特集号
招待論文

複数の異種クラウド間におけるスケールアウトおよびディザスタリカバリ機構の実装とその評価

波戸 邦夫[†1]　上水流 由香[†2]　岡本 隆史[†3]　横山 大作[†4]

[†1] NTTセキュアプラットフォーム研究所　[†2] NTTコミュニケーションズ（株）
[†3] NTTデータ先端技術（株）　[†4] 東京大学

既存のクラウドシステムに対して社会基盤として要求される高い信頼性や多様性に対する課題を解決するため，複数の異種クラウドシステム間で連携してリソースを監視，解析，制御するインタークラウドのシステムアーキテクチャを提案し，テストベッド評価した結果について報告する．

1. はじめに

近年，サーバ仮想化技術[1],[2]の発展と，インターネットやモバイルネットワークのブロードバンド化により，企業ユーザにおいてはその経済性を主たる理由としてクラウドシステムの導入が進んでいる一方，クラウドシステムのメリットであるシステムの俊敏性や柔軟性などに着目し，時代の変化に追随可能な社会基盤としての利用が着目されてきている[3]．また，東日本大震災の被災やクラウドシステムの大規模障害によって自治体や企業等が保持していた再構築不可能な情報を喪失したことから，情報やそれを保持するシステムへの信頼性への要求が非常に高まっている．さらに，ユーザニーズの多様化が進み，それらのニーズを実現可能なシステムが求められている．

上記の要求条件のうち，経済性，俊敏性，柔軟性の実現は単一のクラウドシステムにおいて実現可能であるが，信頼性，多様性の実現については不十分であると考える．広域災害発生時には，単体で災害対策がされたクラウドシステムであっても，それを支える電源供給やアクセスネットワークが断たれサービスを継続できない可能性がある．また，多種多様なサービスメニューを単体のクラウドシステムで実現することは，費用対効果の面から現実的ではない．

そこで筆者らは，既存のクラウドシステムに対して社会基盤として要求される高い信頼性や多様性に対する課題を解決するため，複数の事業者が提供するクラウド間で連携してリソースの双方を監視，解析，制御するインタークラウドのシステムアーキテクチャを提案する．提案アーキテクチャは具体的には，単一クラウドでは対応が困難な大規模災害や大規模故障発生時に，クラウド間でのスケールアウト（SO: Scale Out）☆1やディザスタリカバリ（DR: Disaster Recovery）が可能となり，高信頼なサービスが継続できることを目指す．

本稿では，第2章で従来技術の紹介とその課題，第3章で課題を解決するためのインタークラウド技術とそのフレームワークを示す．第4章では第3章で示した提案方式の実装方法および実装に伴うプラクティスを述べる．第5章では提案方式のテストベッドによる実装評価結果から，クラウド間の連携により信頼性と多様性の向上が可能であることを示し評価に伴うプラクティスを述べる．

2. 従来の高信頼化技術とその課題

既存のSOやDRの技術は，単一クラウド内での高信頼化を達成可能である[1],[4],[5]．これらのシステムやサービスは，動作中のリソースの状態を監視し，リソースの状態とあらかじめ設定された閾値等の情報を比較することで，SOやDRの契機を知ることができる．また，いくつかのIaaS系のクラウドサービスでは[6],[7],[8]，ユーザからの要求によって仮想マシン（VM:Virtual Machine）とともに，VM間を相互接続する仮想ネットワークを提

☆1 本稿中の略語は，末尾の略語一覧を参照

供する．VMと仮想ネットワークを利用するユーザは，VMが動作する物理サーバやネットワーク機器など物理環境を意識することなく，仮想化されたそのユーザ専用の環境を利用する．その際に単一のクラウドサービスが提供するリソースの中から，ユーザが必要とするリソース量を動的に割り当てることで，SOやDRを実施することができる．

しかし，大規模災害や大規模故障への対応には従来技術では限界があった．たとえば，天災等でデータセンタ（DC: Data Center）自体が被災し，電力の供給や外部のネットワークなどが断たれた場合，クラウドシステム内の冗長機能による復旧は役に立たない[9]．また，クラウドシステムの管理システムのバグや故障などが発生した場合，クラウドシステム全体に影響が波及し，その上で提供されるサービスの停止につながる[10]．

上記のような大規模災害や大規模故障への対応は，原理的に単一のクラウドシステムでは不可避であり，複数かつ異種のクラウド間の連携によってのみ解決することができる．クラウド間連携でSOやDRを実現する場合も，ユーザに対しては複数の異種クラウドの存在を意識させることなく利用できることが望ましい．複数の異種クラウドシステムの連携を単一クラウド用の既存技術で実現する場合，以下の4つの課題がある．

• 課題1：異種のクラウド間の互換性

近年，クラウド基盤上では仮想化技術として，KVM（Kernel-based Virtual Machine），Xenなど，基盤ソフトウェアとしてVMWare，Eucalyptus，OpenStackなど多種多様な技術が使われるようになってきている．また，利用するハードウェア性能もクラウド基盤によって異なる．このように，異種のクラウドシステムが前提となる場合，クラウド間のAPIの整合性やVMのフォーマットの違い，性能の違いなどをうまく扱うことが課題となる．

• 課題2：クラウドをまたがるVM間ネットワーク構築とアクセスネットワークの制御

複数の異種クラウドが連携する環境においては，SOやDRのタイミングでユーザが利用するVM間に専用のネットワークが構成できれば，クラウドの違いを意識することなくSOやDRを実現できる．GREトンネルなどにより遠隔拠点間でユーザごとに専用のネットワークを仮想的に構築する技術や製品は提供され始めているが，特定のクラウドプラットフォームやベンダ機器が提供する技術であり，連携するすべてのクラウドシステムへの適用は難しい．

また，エンドユーザがアクセスするサーバが異なるクラウドへ移動する可能性があるため，エンドユーザのアクセス先を，移動先クラウドのサーバへ切り替える制御が必要になる．エンド端末のアクセス先の切替え方式として，DNSによる方法が考えられるが，DNSのエントリを書き替える方法の場合，短時間での切替えや，端末の属性に応じてサーバを選択させるといった，きめ細かな制御は難しい．

• 課題3：複数の異種クラウド間での性能に応じたリソース量の決定

SLAを担保した状態でSOを行うためには，投入リソース量と性能との関係を与える性能モデルが必要である．多数のコンポーネントからなる大規模なクラウドアプリケーションにおいては，性能特性の変動は複雑なものになる．たとえば，3層モデルのアプリケーションで，要求需要が小さい時点ではアプリケーション層の性能が律速要因になるが，需要が大きくなるとDB層がボトルネックになる，といった挙動が発生しがちであり，単純な方法で性能予測をすることは困難である．これを解決するためには，あらかじめリソースを変化させた状況下でアプリケーションを動作させて性能測定を多数行い，性能特性を取得してモデルとし，予測に役立てるという方策が有効である[11],[12]．

しかし，複数のクラウドシステムを利用してアプリケーションを動作させる際，利用可能なクラウドシステムは多数存在し，あらかじめすべての環境で予備実行を行うことは現実的ではなく，このような方策による性能特性予測は困難になる．

• 課題4：データセンタリソースとネットワークリソースの監視・制御が独立

クラウドシステムが連携した場合，ユーザに提供される単一のサービスが複数のクラウドシステムにまたがって提供される可能性が生じる．データセンタリソースとネットワークリソースが異なる事業者によって提供される場合において，両リソースの監視と制御は，既存技術では統合されていないため，データセンタリソースの過負荷に伴うネットワークリソースの変更などを一元的に行うことができないという課題がある．

3. インタークラウドの提案

ユーザに複数の異種クラウドの存在を意識させることなく，その中から最適なリソース配置を決定し，ユーザのシステムを再構成するインタークラウドに必要となる技術群と，それらを組み合わせるシステムアーキテクチ

ャを提案する．

3.1　インタークラウドアーキテクチャ

インタークラウドのシステムアーキテクチャは，複数の異種クラウドシステムを連携することで実現する．ここでクラウドシステムは，コンピューティングリソースとストレージリソースを提供するDCとネットワークリソースを提供するネットワークインフラストラクチャ（NW: Network Infrastructure）の一方または双方によって構成される．

本提案技術は既存のクラウドシステムを置き換えるものではなく，クラウド間の機能として追加する機能である．そのため，既存のクラウドシステムの構成に基づいた機能配備が必要となる．また，単純にクラウド間を結合していくと，システム規模が異常拡大し，破綻する可能性がある．そこで，クラウドシステム内とクラウド間での監視，解析，制御技術間の関係性を規定する．

具体的には**図1**に示すように，(1) クラウドシステムの構成要素である，データの計算処理と蓄積を行うDCを直接監視・制御し，クラウド間で動的にサーバ・ストレージを再構成することでSOやDRを実現するコンピューティングリソース動的プロビジョニング技術を持つデータセンタオペレーションシステム（DC-OPS: Data Center Operation System），(2) データの転送を行うNWを直接，監視・制御し，クラウド間で動的にNWを再構成・最適化するNWリソース動的プロビジョニング技術を持つネットワークオペレーションシステム（NW-OPS: Network Operation System），(3) サービス品質要件から必要なリソース要件を推定するリソース配置提案ツール（RAO: Resource Allocation Optimizer），(4) サーバ・ネットワークリソースを一元監視し，サービス品質要件を考慮してクラウド再構成指示を行うクラウド間自立的リソース・発見技術を持つインタークラウドサーバ（ICS: Inter-cloud Server）を定義する．

DC-OPSとNW-OPSは，それぞれ3.2節で示すクラウド連携マネージャ（CFM: Cloud Federation Manager）と既存のクラウド管理システムの機能，3.3節で示すVirtual Network Generator（VNG）と既存のネットワーク管理システムを含むものであり，各DCやNWの詳細な管理情報を把握し，外部に対してはその詳細な情報を隠蔽する機能を有するものである．またRAOは，3.4節で示す性能に応じたリソース量と構成の決定をソフトウェアモジュールとしたものである．ICSは，DC-OPSとNW-OPSとによって詳細な情報を隠蔽され，抽象化されたDCとNWのリソースの情報を扱うことで，システム規模が拡大しても現実的な時間内でデータ処理が可能である．

3.2　異種のクラウド間の互換性向上

クラウド間を連携させる工夫としては，各クラウドごとのAPIの差を吸収するCFMを提供する．また，仮想システムとイメージの定義を分離し，クラウド間に跨って定義できる情報とクラウド個別に用意が必要な情報の定義を分けた．

クラウド間に跨って定義できる共通情報としては，システム管理標準を策定しているDMTF（Distributed Management Task Force）によりOVF（Open Virtualization Format）としてVMを定義するファイルフォーマットが標準化されている．OVFをベースとしたクラウド共通で利用できるシステム構成情報を定義することにより，他のクラウド事業者が採用しやすいシステム構成情報を定義することができる．

また，VMの起動時の処理の情報など，VM固有の処理情報は，VMの初期化スクリプトとして，クラウド基盤の影響を受けないように実装することにより，共通的に利用することができる．

3.3　クラウド外ネットワークの仮想化と動的制御

複数の異種クラウド間でユーザVM間を接続するネットワークの構築と，エンド端末のアクセス先のきめ細かな切替えという課題に対し，クラウド外へのネットワークの仮想化とその動的制御を実現するDynamic IP-VPN[13]を適用する．

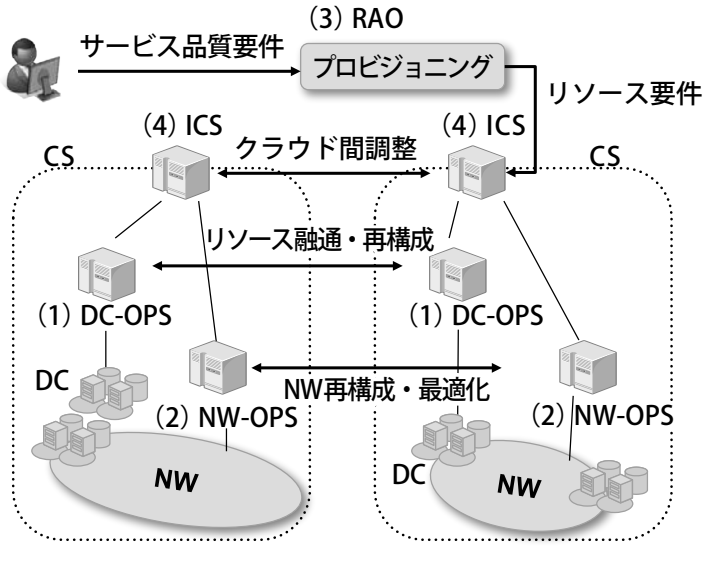

図1　階層的なフレームワーク

Dynamic IP-VPNはVirtual Network Server（VNS）とVirtual Network Client（VNC）と呼ぶソフトウェアの間に，IP-in-IPのVPNにより，端末を遠隔のLANへ仮想的に接続する仮想ネットワークを構成する．VNS/VNC間の接続構成はVNGと呼ぶ管理サーバが制御する．

VNS/VNC間の接続設定は，それぞれに割り当てられたIDを用いて行う．VNS/VNCは起動時にVNGから接続設定に必要な設定情報を取得するため，VPN接続先IPアドレスなどのVPN設定情報を持たない．VNGからの制御により，IDで設定されたVNS/VNCの論理的な構成に従ったVPN接続を動的に実現する．

仮想ネットワークを前提とすれば，複数の異種クラウドで構成される環境においても，クラウドの違いを意識することなく，SOやDRに必要なクラウド間のネットワークや，エンド端末がサーバに接続するアクセス用ネットワークを実現できる．

3.4 性能に応じたリソース量と構成の決定

前述のように，アプリケーションの性能特性調査をすべてのクラウドシステム上で事前に行っておくという手法は実現困難な場合が多い．この問題に対し，我々は，システムとアプリケーションの性能特性を別々に測定し，それらを統合して実行時の性能予測を行うという方策を利用することとした．CPU性能，ディスクI/O性能などの基本性能をよく反映する小規模なベンチマークプログラムを用い，利用するクラウドシステムすべてであらかじめ実行することで，統一基準によるシステムの性能特性を取得しておく．一方，対象アプリケーションの性能はあるシステム上で精密に測定しておく．この2者を利用することで，任意のシステムにおける性能予測が可能になると期待される．我々は，このような方針の下，アプリケーションの性能モデルを精度よく作成する研究を行ってきた[14]．本検証実験では，multi-tier型のWebアプリケーションを対象に，複数の異種クラウドシステム上での性能モデル推定を行った．

このように複数の異種クラウドシステム上でのアプリケーション性能モデルを作成すると，所与のリソースでSLAを担保できるか否かが判定できる．しかし，SLAが担保できるようなリソース構成は通常多数存在するため，この中で最も望ましい構成を選んで，実際にリソース配置を行うことが必要になる．リソース構成の望ましさの基準項目は多岐にわたり，管理者の手間の大小や好みといった定式化しにくい基準も考えられるが，ここでは以下の項目について数値化し，総和コストが少ない構成が望ましいとして定式化することにした．

- インスタンス稼働コスト（円／時間）
  インスタンス実行にかかるランニングコスト．
- サイト利用コスト（円／時間）
  1つ以上インスタンスが実行されているクラウドサイトに課金されるコスト．
- リソース遷移コスト（円）
  リソースの利用開始，利用終了に伴いかかるコスト．

また，どの程度の時間を考慮してコスト計算を行うかを示す時定数を設定する．これを長く設定するほどリソース遷移コストの占める割合が小さくなり，継続的なランニングコストが重視されるようになる．この時定数の期間だけアプリケーションが動作すると仮定したときのコストの総和を計算し，最小コストのリソース配置を提案することで，望ましいリソース配置を得ることが可能になる．

我々は，このような問題定式化に基づいたRAOを開発した．RAOの動作は，最小コストのリソース構成を探す探索問題を解くことに帰着される．すべての組合せを生成して探索すると組合せ爆発を生じるため，本実験ではヒューリスティックに基づいて以下のように解の候補を絞った．

- 利用可能なインスタンスすべての中でコスト最小のものから順に利用する場合．
- なるべく1つのクラウドシステム内のリソースを優先して利用する場合．

これらの場合について，それぞれ，

- 現在利用しているインスタンスを考慮しないか，そのまま保存するか．
- どのアプリケーションコンポーネントにコストの低いリソースを割り当てるか
- どのクラウドシステムを優先するか

という選択肢を加え，SLAを担保できるリソース配置案を生成して，コスト最小の配置を探索した．

4. インタークラウドの実装と得られたプラクティス

提案技術の実装にあたり，インタークラウドに含まれる複数の機能が組み合わされて構成されるため，その機能分担と機能間の連携方式について提案する．

4.1 インタークラウドサーバ

3.1節のアーキテクチャに基づき，CFM，VNG，RAO

と連携し複数の異種クラウドのリソースを監視・制御する機能をICSとして実装した．ICSはリクエスト＆レスポンス形式でCFMやVNGと通信し，リソースの監視・制御を実施する．

4.2　クラウド連携マネージャ

CFMは，**図2**に示すように事業者ごとに異なるクラウドI/Fをラップし，ICSに統一的なインタフェースを提供し，下記のコンポーネントから構成される．

• 連携I/F

各クラウドの実装の違いを隠ぺいし，ICSに対して統一的なインタフェースを提供する．利用できるリソース一覧や利用可能なVMのスペックやコスト，VMの負荷状況などの情報の提供，リソース予約，確保，解放などを行う．

• 構成管理モジュール

システム情報やVM情報を管理する．システムの起動順序を記載したOVFを解釈し，VMの起動やVM上のサービスの起動の順番を制御する．

• クラウド操作モジュール

各クラウドの実際の操作を行う．実装が異なるクラウドごとにドライバのような形でモジュールは用意する．

• VM操作モジュール

VMの中のエージェントと通信し，VMやVMの中のサービス起動・停止時のスクリプトをVMにコピーしたり，サービスの起動・停止処理の指示を行う．

• エージェント

各VMに配置し，サービスの起動・停止の制御など，各VMタイプ（ex. Webサーバ，DBサーバ等）ごとの固有の処理を分担する．

4.3　仮想ネットワークジェネレータ

クラウド間連携に必要な仮想ネットワークを，生成・変更・削除するための仮想ネットワークの制御用インタフェースや，仮想ネットワークの状態の監視用インタフェースなどの機能をVNGに実装した．

仮想ネットワークを構成するためには，VNS/VNCのID情報のVNGへの登録と，VNS/VNCの起動の2つのステップが必要となる．そこでVNGへのIDの登録・削除用のAPIと，端末上でVNS/VNCを起動・停止する機能を作成し，それら機能をCFMへ提供した．

クラウドシステム上でVPN接続を待受けるVNSの配置方法としては，**図3**に示す2種類の方法を用いた．アクセス用の仮想ネットワークでは，クラウドシステムでNAT機能を提供するサーバ上に複数ユーザ向けのVNSを配置した．一方，クラウド間の仮想ネットワークでは，ユーザVMの上にVNSを配置した．どちらの方法も，VNCを起動した遠隔の端末またはVMから，クラウド1のVLANへの接続を実現する．2つの方法を比較すると，後者の方法はVNSの配置にユーザのVMを利用しており，クラウドシステムの基盤機能にVNSを組み込む必要がない．クラウドシステムの基盤機能に手を入れることなく，ユーザVMへの機能追加のみで実現できるため，多様なクラウドシステムでの利用に適した方法である．

4.4　Resource Allocation Optimizer

SLAを担保できるようなリソース配置のプランを作成する機能については，RAOとして機能分割して実装を行った．望ましいリソース配置の提案機能については，クラウドシステムの実装などから生じる個別な制約とは

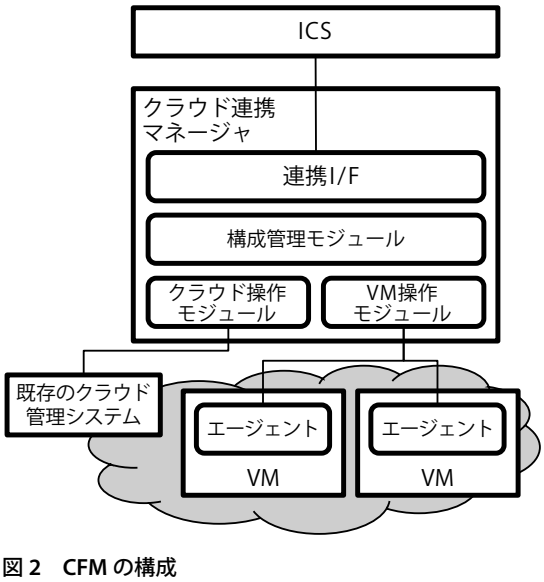

図2　CFMの構成

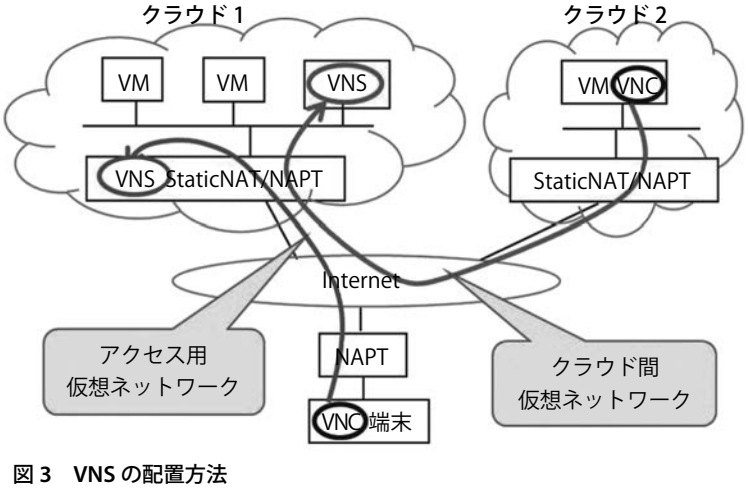

図3　VNSの配置方法

切り離し，できるだけ汎用のリソースプランニングモジュールとなるよう設計，実装を行った．そのため，RAOは出力としてリソース配置案をコストの小さい順に複数個出力する．データを外部サイトにコピーしたくない，などの何らかの制約が存在する場合は，RAOの出力を適宜フィルタして利用することを想定している．

4.5 実装におけるプラクティス

当該実装から得られたプラクティスを述べる．

当初，デバッグを容易にできるようにVMをシーケンシャルに起動する制御を行うように設計していた．そのため，大量のVMを利用したSOやDRを行う際に，VMの起動時間が台数に比例して増加する問題が5.2節に記載する大規模環境の実験を行った際に発覚した．開発時には大規模な仮想環境が利用できず，机上と小規模な検証のみしか行っていなかったため，大規模環境で検証するまでこの問題に気が付かなかった．しかしながら，早期に大規模な仮想環境で検証したため，十分な修正時間を取ることができ，検証を円滑に進めることができた．

災害発生時に利用する安否確認システムなど，普段はまったく利用する必要がないが，突然急激にユーザアクセスが増えるようなシステムでは，事象発生時に迅速に大規模なSOを行う必要がある．このようなシステムでは，普段はアクセスが低いため，まずは平常状態で試験を行い，大規模な環境での試験は後に回されることがある．今回のような特定の条件下でしか高負荷にならないようなシステムでも，アーキテクチャの妥当性を確認するために，早期に大規模環境で検証を行うことが重要であるという知見が得られた．

5. テストベッドによる評価

第4章の実装を用いたテストベッド評価を実施した結果について述べる．

5.1 テストベッド環境

テストベッド環境は，東北（仙台）に1つ，九州（福岡）に1つ，関東（川崎）に2つの合計4つのクラウドを持ち，各クラウド間はJGN-Xによる広域ネットワークで接続されてIPv6でルーティングされている．また，1つのクラウドあたり1Uサーバを10台設置し，VMあたり1CPUコア1GBメモリを割り当てた場合においても50VM以上動作させることを可能とした．

5.2 性能検証

性能検証として，投入する負荷量を調節することでSOするVM数を変化させ，それぞれの処理完了までの時間を計測した．

SOによって増加するVM数をそれぞれ，Small=7VM，Medium=33VM，Large=65VM，Huge=1,000VMとするSO時の処理時間を**図4**に示す．処理順に，事前処理，VM起動，NW変更を積み上げているが，NW変更はいずれも1秒未満であったためグラフでは現れていない．

5.3 テストベッド評価の考察

提案アーキテクチャが，要求条件に従ったテストベッド上で期待通り動作することから，複数の異種クラウドに跨ったインタークラウド環境が構築可能であり，クラウド間で負荷発生から分オーダでSOが可能であることを示した．

5.3.1 複数の異種クラウドにおけるVMと仮想ネットワークの制御

VMの制御により，SO時に離れた拠点のクラウドにWeb/APサーバを作成し，正しくSOができることを確認した．また，Eucalyptus+Xen，OpenStack+KVMと，異なるハイパーバイザーを利用した異種のクラウド間においてDR処理が正しく行えることを確認した．

仮想ネットワークの制御として，**図5**のような異種のクラウド間で仮想ネットワークが動的に構成できることを確認した．図5は，初期構築時にはCloud01にWeb/APサーバとDBサーバを，SO時にはCloud02にWeb/APサーバを作成する構成例を示す．Cloud01とCloud2の間には点線で示す仮想ネットワークが作成され，Cloud02のWeb/APサーバはCloud1のLANに接続された状態になり，

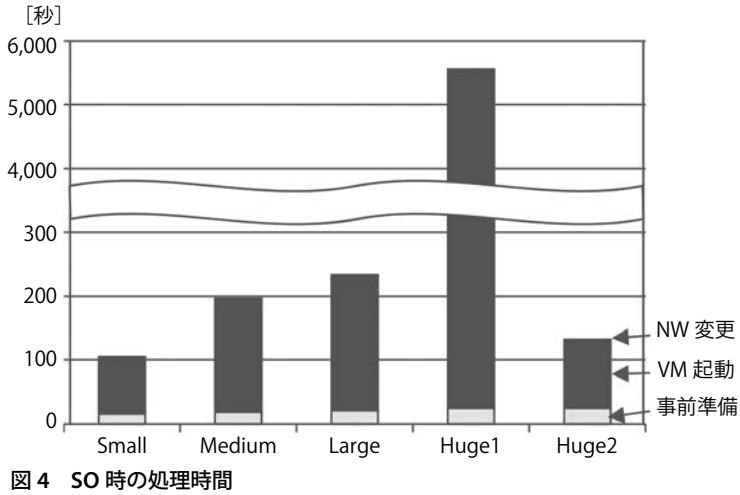

図4 SO時の処理時間

異種のクラウド間でサーバ間通信が可能になる．

5.3.2　リソース最適配置

RAOはリソース配置案をヒューリスティックに基づき絞って探索するため，本検証実験の場合ほぼ即時に候補リストを得ることができた．

また，いくつかの現実的な状況に基づいたSO，DR等のシナリオを作成して行った実験で，管理者の持つ明示的・暗黙的な要求をおおむね反映できるようなリソース構成が提案できることが確認でき，要求の定式化手法がある程度妥当であることが確認できた．一方，本定式化の範囲を超えたより複雑な要求，たとえば，利用するサイトごとに少なくとも1つの管理用インスタンスを必要とするアプリケーションなどが存在することも分かった．今後より多くの事例への適用を試みる中で，必要な記述性能を失わない抽象化方法を検討する必要があると考えられる．

5.3.3　スケールアウト処理時間

SO処理の処理時間の9割弱は，VM起動時間が占めている．よって処理時間短縮にはVMの起動並列化などが有効である．VM起動にかかる時間は，VM数に応じて増加傾向にあるが単純な比例関係ではない．同一ハードウェア上のサーバで起動するVMについては互いの起動が影響するためVM数に応じて処理時間が増加するが，異なるハードウェアや異なるクラウドで起動する場合は互いの影響が少なく並列処理が可能となり，VM数に応じた処理時間が不要となるためだと考えられる．また，クラウド管理システムとしてOpenStack経由で1,000台のVMを起動した場合（図4:Huge1），OpenStackが管理するネットワーク設定や，データベースアクセス等，起動以外の処理に伴うオーバヘッドが発生し，起動時間に大きく影響することが分かった．OpenStackを経由せず1,000台のVMを起動した場合（図4: Huge2）は2分弱で起動が完了したことから，多量のVMの同時高速起動には，OpenStack経由でのオーバヘッドが課題であることが分かった．

5.4　評価実験から得られたプラクティス

本評価実験で大きな労力を要したのは，複数DCを利用するSOやDRが必要になるような現実的な局面において，そこで求められるリソース配置への漠然とした要求を本システムの枠組みである最適化問題へどのように反映するのかを検討した部分である．アプリケーションの提供者は，VMインスタンスの実行にかかる純粋な費用を安くするという以外に，さまざまな漠然とした要求を持つ．ここでは，大きく3つの要素のトレードオフによって要求が規定されていると整理し明確化した．

a) 配置をできる限りシンプルにしたい: 利用するサイト数を減らし，インスタンスのバリエーションを減らしたいという要求．

b) 配置する領域を広げたい: DRのシナリオを考慮するときに顕著であるが，障害に対する耐性を高めるためには，物理的に離れたサイトを利用する，異なるサーバにインスタンスを立てるなどの要求が発生する．また，全世界など広域から需要が発生するアプリケーションにおいては，Webのリクエストをさばくフロントのサーバなどはなるべく需要地に近いサイトに置きたい，すなわちなるべく広域に広がったSOを行いたい，といった要求も存在する．

c) 時間的な変化をできるだけ減らしたい: 現時点の構成からの変更をできるだけ抑えたいという要求．

これらa,b,cは，それぞれ相反する性質の構成を志向するものとなる．aとbは分かりやすく方向性の異なる要求であり，cはaやbで求められる最適構成が現在の状況とかけ離れたものであるときに，現在の構成のままでいようとする要求である．一般的に，アプリケーション提供者の要求はおおむねこの3点の方向性で整理できるのではないかと考えている．

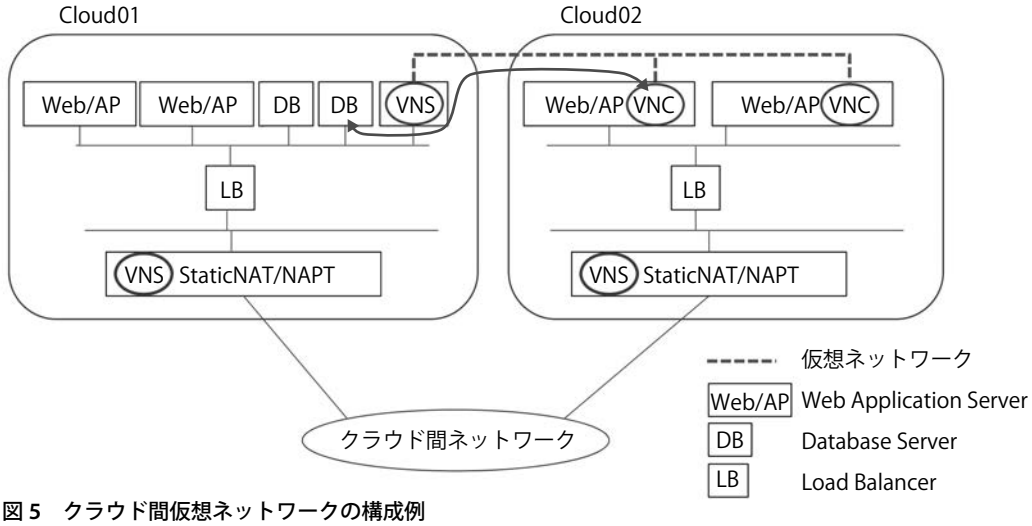

図5　クラウド間仮想ネットワークの構成例

我々が得た知見の

1つは，最適なリソース配置を決定するには，一般に考慮される実行に必要な費用および得られる性能に加えて，上記3点を評価すべきということである．他のアプリケーション配置問題を考える際にも，本知見は役立つと考えられる．

研究開始当初，シンプルな疑似アプリケーションで設計検討を行っている際は，この3点の要求に関しては明確に認識することができなかった．bに関してはターゲットアプリケーションを実際に構築したときに初めて，実装に用いる基盤ソフトウェアの制約に直面したことで認識された．適用対象の実用アプリケーションを設計段階のような早い時期から利用することが，開発における重要なポイントであったといえる．またcに関して，当初は「需要に対してある構成の提供容量がぎりぎりで，頻繁に拡大／縮小処理を繰り返す」ような特異的問題状況を想定したのみであった．このため，リソース配置問題はむしろ時間的なコストを考えず，その時点の瞬間的な利用リソース量に対するコストのみを最適化する問題として設計を行い，RAOを呼び出す制御側で適度に調整を加えるようなシステム構成を検討していた．時間的な変化の抑制がアプリケーション提供者の持つ「要求」であると捉えられていなかったといえる．

これら3点の要求事項を最適化問題に落とし込むために，以下のような項目を評価関数に追加することとした．aについては，サイト数増加に伴って増加するコストとして表現できる．bについては，広がりたいという要求の原因がさまざまに異なることが想定されたため，今回は一般的な表現方法を確立するまでに至らなかった．本実験においては，性能面から需要地に近いサイトを利用したいとの要求については性能モデルにおいて表現することとし，実装の都合上からすべてのサイトを利用したいという要求はハードコーディングによる制約として表現した．cについては，時定数Tを設定することで「これからTの期間だけアプリケーションが稼働するとしたときに，その期間中に必要となるコストを最小化する」という最適化問題に帰着することで表現できる．このような手法で，漠然とした要求を1つの評価関数によって表現することが可能になった．

6. むすび

経済性が求められるクラウドシステムにおいて，一般には背反する要件である高信頼化と多様化を実現するため，クラウド間でリソースを連携するインタークラウドシステムを提案した．テストベッドによる実装評価により，クラウド間でのスケールアウトやディザスタリカバリが可能であることを示し，行政，医療，金融などの高度な信頼性を要求するインフラストラクチャへのクラウドの適用の道筋を開いた．

本稿が対象とした複数の異種クラウドのような大規模システムは，実験室レベルの規模で実施する事前検証のみでは正確な評価が難しいため，技術そのものと，その技術を使いこなすプラクティスの両者に関する知見が特に重要である．今後実運用に入った際にも，事前には予見不可能であった不具合への対応や機能追加，システム更改などが不可避であるが，これらも積極的にプラクティスを蓄積し共有することが効果的であろう．

謝辞　本論文の内容には，総務省委託研究「広域災害対応型クラウド基盤構築に向けた研究開発（高信頼クラウドサービス制御基盤技術）」の成果が含まれます．

参考文献

[1] VMware, http://www.vmware.com/
[2] RedHat, KVM, http://www.redhat.com/virtualization/
[3] 坂井　博 他：インタークラウドのユースケースと機能要件，電子情報通信学会技術研究報告.
[4] Amazon Web Services: Auto Scaling, http://aws.amazon.com/jp/autoscaling/
[5] Amazon Web Services: 災害対策（DR), http://aws.amazon.com/jp/disaster-recovery/
[6] NTT コミュニケーションズ：Cloudn JP-WEST/US リージョン，http://www.ntt.com/cloudn/data/server.html
[7] ニフティ：NIFTY Cloud プライベート VLAN，http://cloud.nifty.com/service/plan.htm
[8] Amazon Web Services: Amazon Virtual Private Cloud（Amazon VPC), http://aws.amazon.com/jp/vpc/
[9] Amazon Web Services: Summary of the AWS Service Event in the US East Region, http://aws.amazon.com/message/67457/
[10] Google: This is your pilot speaking. Now, about that holding pattern... , http://googleblog.blogspot.jp/2009/05/this-is-your-pilot-speaking-now-about.html
[11] Yang, H., Luan, Z., Li, W. and Qian, D.: MapReduce Workload Modeling with Statistical Approach, Journal of Grid Computing, 10:279-310 (2012).
[12] Jayasinghe, D., Swint, G., Malkowski, S., Li, J., Wang, Q., Park, J. and Pu, C.: Expertus, A Generator Approach to Automate Performance Testing in IaaS Clouds, IEEE CLOUD 2012.
[13] Hata, H., Kamizuru, Y., Honda, A., Hashimoto, T., Shimizu, K. and Yao, H.: Dynamic IP-VPN Architecture for Cloud Computing, APSITT2010 (June 2010).
[14] Groot, S., Goda, K., Yokoyama, D., Nakano, M. and Kitsuregawa, M.: Modeling I/O Interference for Data Intensive Distributed Applications, SAC 2013.

波戸 邦夫（非会員）hato.kunio@lab.ntt.co.jp
日本電信電話（株）NTT セキュアプラットフォーム研究所 主任研究員．1997 年 東京工業大学工学部電気電子工学科卒業．1999 年 同大学院総合理工学研究科物理情報工学専攻修士課程修了．修士（工学）．IP-VPN，広域イーサネット，ネットワークセキュリティに関する研究開発に従事．IEICE 会員．

上水流 由香（正会員）yuka.kamizuru@ntt.com
NTT コミュニケーションズ（株）先端 IP アーキテクチャセンタ 担当課長．1991 年 電気通信大学電気通信学部計算機科学科卒業．1993 年 同大学院情報工学専攻修士課程修了．修士（情報工学）．IPv6，IP-VPN に関する研究開発に従事．IEICE 会員．

岡本 隆史（非会員）okamototk@intellilink.co.jp
NTT データ先端技術（株）ソリューション事業部 シニア IT スペシャリスト．1997 年 岡山大学大学院自然科学研究科電子情報システム工学専攻博士前期課程修了．修士（工学）．同年（株）NTT データに入社，2013 年 4 月より現職．Web サービス，クラウド基盤，OS の研究開発に従事．

横山 大作（正会員）yokoyama@tkl.iis.u-tokyo.ac.jp
2006 年東京大学より博士号取得．博士（科学）．2002 年より同大新領域創成科学研究科助手などを経て，2009 年より同生産技術研究所助教，現在に至る．並列プログラミング環境，ゲームプログラミングに関する研究に従事．

投稿受付：2013 年 5 月 22 日
採録決定：2013 年 7 月 26 日
編集担当：平田圭二（公立はこだて未来大学）

付録：略語一覧

CFM	Cloud Federation Manager: クラウド連携マネージャ
CS	Cloud Server: クラウドサーバ
DB	Database: データベース
DC	Data Center: データセンタ
DC-OPS	Data Center Operation System: データセンタオペレーションシステム
DMTF	Distributed Management Task Force
DR	Disaster Recovery: ディザスタリカバリ
ICS	Inter-cloud Server: インタークラウドサーバ
KVM	Kernel-based Virtual Machine
NW	Network Infrastructure: ネットワークインフラストラクチャ
NW-OPS	Network Operation System: ネットワークオペレーションシステム
OVF	Open Virtualization Format
RAO	Resource Allocation Optimizer: リソース配置提案ツール
SO	Scale Out: スケールアウト
VM	Virtual Machine: 仮想マシン
VNC	Virtual Network Client
VNG	Virtual Network Generator
VNS	Virtual Network Server