するシナリオを考える。以降、L_0 を BM (Bare Metal)、L_1 がユーザ VM の場合を IaaS UVM、L_2 がユーザ VM の場合を HaaS UVM (L_1) と記す。後者において、L_1 が LXC の場合は HaaS UVM (L)、KVM の場合は HaaS UVM (K) と記す。

5.1 実験環境

二つのデータセンタが存在するインタークラウド環境を模擬した実験環境（**図 6**）を準備した。左側のクラスタが HaaS 事業者のデータセンタ、右側のクラスタが IaaS 事業者のデータセンタとする。Iris ゲートウェイは head0 および head3 で、CloudStack 管理サーバは head3 で実行し、残り 4 台の計算ノード (hostX) にユーザ VM をデプロイする。実験機材の制約から IaaS 側のゲートウェイと CloudStack の管理サーバを同じ計算機上（図中の head3）で実行したが、本来は 4.1 節で述べたように、異なる計算機上で実行することを想定している。

各 L_0 計算機は Intel Core 2 Quad-core Q9550 2.83 GHz CPU、メモリ 8 GB、複数のギガビットイーサネット NIC を有す。L_0 ホスト OS として Ubuntu Linux 12.04 を用いた。CloudStack はバージョン 4.0.0-bata を用い、ゲストネットワーク間で VLAN による隔離が可能な拡張ゾーンを構築した。なお、CloudStack は一つの物理ネットワークだけを制御し、ゲストネットワークを含めて、管理ネットワーク、パブリックネットワークも VLAN により隔離した。ユーザ VM のインスタンスサイズは Small を用いた。その諸元を**表 1** にまとめる。L_1 ゲスト OS として Ubuntu Linux 12.04、L_2 ゲスト OS として CentOS 5.5 を用いた。

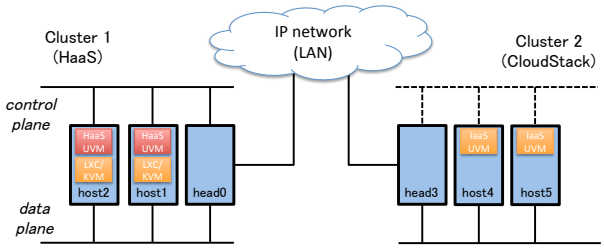


図 6 実験環境

表 2 ユーザ VM のデプロイ時間 [秒]

IaaS UVM	HaaS UVM (L)	HaaS UVM (K)
15.81	15.82	15.96

5.2 ユーザ VM のデプロイ時間

CloudStack の管理ポータルから、ユーザ VM をデプロイするのに掛かる時間を計測した。時間は CloudStack のログを基に、3 回実行した際の平均時間を求めた。なお、デプロイ時間には OS の起動時間は含まない。結果を**表 2** に示す。なお、ゾーン内に初めてユーザ VM をデプロイする場合、まず各種システム VM が作成されるので、デプロイ時間は長くなる。この影響を排除するために、測定対象外の計算ノードでダミーのユーザ VM を起動した後、測定対象の計算ノードでのユーザ VM の起動時間を測定した。以下すべての測定において、システム VM は host4 で動作している。

表 2 に示す結果から、HaaS でも IaaS と同等の時間でユーザ VM をデプロイできている。ただし実環境では、データセンタ間ネットワークには数ミリから数 10 ミリ秒の往復遅延があるので、HaaS へのユーザ VM のデプロイ時間は長くなる。しかし、処理全体に占める遅延の影響は限定的だと推測する。

さらにデプロイしたユーザ VM が、管理ポータルからの操作によって、両データセンタ間で正常にライブマイグレーションできることを確認した。

ⓒ 2013 Information Processing Society of Japan

5.3 Nested KVM と LXC の性能比較

L_1 VM 実装技術を検討するために、Nested KVM と LXC の性能を比較した。ベンチマークプログラムとして、演算性能やファイル IO やシステムコールなどの OS 処理の性能を調べるために BYTE UNIX benchmark バージョン 5.1.3 [14] を、ネットワーク IO 性能を調べるために ping コマンドと Iperf バージョン 2.0.5 を用いた。

BYTE UNIX benchmark の結果を**表 3** に示す。値はベースライン性能に対する向上率を示した指標であり、その値が大きい方が性能がよい。IaaS UVM と比較して、HaaS UVM (L) はすべてのベンチマークにおいて、同等の性能が得られており、LXC による性能オーバヘッドは無視できることがわかる。一方、HaaS UVM (K) ではベンチマークによって傾向が分かれている。演算、ファイル IO、システムコールの性能は 15%程度低下しているが、プロセスの生成、コンテキストスイッチ、シェルスクリプトの性能が 90%と著しく低下している。この原因については 6 節にて議論する。

ネットワーク性能として、ユーザ VM 間の往復遅延時間 (RTT) とグッドプットを測定した。ユーザ VM 間の RTT を ping コマンドを用いて計測した結果を**表 4** に、Iperf のサーバとクライアントが動作するユーザ VM の組合せを変えながら計測したグッドプットを**表 5** に示す。それぞれ値は 60 秒間の平均時間である。なお、予備実験として 2 台の BM 間の性能を測定したところ、RTT は 0.13 ミリ秒、グッドプットは 941 Mbps であった。

ネットワーク性能に及ぼす要因として、仮想化 I/O と GRE トンネリングによるオーバヘッドの二つが挙げられる。前者には、virtio_net や veth による I/O デバイスの仮想化とソフトウェアスイッチ（ブリッジ）によるオーバヘッドが含まれる。HaaS のユーザ VM では、ゲスト OS から物理ネットワークに至るまで、L_1 VM と L_2 VM のそれぞれにおいてブリッジを経由する。

HaaS UVM (LXC) 間の RTT とグッドプットは共に IaaS UVM 間と遜色ない。したがって、L_1 VM として LXC を用いた場合のオーバヘッドは無視できることが分かる。一方、HaaS UVM (K) は IaaS UVM と比較して、RTT が 4.6 倍に増加、グッドプットが 66%に低下した。HaaS UVM (L) 同士のグッドプットは 934 Mbps だったのに対して、HaaS UVM (K) 同士のグッドプットは 621 Mbps に低下した。RTT も 0.35 ミリ秒に対して、1.66 ミリ秒と増えている。性能低下の原因を分析するために、受信側を BM に変更したところ、HaaS UVM (K) のグッドプットは約 800 Mbps であった。このことから受信処理がボトルネックとなり性能低下が起きたことが分かる。

IaaS UVM 間の通信、または HaaS UVM 間の通信は GRE を介さないが、IaaS UVM と HaaS UVM 間の通信では GRE を介す。その結果、HaaS UVM (L) 間の性能が、IaaS UVM と HaaS UVM (L) を上回った。したがって、GRE 処理のオーバヘッドは、RTT で 0.3 ミリ秒増加、グッドプットで 20 Mbps 減少と推測できる。マルチテナント化によりデータセンタ内ネットワークでも GRE トンネリングする場合は、HaaS UVM 間の通信でも上記のオーバヘッドが加わることになる。

表 4 ユーザ VM 間の往復遅延時間 [ミリ秒]

	IaaS UVM	HaaS UVM (L)	HaaS UVM (K)
IaaS UVM	0.36	0.64	1.22
HaaS UVM (L)	-	0.35	0.87
HaaS UVM (K)	-	-	1.66

表 5 ユーザ VM 間のグッドプット [Mbps]

server \ client	IaaS UVM	HaaS UVM (L)	HaaS UVM (K)
IaaS UVM	934	913	806
HaaS UVM (L)	913	934	872
HaaS UVM (K)	635	647	621

6. 議論

6.1 LXC 実装の問題点

LXC は OS カーネルレベルで各資源の名前空間を切り替えることによって、テナント間の隔離を実現している。しかし、その名前空間の分離が不十分であるため次のような問題に直面した。いずれも ad hoc な回避策は存在するが、根本的な修正が必要と考える。

- LXC コンテナ (L_1 VM) 上で KVM を利用する場合、コンテナ内の/usr/sbin/libvirtd のグループを tty にし、かつ SGID ビットを有効にする必要がある。さもなければ、QEMU の起動に失敗する。これは、疑似端末デバイスの不具合に起因すると考えるが、詳細は調査できていない。
- LXC コンテナ内の NFS クライアントが正常に動作しない。NFS カーネルモジュールに起因する問題であり、NFS の遠隔手続き呼び出しの送信元アドレスが、LXC コンテナではなく LXC ホストのものになることが原因である。
- ホスト OS で Open vSwitch を実行すると、LXC コンテナ上にユーザ VM を追加できない。ユーザ VM 追加時に作成されるブリッジデバイスが、コンテナ側でなく、ホスト側に作成されることが原因である。

6.2 Nested KVM 実装の問題点

Nested KVM は LXC と比較すると新しい技術ではあるが、導入は容易であった。VM 間の隔離が堅固である一方で、5.3 節に示すように性能オーバヘッドが大きい。さらに、BYTE UNIX benchmark 実行中に、プロセスが SEGV

表 3 BYTE UNIX benchmark の結果 [INDEX 値]

Benchmarks	BM	IaaS UVM	HaaS UVM (L)	HaaS UVM (K)
Dhrystone 2 using register variables	2396.9	1317.4	1318.0	1061.1
Double-Precision Whetstone	594.7	582.3	581.6	492.0
Execl Throughput	564.5	194.5	198.2	17.2
File Copy 1024 bufsize 2000 maxblocks	2048.4	1441.2	1425.7	1198.3
File Copy 256 bufsize 500 maxblocks	1448.4	979.9	975.5	824.1
File Copy 4096 bufsize 8000 maxblocks	2095.7	1617.3	1669.6	1357.3
Pipe Throughput	1305.6	770.6	772.4	636.5
Pipe-based Context Switching	315.6	343.1	348.2	41.1
Process Creation	535.3	138.6	141.0	10.5
Shell Scripts (1 concurrent)	1688.0	425.0	434.0	40.9
Shell Scripts (8 concurrent)	4675.7	409.7	419.7	37.3
System Call Overhead	1508.2	616.2	615.7	517.7
System Benchmarks Index Score	1239.4	576.2	581.7	192.7

により異常終了することもあり、実運用に向けて依然課題が残る [*1]。

L_2 VM の性能低下の原因として、VM Exits 処理の増加が挙げられる。1 回の L_2 VM Exits に対して 40〜50 回の L_1 VM Exits が発生するという報告もある [8]。また、BYTE UNIX benchmark によって、プロセス生成やコンテキスト切り替えなど、アドレス空間切替えのオーバヘッドが大きいことが分かった。この原因としても VM Exits 回数の増加が考えられるが、詳しい解析は今後の課題である。現在の実装では、EPT (Extended Page Table) や IOMMU を入れ子できないという問題があるので、ソフトウェア処理によるオーバヘッドが大きい。これらの機能は現在実装が進められているところであるが、これにより VM Exits の 1 回あたりのオーバヘッドと頻度を抑えることが可能であり、性能改善が見込まれる。このように今後の改善も見込まれることから、L_1 VM として KVM を使用することは、長期的には有望であると考える。

6.3 ゲートウェイの実装

ゲートウェイの実装に際して、Linux 標準のブリッジデバイスと Open vSwitch の両者を用いた実装を検討した結果、後者を採用することにした。

まずブリッジデバイスでは、複数の VLAN をトランクすることができず、VLAN ごとに GRE デバイスを作る必要があった。IaaS システムでは通常利用する VLAN ID の範囲を指定できるが、実際にユーザ VM に割り当てられる VLAN ID を前もって予測できない。したがって、範囲内の GRE トンネルすべてをあらかじめ準備しておくか、何からの手段を用いて VLAN ID 割当てを検出する必要があった。一方、Open vSwitch では複数の VLAN を 1 本のトンネルにトランクできるので、設定が容易である。さらにマルチテナント化すると、テナントごとに利用する VLAN ID が重複することが考えられる。このような設定をブリッジデバイスで処理することは不可能である。

今回の実験では機器の関係から、ゲートウェイと NFS や NAT を同一計算機で動かしたが、このような構成は運用上の問題が多い。例えば、何らかの理由で VRVM が HaaS 側にマイグレーションした場合、パブリックネットワークにアクセスできなくなるため、HaaS 側のゲートウェイの設定を修正する必要がある。

7. 関連研究

入れ子型仮想化 (Nested Virtualization) の用途として、仮想マシンモニタのデバッグ・開発用、クラウドサービスのセキュリティ強化、Windows 7 の XP モードなど過去のソフトウェア資産の活用などが考えられる。我々の知る限り、HaaS サービスとして入れ子型仮想化を用いた事例はない。Turtle [8] プロジェクトでは、KVM を用いた入れ子型仮想化技術およびその性能最適化手法を提案している。その成果は KVM のメインストリームにフィードバックされている。一方、xCloud プロジェクト [15] では、Xen を用いた入れ子型仮想化技術 Xen-Blanket を提案しており、Amazon EC2 上に L_2 VM として KVM を起動し、EC2 外部とのライブマイグレーションを実現している。CloudVisor [16] は、VMM も信頼できないという前提に立ち、ユーザ VM のメモリやストレージを暗号化するために、VMM の下位層で動作する。

従来からデータセンタでマルチテナントを実現するために VLAN 技術が用いられてきたが、大規模システムでは VLAN ID 数の制限が問題になっている。この制限を回避するために、エッジ・オーバレイ型とホップ・バイ・ホップ型の仮想ネットワーク技術が提案されている。前者は、計算ノード (エッジ) 間を STT、VXLAN、NRGRE を用いてトンネル接続した上に仮想ネットワークを構築する。

[*1] ホスト OS として Ubuntu 12.10 を用いた場合には、同様の異常終了は観測されていないこともあり、Ubuntu 12.04 には Nested VMX に関係するバグが存在すると考える。

物理ネットワークでは VLAN を使わないので上記の制限はないことに加え、特殊なネットワーク機器は必要ない。例えば、Nicira NPV [17] は各計算ノードに Open vSwitch を配置し、その間を STT を用いてフルメッシュのトンネルを設定する。一方、後者として、スライスルーティングスイッチなど、OpenFlow 技術を用いることで、物理ネットワークを仮想的に複数のスライスに分割する方式が提案されている。この場合は、HaaS データセンタ内部をすべて OpenFlow スイッチで構成する必要がある。我々はエッジ・オーバレイ型の GRE 方式を利用する予定であるが、VXLAN 等も実装の成熟を睨みながら検討を進める。

さらに近年、データセンタネットワークの仮想化に関する研究が数多く発表されている。SecondNet [18] は仮想データセンタに対して予約ベースの帯域保証を提供している。これは自由な仮想トポロジを組めるが、Non-work-conserving 型なので帯域の利用率が低下する欠点がある。一方、Seawall [19] は Work-conserving 型かつ TCP のような Min-Max 公平を実現することで高い帯域利用率を実現している。本研究では、L_1 VM レベルで、データセンタ内ネットワークとデータセンタ間ネットワークの帯域制御を一括して制御することを目論んでいる。

8. まとめと今後の課題

本論文では、IaaS 事業者からの要求に対して、ネットワーク越しにハードウェア資源を貸し出す HaaS サービスモデルを提案した。そして本モデルを実現する HaaS 資源管理システム Iris を設計し、その設計の妥当性を検証するためにプロトタイプ実装を開発した。プロトタイム実装では、1 台以上の計算ノードを IaaS データセンタの L2 ネットワークに接続するサービスモデルを提供するために最低限必要な機能を実装した。Iris は計算機とネットワークを仮想化するが、前者として LXC もしくは Nested KVM、後者として Open vSwitch および GRE トンネルを利用した。インタークラウド環境を模擬した実験から、(1) 既存 IaaS システムに変更を加えることなく、ゲートウェイを設定するだけで、HaaS の資源を透過的に利用できること、(2) 2 段階の仮想化によりユーザ VM から HaaS の制御ネットワークを隠蔽できることを示した。さらに (3) HaaS システム上へのユーザ VM のデプロイ時間は、IaaS システム上と遜色ないことが分かった。

今後の課題として、マルチテナント対応、未実装のサービスモデルへの対応、実ネットワーク環境上での実証実験が挙げられる。また、今回得られた知見を基に、資源コーディネータや各種資源マネージャに対する Web サービスインタフェースの設計と実装を進める。この際、SDN (Sofeware Defined Network) 関連技術として議論されている northbound API の動向も睨みながら設計を進める。

参考文献

[1] GridARS: http://www.g-lambda.net/gridars/.
[2] Takefusa, A., Nakada, H., Takano, R., Yanagita, S., Ohkubo, K., Kudoh, T. and Tanaka, Y.: Resource Management Framework for Multi-Domain Cloud, *IEICE Transactions on Communications*, Vol. E94-B, No. 10, pp. 1332–1340 (2011).
[3] Takano, R., Nakada, H., Takefusa, A. and Kudoh, T.: A Distributed Application Execution System for an Infrastructure with Dynamically Configured Networks, *Proc. of NetCloud 2012*, pp. 675–681 (2012).
[4] Open Grid Forum (OGF) Network Service Interface (NSI) Working Group: http://forge.gridforum.org/sf/projects/nsi-wg.
[5] Ben-Yehuda, O. A., Ben-Yehuda, M., Schuster, A. and Tsafrir, D.: Deconstructing Amazon EC2 Spot Instance Pricing, *Proc. of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 304–311 (2011).
[6] CloudStack: http://cloudstack.org/.
[7] G-lambda Project: http://www.g-lambda.net/.
[8] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O. and Yassour, B.-A.: The turtles project: design and implementation of nested virtualization, *Proc. of the 9th USENIX conference on Operating systems design and implementation (OSDI)*, pp. 1–6 (2010).
[9] Linux Containers: http://lxc.sourceforge.net/.
[10] 菊池俊介，今井祐二，福井恵右，小田部繁：クラウドデータセンターにおけるネットワーク仮想化方式（L2 トンネリング方式）の提案と評価，電子情報通信学会技術研究報告 CPSY2010-15，pp. 43–48 (2010).
[11] 高野了成，工藤知宏，児玉祐悦，松田元彦，石川　裕，岡崎史裕：ギャップパケットを用いたソフトウェアによる精密ペーシング方式，情報処理学会論文誌， Vol. 47, No. SIG 7 (ACS 14), pp. 194–206 (2006).
[12] 高野了成，岡崎史裕，工藤知宏：トラフィックの性質に基づいた適応型トラフィック制御手法，電子情報通信学会技術研究報告 NS2012-392 (2013 発表予定).
[13] Open vSwitch: http://openvswitch.org/.
[14] BYTE UNIX benchmark: https://code.google.com/p/byte-unixbench/.
[15] Williams, D., Jamjoom, H. and Weatherspoon, H.: The Xen-Blanket: Virtualize Once, Run Everywhere, *Proc. of the ACM European Conference on Computer Systems (EuroSys)* (2012).
[16] Zhang, F., Chen, J., Chen, H. and Zang, B.: CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization, *Proc. of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, pp. 203–216 (2011).
[17] Nicira Network Virtualization Platform (NVP): http://nicira.com/en/network-virtualization-platform.
[18] Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., Wu, W. and Zhang, Y.: SecondNet: a Data Center Network Virtualization Architecture with Bandwidth Guarantees, *Proc. of the 6th International Conference on emerging Networking EXperiments and Technologies (CoNext)*, pp. 15:1–15:12 (2010).
[19] Shieh, A., Kandula, S., Greenberg, A. and Kim, C.: Sharing the Data Center Network, *In Pro. of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2011).

ⓒ 2013 Information Processing Society of Japan