WIDE クラウド WG 2013 年度活動報告

島 慶一　　中村 遼

2014 年 1 月 22 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。

2013 年度の主な活動は以下の通りである。

- クラウド環境のネットワーク技術 (第 2 章)
- 仮想ディスクイメージ用ストレージ技術 (第 3 章)
- 仮想計算機モニター MIB (第 4 章)

以降、それぞれの活動の詳細を報告する。

2　SDN によるクラウドネットワーク

2.1　背景

仮想化技術の発展によって登場した IaaS クラウドは、現在多くのサービスのインフラとして利用されている。IaaS クラウドによる仮想化されたサーバ資源の利用によって、サーバ環境の構成は非常に柔軟なった。しかし一方で、現在仮想マシン (VM) を含むネットワークそのものに対する要求が増加している。サービスは、サーバだけでは実現できない。必ずサーバ間や外部への接続性を提供するネットワークが必要である。現在の IT インフラ環境におけるネットワークに対する要求は多岐に渡り、それらを実現するために多くの機能が存在する。こういったネットワークとその機能は、物理環境か仮想環境かに関わらず、必ず必要なものである。このように、IaaS クラウド上の仮想環境では、VM だけでなく、ユーザの要求に応じてネットワークの機能を提供する、仮想ネットワークが必要である。

この仮想ネットワークにおいて実現しなければならない機能は多岐にわたる。IaaS クラウドにおいて VM の数やその性能をユーザが要求に応じて選択、変更するように、VM を収容する仮想ネットワークの要件は、ユーザが VM 上で稼働させるサービスの種類や構成によって変化する。たとえば、外部に対して公開するサービスなのか、内部のユーザのみで利用する機密性の高いものなのかによって、アクセス制御や Firewall の構成は変化する。このように、ユーザごとに要求が異なるため、IaaS クラウド上において構築される仮想ネットワークは、必要な機能を提供すると同時に、ユーザごとに分離されたネットワークである必要がある。IETF のオーバーレイを用いた仮想ネットワークを議論する NVO3 ワーキンググループでは、このユーザごとのネットワーク空間の分離を仮想ネットワークの要件としてあげている [1]。

ユーザごとに分離された仮想ネットワークを提供する場合、そのネットワークの運用が問題となる。ユーザごとに要求や構成の異なるネットワークをクラウド事業者が 1 つ 1 つ設定するのは現実的ではない。そこで、仮想ネットワークを動的に構築すること、そしてユーザが必要な機能を必要なときに変更できるような IaaS 自体の構成が必要になる。このような要求に対して、近年ネットワークを動的に操作するための 1 つの概念である、Software Defined Network が注目されている。SDN では、ネットワークの機能をソフトウェアで抽象化、実現することによって、ネットワーク自体のプログラミングを可能にする。

そこで本研究では、この SDN を用いて、IaaS クラウドにおける仮想ネットワークの柔軟な構築手法を提案する。仮想ネットワークに必要な機能と IP ルーティ

ングの機能を、VM として動作するルータである仮想ルータによって実現する。また HV 間や外部ネットワークへの接続といった L2 の機能を OpenFlow によって実現する。ソフトウェアによる制御が可能な仮想ルータと OpenFlow を用いることによって、IaaS クラウドにおける仮想ネットワークの構築の自動化を行う。また、ユーザごとに仮想ルータを構築することで、仮想ネットワークの分離を実現する。

本研究では、この提案手法を 2013 年の INTEROP Tokyo にて実施された ShowNet 内において適用し、実際の出展者収容ネットワークの構築を行った。

2.2 目的

本研究では、IaaS クラウド上での仮想ネットワークの構築とその自動化を目的とする。IaaS クラウドにおいても、VM 間を接続しサービスを実現するためにネットワークの機能が必要である。またそのネットワークはユーザごとに分離されなければならない。本研究では、仮想環境におけるネットワークの機能と構成を、仮想ルータと OpenFlow を用いて抽象化し、ソフトウェアで制御する運用手法を提案する。本運用手法によって、2.1 章で述べた、ユーザごとの様々な機能を提供する仮想ネットワークの構築の自動化を実現する。

2.3 関連研究

IaaS クラウドにおける柔軟なユーザネットワークの構築については、多くの事例や先攻研究がある。それぞれに解決する部分があり、IaaS クラウドにおける仮想ネットワークを構築するための手法や要素技術を提案している。

Amazon の提供する IaaS サービスである Amazon EC2 では、Amazon VPC[2] を用いてユーザが Web インターフェースからセグメントの設計やルーティングの設定を行うことが可能である。しかし、Amazon VPC を用いて設定できることは、Amazon VPC の提供する機能のみであり、ユーザが自由に機能を追加したりすることはできない。また、VPC 側の制約によって、利用できる機能や規模に制限が存在する。CloudNaaS[3] では、ユーザが記述したポリシーに従って、IaaS 上にユーザのネットワークが構築される。ポリシーの中には抽象化されたネットワークの要素が記述可能であり、セグメント設計やミドルボックス、QoS などが記述できる。この場合も、Amazon VPC と同様に、ポリシーで記述できる機能しかユーザが利用することはできず、ユーザが新しく機能を追加することは難しい。

また分離された仮想ネットワークを構築するための手法も存在する。従来のネットワークでは VLAN を用いてネットワークを識別する方法が一般的であるる。一方で VLAN には、その VLAN の通る全てのスイッチを、ホップバイホップ方式に設定変更する必要があることや、識別子のサイズによる VLAN 数の制限が存在する。そこで、イーサネットフレームを IP でカプセル化する VXLAN[4] などのトンネリング手法がある。一方 OpenFlow ネットワーク上でネットワークの分離を行うものもある [5]。これらの技術は、L2 ネットワークを論理的に分割する技術であり、IaaS クラウドにおける仮想ネットワークのセグメントの分割にも用いることができる。しかし、あくまでセグメントの分離であるため、L3 以上の機能である NAT や Firewall、VPN といった機能は別の手法で用意する必要がある。

2.4 設計

IaaS クラウド上に仮想ネットワークを構築し、ユーザに提供するための機能要件を以下に示す。

- ユーザへのネットワーク機能の提供
- ユーザ毎に分離された仮想ネットワークの構築
- 仮想ネットワークの自動構築

ユーザが IaaS クラウド上で利用したい機能は多岐わたる。そこでまず仮想ネットワークとしてそれらの機能を提供できる環境を用意しなければらない。そして、この仮想ネットワークはトポロジ、そして ID 空間がユーザごとに分離されていなければらない。また、ユーザごとの仮想ネットワークを IaaS 事業者が手で設定するのは現実的でないため、これを自動で構築する必要がある。

そこで本研究では、仮想ルータと OpenFlow を用いた運用手法を提案する。図 1 に提案する仮想ネットワークの概要を示す。仮想ルータとは、既存のネットワークの機能を実現する物理ルータの OS ソフトウェアを、HV 上で動作する VM として提供したものである。この仮想ルータをユーザごとに提供することで、ユーザ

は仮想ルータの機能を自由に利用することができる。さらに、IaaS クラウドにおける物理ネットワーク上で仮想ネットワークを実現する手法として、OpenFlow を用いる。OpenFlow を用いることによって、物理スイッチ上の仮想ネットワークの分離と、プログラムによる構築の自動化を実現する。

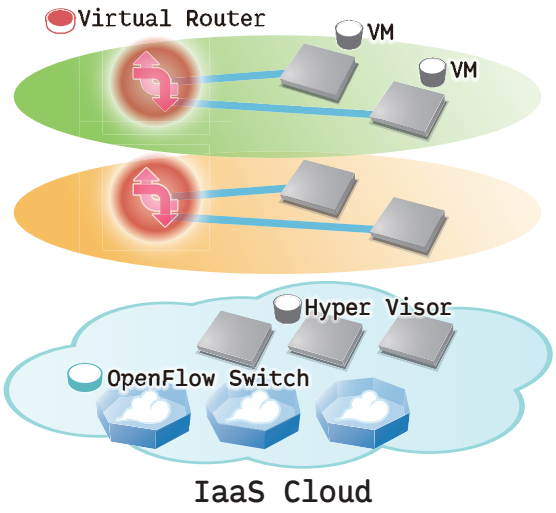

図 1: 構築するネットワーク概要

2.4.1 仮想ルータ

仮想ルータは、今まで専用機器上で動作していたルータの OS を、VM として動作させるものである。汎用サーバの高度化と、VM やソフトウェアによるパケット転送の高速化によって、仮想ルータでも十分な性能を引き出すことが可能となった。さらに仮想ルータでは、現在にいたるまで物理筐体のルータに実装された多くのネットワーク機能を利用することが可能である。また、それらの機能の設定方法などは従来と同様であり、今までの運用手法や知見をそのまま仮想ルータに適用することができる。既にいくつかの仮想ルータが提供されており、ユーザが用途に応じて必要な機能を持つ仮想ルータを選択し、VM として利用することができる。さらに、VM として動作するため、仮想ルータの構成、たとえば CPU の数や NIC の数、その NIC に所属させるネットワークの設定などは、HV 側からの制御が可能である。つまり HV の API やコマンドを利用することによって、仮想ルータの構成変更や起動などのプログラムによる動的な制御が可能になる。

2.4.2 OpenFlow

OpenFlow は、スイッチのパケット転送部を制御するための共通化された API と、それを伝達するためのプロトコルである。OpenFlow スイッチは、コントローラから OpenFlow プロトコルを用いて、パケットのマッチフィールドとアクションからなる Flow エントリを記憶する。この Flow エントリに従ってパケットを転送する。OpenFlow では、スイッチ自身が経路を計算することなく、全てをコントローラから指示することで、コントローラからネットワークを自由に制御することができる。2.3 章で述べた通り、物理ネットワーク上に OpenFlow を用いて分割された L2 ネットワークを複数構築することは可能である。本研究では、この OpenFlow を VM や仮想ルータを収容する HV 間、及び外部への接続するための L2 ネットワークに用いる。OpenFlow を用いることによって、ユーザ毎の仮想ネットワークの L2 網を必要に応じて動的に追加、削除を行うことができる。

2.4.3 提案手法

本研究では、仮想ネットワークのテンプレートを作成し、L3 以上の機能を仮想ルータで、L2 以下の機能を OpenFlow を用いてインスタンスとなる仮想ネットワークを作成する。1 つのユーザのための仮想ネットワークを作るためのフローを下記に示す。

1. テンプレートから仮想ルータの設定を生成
2. 仮想ルータを HV 上にデプロイ
3. HV 内で仮想ルータのインターフェースを適切なネットワークに接続
4. OpenFlow を用いて HV から集約ルータや他の HV への L2 接続を作成

仮想ルータの設定は、ユーザ情報に依存しない部分を共通化することができる。例えばルータ自身へのアクセス制御や、集約ルータへのルーティングの設定、統計情報の取得などである。こういった共通する部分の設定をテンプレート化する。新しく仮想ネットワー

クを作成する際は、ユーザに固有の情報である IP アドレスやホスト名をテンプレートに埋め込むことで、そのユーザ専用の仮想ルータを生成する。この仮想ルータを HV 上にデプロイし、HV 内で仮想ルータの NIC にそのユーザの L2 セグメントをアタッチする。

次に、OpenFlow スイッチによって構成される IaaS の物理網に対して、ユーザの L2 セグメントを、仮想ルータや VM の存在する HV、集約ルータへと延ばすよう設定する。これによって、1 つのユーザのための仮想ネットワークが構築される。この手順は、テンプレートの事前作成を除けば、全ての作業をソフトウェアによって自動化することができる。

また、ユーザは要求に応じて仮想ルータの機能を自由に利用することができる。仮想ルータはユーザごとに作成されるため、自身の仮想ルータの構成変更を行っても、他のユーザの仮想ネットワークに影響を与えることは無い。このように、仮想ルータと OpenFlow を用いることによって、既存のネットワークの機能をユーザに提供しつつ、ユーザごとの仮想ネットワークの分離と、その仮想ネットワークの自動生成が可能となる。

2.5 ShowNet における運用

本研究では、提案手法を INTEROP Tokyo[6] のネットワークである ShowNet[7] にて適用し、実際に構築と運用を行った。INTEROP Tokyo は、毎年 6 月に開催される、日本最大規模のネットワーク機器と技術に関する展示会である。この INTEROP Tokyo において構築される ShowNet は、世界最大のデモンストレーションネットワークである。ShowNet は、企業からコントリビューションされたその時々の最新機器の相互接続性検証のためのネットワークであると同時に、将来のネットワーク構成を示すデモであり、また展示会出展者へネットワーク接続性を提供する ISP ネットワークとしての側面を持つ。

本研究では、2013 年の ShowNet においてコントリビューションされた仮想ルータ、OpenFlow スイッチ及びコントローラ、HV となるサーバを用いて、提案手法を実装し、展示会出展者のブースを収容するネットワークの 1 つとして運用を行った。表 1 に実際に使用した機器を示す。ShowNet では、提案手法を用いて出展者毎に仮想ルータを提供し、VM の替わりに出展者への接続性を提供した。この際の構成を図 2 に示す。各出展者の IP アドレスやブース名、VLAN 番号などの情報は ShowNet の出展者データベースに保存される。この情報をもとに 1 つの出展者のための仮想ネットワークを構築する手順を以下に示す。

表 1: 使用した機器

種類	機器名
仮想ルータ	SoftAX
仮想ルータ	CSR1000V
仮想ルータ	JUNOSv Firefly
OpenFlow スイッチ	PF5240
OpenFlow スイッチ	PF5248
OpenFlow スイッチ	PF5220
OpenFlow コントローラ	PF6800
HV サーバ	R320
HV サーバ	R420
HV サーバ	R610
HV サーバ	C6100
HV ソフトウェア	VMware ESX

まず、ユーザごとに固有の情報であり、テンプレートで変数化されている IP アドレスや VLAN 番号等の情報を、出展者データベースから取り出す。これらの情報をもとに、事前に作成した仮想ルータのテンプレートから、実体となる仮想ルータを生成する。仮想ルータのテンプレートとは、事前に作成した仮想ルータの OVA ファイルと、Jinja2[8] を用いて作成した設定ファイルである。仮想ルータは OVA ファイルから VMware ovftool を用いて HV へデプロイされる。そして、生成した設定ファイルを仮想ルータに読み込ませることで、仮想ルータのデプロイが完了する。次に、仮想ルータの NIC を適切なネットワークにアタッチする。仮想ルータは、上位の集約ルータと接続するためのセグメントと、出展者を収容するユーザセグメント、そしてコンソールログインなどを行うための管理セグメントへの 3 つ NIC をもつ。これらの NIC を、VMware ESX の CLI を用いて外部から操作し、それぞれのネットワークへと接続した。

次に、仮想ルータの存在する HV から出展者収容ポート及び集約ルータへの L2 接続を構築する。本構成での L2 ネットワークは、NEC Programmable Flow による OpenFlow 網で構築した。図 2 の中段の pf5248-1 と

pf5240-1 が仮想ルータを動作させる HV 群を収容し、最下段の pf5240-2 及び pf5248-2 の 2 台が出展者ブースへの物理接続を収容した。まず、ユーザがどのポートに収容されているかの情報を出展者データベースから取り出す。そのポート情報にしたがって、OpenFlow コントローラである PF6800 の提供する REST API を用いて、出展者ネットワークの VLAN を OpenFlow スイッチの物理ポートに加え、仮想ルータの存在する HV へのポートに VLAN を加える。このようにして、出展者ネットワークの L2 接続を構築した。

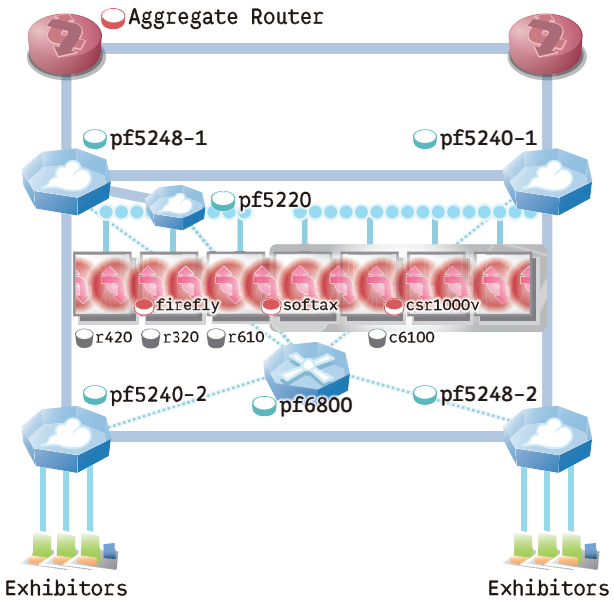

図 2: ShowNet において構築したネットワークの構成

ShowNet において提案手法を用いて構築したネットワークでは、上記の手順によって、出展者ネットワークの構築を完全に自動化することができた。出展者データベースからの情報の取り出し、テンプレートからの設定の生成、仮想ルータのデプロイ、OpenFlow コントローラの REST API を経由した L2 網の設定、という一連の作業は、全てソフトウェアによって自動的に実行された。以上のように、本提案手法によって、ShowNet における出展者ネットワークの構築の自動化を実現した。この本提案手法で構築したネットワークは、INTEORP Tokyo の展示会期間において安定して動作した。これによって、提案手法が実現可能であることを示した。

2.6 結論と課題

本研究では、IaaS クラウド上の仮想ネットワークの構築とその自動化を目的として、仮想ルータと OpenFlow を用いたユーザごとの仮想ネットワークの構成と、その構築手法の提案を行った。仮想ルータを用いることにより、ユーザは従来のネットワークの機能を必要に応じて利用することができる。また、仮想ルータをユーザごとに用意することによって、ユーザの仮想ネットワークの分離を行った。IaaS 環境における L2 ネットワークの構築に OpenFlow を用いることによって、HV 間の L2 ネットワークの分離と自動構築を行った。これらソフトウェアによる制御が可能な仮想ルータと OpenFlow を用いることで、ユーザごとに仮想ネットワークを自動的に構築する運用手法を示した。

また本研究では、提案手法を 2013 年の INTEROP Tokyo にて開催された ShowNet において適用し、実際に構築と運用を行った。ShowNet の出展者データベースの情報をもとに、複数種類の仮想ルータとそれを動作させる HV の API、そして OpenFlow コントローラの提供する REST API を用いることによって、出展者ネットワークのソフトウェアによる自動生成を実現した。これにより、本提案手法が実際のネットワーク構築と運用において実現可能であり、また IaaS クラウドにおける仮想ネットワーク構築にかかるコストを削減できることを示した。

2.7 今後の課題

今後の課題として、まずパフォーマンスの計測が必要である。仮想ルータによるパケット転送は、同一 HV 上の仮想ルータが多くなる程性能が落ちる。その際の性能の低下は、HV 上で通常の VM が動作する場合とは異なることが予想される。そのため、仮想ルータをデプロイする際のボトルネックと、単一の HV 上で実効的に動作する仮想ルータの数を、詳細に評価する必要がある。

運用面の課題としては、仮想ネットワークの構成の問題がある。仮想ネットワークにおいて、ユーザは自由にサブネットを追加し、トポロジを構築できる必要がある。しかし、ShowNet にて構築した構成では、仮想ネットワークのテンプレートは 1 ユーザに 1 仮想ルータという構成のみであった。そのため、今後はテンプ

レートを複数用意することや、テンプレートから作成した仮想ネットワークにユーザが自由にルータの追加やセグメントの追加を行える様にする必要がある。

3 UKAI: Location Aware Distributed Local Storage for Virtual Machine Disk Images

3.1 Introduction

Virtual machine operation has become one of the core tasks of successful service providers. One problem still to be solved in this area is that of resource migration. For instance, it is sometimes necessary to perform actions such as move a virtual machine from one hypervisor to another, to decrease the load of the origin hypervisor, upgrade a hypervisor operating system, move virtual machines to a newly built datacenter, or discontinue an existing datacenter. A virtual machine basically consists of three parts: i) CPU and memory resources, ii) a network resource, and iii) a storage resource. To keep a virtual machine running after migration, an operator must transparently supply the above three resources to the virtual machine. For i), the recent major virtualization technologies [9, 10, 11, 12, 13] already have the ability to migrate CPU and memory resources. For ii), recent progress in software defined networking technology means that is now a candidate solution for network resource migration [14]. For iii), there are a few technologies to support storage migration [15, 16, 17].

This paper focuses on the storage resource migration issue. We believe the following three requirements are important when operating a virtual machine storage backend.

Req.1 *Controllablity*: to provide the ability to store a disk image to a specific location for each virtual machine

Req.2 *Availability*: to provide a flexible level of redundancy to avoid data loss

Req.3 *Locality*: to place a disk image as near its owner virtual machine as possible to increase performance and robustness against network failures

Based on the above requirements, we have designed a new storage system that serves virtual machine disk images to hypervisors. The proposed system offers full control over the storage location of disk image data for each virtual machine. An operator can assign multiple locations for the same disk image to increase redundancy and can even specify location information on a portion of a disk image to protect important parts.

The location specification can be added or removed at any time. When a virtual machine moves from one location to another, an operator can add local locations, migrate the existing data to local locations without disrupting machine operations, and remove remote locations after migration. Unlike autonomously distributed file/storage systems, an operator can specify storage location. This is important because network services usually consist of multiple virtual machine instances. How storage access is optimized for one machine may affect the collaborative operational performance of the set of virtual machines.

As might be anticipated, providing full location control could lead to a lot of configuration tasks when the number of virtual disks is large. In our proposal, we do not consider the automated support of location control; however, it is possible to design a super-layer for storage management policy. This higher-layer module may provide automatic location determination functions. To do that, the lower-layer module must have detailed location management functions. This proposal aims to provide the basis for more intelligent management systems in future.

3.2 Related Work

The most basic technique of storage migration is incremental block migration. In this mechanism, the

entire storage image is moved from the source to the destination location. A simple method was proposed in [15] that moves disk image blocks from head to tail after a virtual machine has been moved to a destination location. If the virtual machine tries to access a not-yet moved block, then an on-demand copy is initiated. These mechanisms satisfy requirements 1 and 3. However, since the disk image is stored at single location, requirement 2 is not achieved.

To achieve redundancy, a distributed filesystem is sometimes used. Distributed Replicated Block Device (DRBD) [18] provides a block device, replicated over a network. Since DRBD is a type of mirrored disk, the operations for disk management are similar to those of local disk mirroring operations. In this sense, requirement 1 is satisfied; however, it is not possible to use this system for each virtual machine. DRBD also partially satisfies requirement 2 as it can have up to two replicated disks in a basic configuration. When more than two replicas are needed, a cascading configuration is required. For requirement 3, DRBD allows a dual primary operation mode that enables concurrent access to both mirrored volumes. In this sense, data locality can be achieved. However, it is difficult to change the mirroring volume from a remote site to a local site. Even though a user can access the local mirrored volume, the remote mirrored volume is still located remotely. The reconfiguration operations in DRBD are less flexible.

There are other distributed filesystems such as Ceph [19] or GlusterFS [20]. There is also a distributed block storage mechanism called Sheepdog [21] that is a special block device designed for virtual machine disk images. These distributed systems use a consistent hash mechanism to determine the locations for data distribution. An operator cannot control which part of a disk image is stored on which storage node. This is problematic when an operator wants to move a virtual machine to a distant location. If a storage cluster is configured locally near the origin hypervisor, the migrated virtual machine will suffer from poor disk performance because of the long network delay. If the storage cluster is deployed over a wide area network, the daily disk operation performance will worsen. Hence, these systems only satisfy requirement 2.

3.3 The UKAI System

In this section, we discuss the design of the *UKAI* system[1], a centrally controllable distributed local storage system.

3.4 Constraints and Advantages

When designing a new storage system that satisfies the requirements defined in Section 3.1, we must first clarify the constraints of a virtual machine operation mechanism. We believe the operational environment of a virtual machine is unique, as a storage interface is not always required to provide a fully functional distributed filesystem. The constraints are given below.

Con.1 No concurrency: there is only one entity for accessing a specific disk image at one time.

Con.2 Limited metadata elements: the metadata information of a disk image is limited.

Since we are designing a storage system for virtual machine disk images, we can assume the disk images will only be used by hypervisors and virtual machines. Considering that a disk image is associated with a specific virtual machine, it is not necessary to allow concurrent access to a specific disk image from multiple entities. This means that it is not necessary to design a distributed resource locking mechanism.

A disk image is seen as a block device from a virtual machine's point of view. Creation, access, or modification times are not meaningful in such an environment. The size of a disk image is also meaningless, as it does not usually change. Based on these constraints, a UKAI storage can be implemented with only simple data I/O interfaces.

[1] UKAI is named after a traditional Japanese fishing method that uses cormorants (`http://en.wikipedia.org/wiki/Cormorant_fishing`).

3.5 Disk Image Design

In the UKAI system, a disk image is divided into small blocks. Each block has its own location record. If redundancy is required, a block may have multiple location records. A block is the unit of a synchronization operation, and its location record contains a flag that indicates if the block at that location is in-sync or out-of-sync. If the disk image does not have redundancy, then all the locations must be in-sync. Having an out-of-sync location means that the disk image is broken, if a block has only one location record. If there are multiple locations, the operational requirement is to have at least one in-sync location. When reading data from a block, one of the in-sync locations is chosen for data retrieval. When writing, data is transferred to all locations and written to a local storage device at that location. If a location is in the out-of-sync state, data from one of the in-sync blocks is transferred to the out-of-sync location. Once the data transfer completes, the state is changed to in-sync. Fig. 3 shows the concept of the UKAI disk image structure.

In the figure, a disk image is divided into four blocks. Each block has two location records, some of which are flagged as out-of-sync. Data is read from the node indicated by the white location information box. When writing, for example, to block 3, the data stored in node B is copied to node A before any data is written. Then the location of node A is changed to an in-sync state and the actual data is written to both locations.

3.6 Metadata Design

Metadata information for a virtual disk is stored in the hypervisor on which the virtual machine using that disk is run. It consists of the size and name of the disk image, the size of the block, and location information of each block. The size and name are used as a filename when the disk image is exposed as a file through a filesystem or as a block device name of a hypervisor, depending on how the system

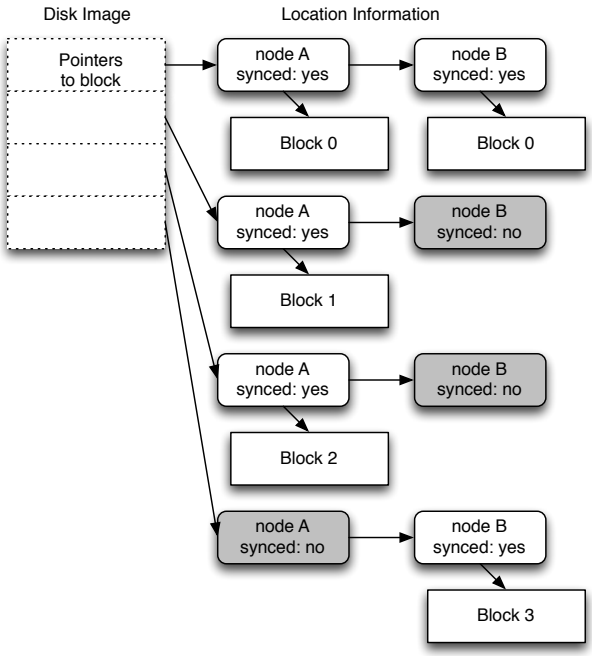

図 3: The concept of the UKAI disk image structure consisting of four blocks and two location records

is implemented. The block size and location records are used in the UKAI system internally.

3.7 Error Handling

When operating a distributed system, an error is not an option. For example, when reading data from one of the nodes defined in the location record, it may not be available because of node failure, network failure, or some other reason. In that case, the UKAI system records the node address in the locally managed failure node list. When accessing a block, the UKAI system first checks the failure node list. If the node is listed in when the UKAI system is writing data, the location information is marked as out-of-sync. The data is written to all the other nodes listed in the location record list. When reading, the UKAI system looks for another candidate from the list of locations of the block being accessed. If there is no in-sync candidate, the situation is considered a fatal error.

The failure node list has a time limit for each en-

try. When this limit expires, access to the node may be resumed. If the node recovers before the time limit expires, then the data synchronizing process will be initiated when a write operation occurs on blocks marked as out-of-sync.

3.8 Control

The following control operations are defined for the minimum operation of the UKAI system.

Add image: adds a new virtual disk image to the UKAI system. An image must be added before being used as a disk image by a virtual machine.
Remove image: removes an existing virtual disk image from the UKAI system. Before removing a disk image, the virtual machine that uses the target disk image must be powered off.
Add location: adds a location specification to a range of blocks.
Remove location: removes a location specification from a range of blocks. When removing a location, the system must ensure that at least one in-sync location is left for each block. Otherwise, the data of the disk will be lost and a fatal error will occur.
Get metadata: returns the metadata information of a specific disk image containing the name of the image, the size of the image, the block size of the image, and the list of locations for all the blocks.
Synchronize: synchronizes a range of blocks between nodes defined in the location records of the specified blocks.

Other operations may be defined in future UKAI systems.

3.9 Implementation

We have implemented the concepts of the UKAI system in a prototype using FUSE [22] and Python. Each disk image is exposed through the FUSE mechanism as a file. We used QEMU as a hypervisor since it has the ability to use a file as a virtual disk image. Any other hypervisors can also be used if they can use a file as a virtual disk image. Note that there is no limitation on how to implement the UKAI system. An implementation as a filesystem interface is just one example. It could also be implemented in a special block driver form for QEMU or other virtualization systems.

The prototype code is available at the GitHub repository[2].

```
{
 "name": "disk01",
 "size": 200000000,
 "block_size": 50000000,
 "blocks": [{
   "192.0.2.100": {"synced": true},
   "192.0.2.101": {"synced": true}
  },{
   "192.0.2.100": {"synced": true},
   "192.0.2.101": {"synced": false}
  },{
   "192.0.2.100": {"synced": true},
   "192.0.2.101": {"synced": false}
  },{
   "192.0.2.100": {"synced": false},
   "192.0.2.101": {"synced": true}
  }]
}
```

図 4: The metadata structure that represents the disk image shown in Fig. 3, where the size of the image is set to 200 MB, the size of each block is set to 50 MB, and the addresses of nodes A and B are set to 192.0.2.100 and 192.0.2.101, respectively

3.10 Metadata Handling

Fig. 4 shows the metadata structure for the example disk image shown in Fig. 3. The data is structured using a JSON [23] format. The name of the disk image is `disk01` and this name is used as a file name under the FUSE mount point of the UKAI system. The size of this disk image is defined to be 200 MB. Since the block size is 50 MB, the total number of blocks will be four blocks.

The metadata file for a virtual disk is stored in the

[2] https://github.com/keiichishima/ukai/

hypervisor that runs its virtual machine. When a virtual machine is migrated to another hypervisor, the related metadata must be accessible from the destination hypervisor. This can be achieved in several ways. One method is to copy the entire metadata file at the last moment when the virtual machine state is migrated to the destination. Since the size of a metadata file is small compared to memory or storage data, copying a metadata file does not affect migration performance. The other method is to use some kind of distributed filesystem, such as GlusterFS, to share metadata files. In the latter case, it is necessary to operate a wide-area distributed filesystem; however, the update of metadata information occurs only when the synchronization status changes, so its influence is small. In the test operations discussed in Section 3.14, we simply share metadata files using NFS. For performance reasons, we plan to integrate the former metadata transition method in the final form of the implementation.

3.11 FUSE Interface

To implement a FUSE interface in Python, we utilized fusepy[3], a FUSE-Python binding library. As we have discussed in Section 3.4, not all the filesystem interfaces must be implemented. In the prototype implementation, only the interfaces shown in TABLE 2 are implemented.

Other filesystem interfaces are either not implemented (such that the call falls back to the default behavior of fusepy), or do nothing.

3.12 Remote Read/Write Operations

In some cases, read or write operations may require access to a remote UKAI node. This is implemented using the XML-RPC mechanism. As we discussed in Section 3.4, we do not need to take care of conflicting accesses to a disk image since only one virtual machine accesses a specific disk image at one time.

[3] `https://github.com/terencehonles/fusepy/`

表 3: Configuration of location information of three UKAI disk images

Location type	Configuration detail
Local	A virtual machine and its disk image are located on the same node.
Remote	A virtual machine and its disk image are located on different nodes.
Mirror	A disk image has two locations: one is on the same node as the virtual machine and the other is on a different node.

When receiving read/write requests from a remote node, the UKAI storage node just performs read or write operations on the local disk without any exclusive control because it is certain that there is no other entities accessing the disk.

3.13 Control Operation

Control operations described in Section 3.8 are also implemented using the XML-RPC mechanism. The RPC interface is open to a local node. An operator can issue management commands such as adding an image or location through this interface.

3.14 Performance Measurement

We measured the I/O bandwidth with our prototype implementation. We prepared three 8 GB UKAI disk images with different block size values (5 MB, 10 MB, and 20 MB) and three different locations as shown in TABLE 3.

For comparison, a disk image was created on local storage as a single file, and a disk image served by NFS was also prepared.

Two physical machines connected with a 1 Gbps Ethernet switch were prepared and configured as hypervisors and UKAI nodes. These nodes were located in the same network segment. One of the nodes was

表 2: FUSE interfaces mandatory for the UKAI system

Interface	Action
`init()`	Initializes the UKAI system. The function launches two threads, one for receiving read/write operation requests from remote UKAI nodes as described in Section 3.12 and the other for receiving control commands as described in Section 3.13.
`getattr()`	Returns a `stat` structure of a disk image file. The total image size is the only meaningful information.
`open()`	Returns a file descriptor of the specified disk image.
`readdir()`	Returns a list of disk image file names. To conform to the normal filesystem `readdir()` operation, the current and parent directories of the UKAI mount point are also returned.
`read()`	Reads data from a virtual disk and returns it to the caller. One read operation may contact multiple blocks depending on the specified read size and offset value.
`write()`	Writes data to a virtual disk. Just as for the read operation, multiple blocks may be accessed depending on the specified write size and offset. The write operation may initiate a synchronization operation if the blocks to be accessed have location information with an out-of-sync status.

表 4: Specifications of the measurement equipments

Nodes	
Product ID	EPSON Endeavor AT971
CPU	Intel® Core™2 Duo E8400 3.00 GHz
Memory	4 GB
HDD	250 GB SATA
NIC	Intel® PRO/1000 Gigabit Network Adapter
Switch	
Product ID	Corega CG-SW05GTLXW

also configured as a NFS server for the NFS disk image mentioned above. TABLE 4 shows the specifications of the equipments.

The virtual machines used for the measurement ran Ubuntu 12.04 LTS with 512 MB memory.

Fig. 5 shows the system diagram of the measurement system. There were five different virtual machine configurations, each using a different disk image configuration. The three that used UKAI disk images also had three different block size configurations as described earlier. TABLE 5 shows all the combination cases of the disks and virtual machines

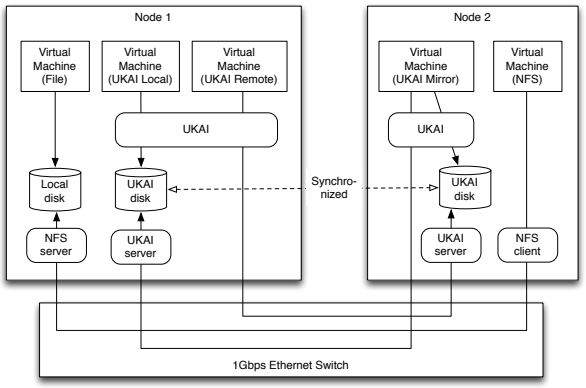

図 5: System diagram of the performance measurement system

I/O pattern	Random read and random write
I/O file size	128 MB, 256 MB, 512 MB, and 1 GB
I/O block size	4 KB
I/O API	Use standard `read()` and `write()` system calls and `fseek()` library call

表 6: Configuration parameters of the fio command

used in the measurement.

Location type	Virtual disk type	Owner virtual machine
Local	Local file	Local
	UKAI 5 MB BS	UKAI Local
	UKAI 10 MB BS	
	UKAI 20 MB BS	
Remote	NFS file	NFS
	UKAI 5 MB BS	UKAI Remote
	UKAI 10 MB BS	
	UKAI 20 MB BS	
Mirror	UKAI 5 MB BS	UKAI Mirror
	UKAI 10 MB BS	
	UKAI 20 MB BS	

表 5: Mapping table of virtual disk location types, virtual disk configuration types, and their owner virtual machines

We used the fio[4] measurement tool. The configuration parameters for fio are shown in TABLE 6. Measurement operations were done one-by-one for each virtual machine. While one virtual machine was running, the other four machines were shut down. As shown in TABLE 6, four different file sizes, 128 MB, 256 MB, 512 MB, and 1 GB were used in the experiment to compare the effect of files size on performance variation.

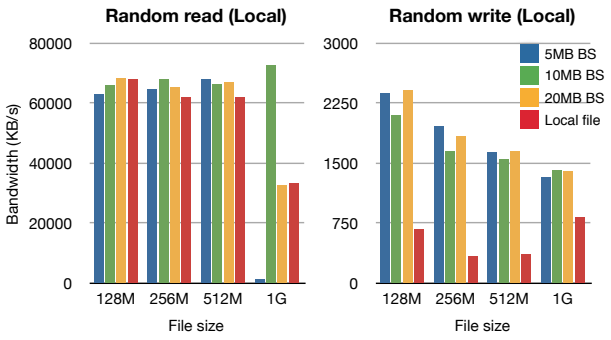

図 6: Comparison of I/O bandwidth of local disk images

3.15 Local Storage

Fig. 6 shows the results for local storage cases. The graph shows four different types of local disk: the first three are UKAI images with different block sizes and the fourth is a local file disk image.

In the random read case, we see unstable behavior for UKAI disks of 5 MB and 10 MB block sizes in the 1 GB file test case. However, the overall performance is not worse than that of the local file storage method. For the random write case, the UKAI disks often achieved better bandwidth than the local file storage method.

3.16 Remote Storage

Fig. 7 shows the results for remote location cases. In this case, we used a NFS mounted disk image for comparison instead of a local file disk image.

In the remote location cases, the UKAI disks achieved better performance in both the random read and random write cases. However, we observed unstable behavior when reading, just as for the local case.

3.17 Mirrored Storage

Fig. 8 shows the results for three mirrored (one on local and the other on remote) UKAI disk im-

[4] http://freecode.com/projects/fio

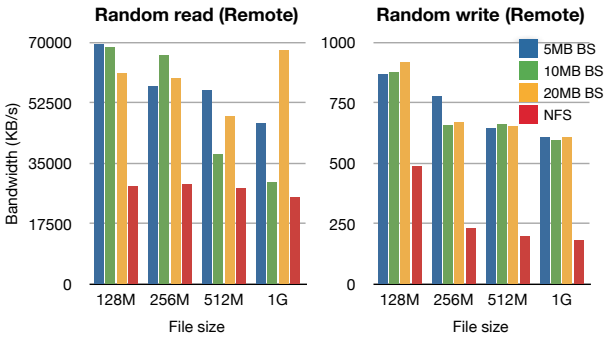

図 7: Comparison of I/O bandwidth of remote disk images

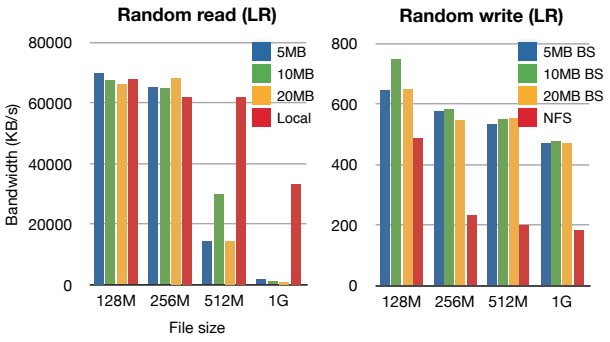

図 8: Comparison of I/O bandwidth of mirrored UKAI disk images and other images

ages. For comparison, a local file disk image result is shown for the random read case and a NFS disk image result is shown for the random write case. Since the prototype UKAI implementation prefers reading from a local node whenever available, it is natural to compare it to a local file when reading. For writing, the UKAI filesystem has to write to both locations. Since a network write operation always happens in this case, a NFS disk is used for comparison.

For reading, the UKAI performance should be similar to that of a local disk because data is read from the local side of a disk in the local-remote mirror case. However, for the 512 MB and 1 GB file size cases, the actual performance was worse than expected. For write operations, even though UKAI writes to two locations, the performance was better than that of NFS.

3.18 Measurement Summary

The results show that performance degrades when the size of a test file is increased. There were some cases where the performance of the UKAI disk was much worse than expected, for example, for random reads in the local storage and mirrored cases. On the contrary, for all the random write cases, its performance was better than local file and NFS storage.

The block size configuration of UKAI storage does not have a serious impact on the overall read/write performance; however, we do not recommend using a large block size because it will impact synchronization performance. If a block size is large, the possibility of accessing a block that is being synchronized increases. Such access results in device-level disk I/O blocking and will cause serious performance degradation at the virtual machine operation level.

3.19 Discussion

We have not identified the reason for the variation in the UKAI storage system performance that we observed in the previous section. Our current hypothesis is that the load of other user-space programs might affect its performance because the UKAI system is implemented in user space. Another hypothesis is that the variation is caused by the nature of a layered filesystem. A virtual machine has its own filesystem and buffering mechanism. Its disk device is in reality provided by a hypervisor and is built on top of FUSE and hypervisor files, both of which also have buffering mechanisms. Because of these layered mechanisms, it is difficult to gain the control needed to optimize the I/O operations of a virtual machine. Where or how to buffer I/O data is currently a vital topic in virtual storage research [24]. We need to investigate the true source of this behavior to achieve a more stable and predictable performance.

We initially thought that using a smaller block size would increase access overhead, especially when operating with large size files. However, it seems that block size did not significantly influence read/write

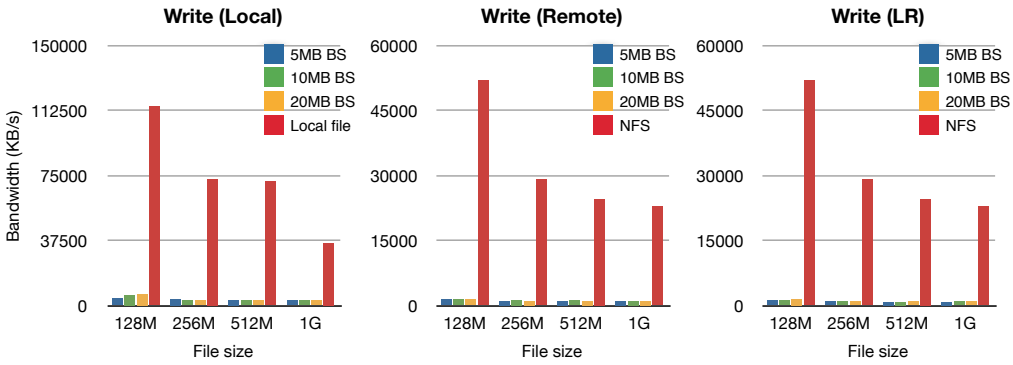

図 9: Comparison of I/O bandwidth for sequential write operations over nine different UKAI disk image configurations

operations. The analysis of its impact on synchronization operations and the determination of the best block size during synchronization are future issues we plan to address.

It was surprising that the random write performance was better than for the local file and NFS storage cases, considering that the current UKAI is implemented in user space in Python and FUSE. We think this is because the filesystem buffering mechanism works efficiently for random write operations on the UKAI disk image blocks that have a much smaller file size compared to the disk image file used by local file and NFS storage. We have not yet measured the amount of resources consumed for UKAI I/O operations when handling a large number of block files, but this may have to be investigated in order to understand the overhead of the UKAI system.

One negative observation not mentioned in the previous section is that we found particularly poor sequential write performance in every case (Fig. 9). This is probably because the UKAI system uses small files to build a disk image. Buffering may not work efficiently in a sequential write operation that spreads over many small files.

3.20 Conclusion

Flexible virtual machine location and/or relocation is a key function for efficient virtual machine resource management. Research on storage management mechanisms for virtual machine image storage is vitally needed to enable relocatable virtual machines. We defined three requirements necessary for a distributed virtual machine image storage system: controllability, redundancy, and locality and proposed the UKAI system. We implemented the concepts of the UKAI system in a prototype and achieved as good or better throughput compared to existing virtual disk mechanisms in most of the random read/write cases common to real-life operations. However, we also found that in some cases, the performance was not stable. We also found that sequential write performance was particularly poor. We continue to investigate the reason for these behaviors and will improve the design and implementation of the UKAI system to provide a better virtual machine image storage mechanism.

4 仮想計算機モニター MIB

WIDE クラウドワーキンググループでは、仮想計算機の状態を運用的に把握する技術として、仮想計算機の情報構造および状態情報へのアクセス手段を規定する MIB オブジェクトを提案している [25]。

昨年度の報告書で述べた通り、Juergen Schoenwaelder 他から同様の MIB オブジェクトが提案されていたため、今年度はこれらの提案を統合し、共同で IETF OPSAWG にインターネットドラフトとして提

出した。提案ドラフトはバンクーバーで開催された第88回IETFでの議論を経て、OPSAWGのワーキンググループドラフトとして承認されるに至っている。今後、RFC化に向けて内容の精査を継続していく。

5 まとめ

2013年度はクラウド基盤の運用技術、管理技術の研究開発、広域データセンター運用における仮想ディスクイメージ提供技術の研究開発を実施した。また、今後重要になるクラウド環境におけるネットワーク技術への取り組みを開始し、クラウドデータセンター運用の参照モデルの検討を開始した。WIDEクラウドワーキンググループでは、今後も広域分散環境でのクラウド運用を実現するための技術開発を継続していく予定である。

参考文献

[1] Marc Lasserre, Florin Balus, Thomas Morin, Nabil Bitar, and Yakov Rekhter. Framework for DC Network Virtualization, draft-ietf-nvo3-framework-03.txt. Internet Draft, IETF, July 2013.

[2] Amazon VPC. `http://aws.amazon.com/vpc/`.

[3] Theophilus Benson, Aditya Akella, Anees Shaikh, and Sambit Sahu. CloudNaaS: a cloud networking platform for enterprise applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 8:1–8:13, New York, NY, USA, 2011. ACM.

[4] M.Mahalingam, D.Dutt, K.Duda, P.Agarwal, L. Kreeger, T. Sridhar, M.Bursell, and C.Wright. draft-mahalingam-dutt-dcops-vxlan-00.txt. ID, IETF, Aug 2011.

[5] Yoshihiko Kanaumi, Shuichi Saito, Eiji Kawai, Shuji Ishii, Kazumasa Kobayashi, and Shinji Shimojo. Rise: A wide-area hybrid openflow network testbed. *IEICE Transactions*, 96-B(1):108–118, 2013.

[6] INTEROP Tokyo 2013. `http://www.interop.jp`.

[7] ShowNet 2013. `http://www.interop.jp/2013/shownet/index.html`.

[8] Python Jinja2. `http://jinja.pocoo.org/docs/`.

[9] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP'03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM, 2003.

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

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

[12] VMware, Inc. VMWare. http://www.vmware.com/.

[13] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI'05)*, volume 2, pages 273–286, 2005.

[14] Jennifer Rexford. Programming Languages for Programmable Network. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL'12)*, pages 215–216, 2012.

[15] 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)*, pages 460–465, 2009.

[16] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *Proceedings of the 3rd internatiolan conference on Virtual execution environments*, pages 169–179, 2007.

[17] Jie Zheng, Tze Sing Eugene Ng, and Kunwadee Sripanidkulchai. Workload-Aware Live Storage Migration for Clouds. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE'11)*, pages 133–144, 2011.

[18] Lars Ellenberg. DRBD® 9 & Device-Mapper Linux® Block Level Storage Replication. In *Proceedings of Linux-Kongress 2008*, October 2008.

[19] 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)*, pages 307–320. USENIX, 2006.

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

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

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

[23] Douglas Crockford. *The application/json Media Type for JavaScript Object Notation (JSON).* IETF, July 2006. RFC4627.

[24] Vasily Tarasov, Deepak Jain, Dean Hildebrand, Renu Tewari, Geoff Kuenning, and Erez Zadok. Improving I/O Performance Using Virtual Disk Introspection. In *Proceedings of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*, June 2013.

[25] Hirochika Asai, Michael MacFaden, Juergen Schoenwaelder, Yuji Sekiya, Keiichi Shima, Tina Tsou, Cathy Zhou, and Hiroshi Esaki. *Management Information Base for the Virtual Machine Monitoring.* IETF, October 2013. draft-asai-vmm-mib-05.