Vol.2018-OS-144 No.10
2018/7/31

インタークラウド環境構築システムによるアプリケーション環境構築支援のための機能拡張

竹房 あつ子[1,a)]　丹生 智也[1]　佐賀 一繁[1]　横山 重俊[2]　合田 憲人[1]

概要：NII では，SINET とクラウドを活用して再現性のあるアプリケーション環境の構築・運用を支援するオンデマンドクラウド構築サービスの提供を計画しており，基盤ソフトウェア VCP と Jupyter 形式の研究・教育アプリケーションの構築・運用手順書の開発を進めている．本稿では，VCP の利便性向上とより実用的なアプリケーションテンプレートの開発を支援するため，VCP を改良して (1) 利用者の管理機能，(2) モニタリング機能，(3) 安全なデータ管理支援機構，(4) クラウドプロバイダ対応モジュールのプラグイン機構の開発を行うとともに，(5)VCP での GPU インスタンス利用検証を行った．実験から，VCP で GPU インスタンスを利用することが可能であり，その性能も通常の利用形態と比較して遜色ないことを確認した．また，VCP を用いてゲノム解析環境をオンプレミス環境から SINET5 を介して商用クラウドへスケールアウトする実験について紹介し，実用性を示す．

Enhancement of an On-demand Configuration System for Inter-Cloud Applications

1. はじめに

高性能学術ネットワークとクラウド技術の普及により、インタークラウド [1] と呼ばれる複数のクラウド資源を利用した計算基盤の構築・利用が可能となった．国立情報学研究所（NII）が運用している SINET5[2], [3], [4] は，国内 800 組織以上の学術研究機関に接続された学術ネットワークであり，全都道府県のルータ間を 100Gbps で相互接続させた安全かつ広帯域低遅延のネットワークを提供するとともに，米国，欧州，アジアの学術ネットワークに対しても広帯域なネットワークで接続されている．また，最近では SINET5 を始めとする学術ネットワークから複数の商用クラウドデータセンタに直接接続することができるようになったため，学術研究機関と商用クラウドを仮想プライベートネットワーク（VPN）で接続させ，安全，安定，高性能なインタークラウド環境を構築できるようになってきた．

インタークラウド環境で効率よくアプリケーション環境を構築・運用するには，対象アプリケーション以外の様々な知識が必要となり，簡単ではない．具体的には，学術ネットワークとクラウド接続に関する経路制御設定，個々のクラウドプロバイダのサービスインタフェースの習得，利用する資源の最適化やスケールアウト等のクラウドらしい利用方法の適用など，多くの技術的課題がある．また，アプリケーション環境の構築・運用自体も，様々なライブラリの依存関係の解決やソフトウェアのバージョンアップへの対応など，簡単に行えるわけではない．

NII では，SINET とクラウドを活用して再現性のあるアプリケーション環境の構築・運用を支援するため，平成 30 年 10 月からオンデマンドクラウド構築サービス [5] の提供を計画している．オンデマンドクラウド構築サービスは，主に基盤ソフトウェアとアプリケーションテンプレートで構成される．基盤ソフトウェアは，SINET に接続された様々なクラウドデータセンターの資源上にアイソレートされた仮想計算基盤（仮想クラウドと呼ぶ）を構築する．アプリケーションテンプレートは，大学等で利用されている学習管理システム（LMS）等のアプリケーションの構築・運用手順をテンプレート化したものであり，仮想クラウドでのアプリケーション環境構築・運用を支援する．

[1] 国立情報学研究所
National Institute of Informatics
[2] 群馬大学
Gunma University
[a)] takefusa@nii.ac.jp

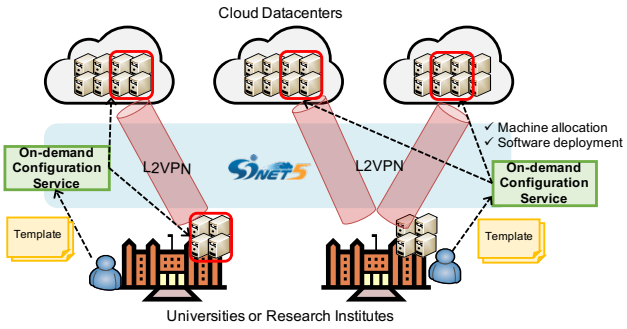

図 1　オンデマンドクラウド構築サービスの概要.

我々は，これまでの研究において基盤ソフトウェアである Virtual Cloud Provider（VCP）[6], [7] の開発と，e ラーニングプラットフォーム Moodle を始めとするアプリケーションテンプレート開発を進めている [8]．VCP の開発では，Docker[9] コンテナを利用した仮想クラウドを構築して再現性のあるゲノム解析環境が構築できることを示す [6] とともに，仮想クラウドの容易な制御を可能にするサービスインタフェースの提供と，SINET5 と商用・学術クラウドで構成される仮想クラウドの構築を可能にしてその基本性能を示した [7]．アプリケーションテンプレートでは，ブラウザベースのインタフェースを提供する Jupyter Notebook[10] を用いて計算機の操作，運用を行う手順書および実行環境を提供する．NII では，構築手順の再現性や情報共有を可能にする Literate Computing for Reproducible Infrastructure（LC4RI）という方法論に基づいて Jupyter Notebook を所内クラウド基盤の構築，運用に活用しており [11], [12], [13]，オンデマンドクラウド構築サービスでもこれを採用している．

しかしながら，VCP の利便性向上とより実用的なアプリケーションテンプレートの開発のために，利用者の認証認可，仮想クラウドのモニタリング，安全なデータの管理，利用可能なクラウドプロバイダの拡充，GPU インスタンスの対応等，VCP の機能拡張が必要となることが明らかになった．

本稿では，VCP を改良し (1) 利用者の管理機能，(2) モニタリング機能，(3) 安全なデータ管理支援機構，(4) クラウドプロバイダ対応モジュールのプラグイン機構を開発した．また，(5)VCP での GPU インスタンス利用検証を行い，GPU インスタンスも VCP から利用できることを確認した．実験から，VCP で GPU インスタンスを利用する際も通常の利用形態と比較して性能が遜色ないことを示す．また，VCP を用いてゲノム解析環境をオンプレミス環境から SINET5 を介して商用クラウドへスケールアウトする実験について紹介する．

2. オンデマンドクラウド構築サービスの概要

オンデマンドクラウド構築サービスは，SINET と学術・商用クラウドの資源を活用したアプリケーションの構築・運用を支援する．図 1 にオンデマンドクラウド構築サービスの概要を示す．オンデマンドクラウド構築サービスは，基盤ソフトウェアにより SINET の L2VPN で接続された 1 つ以上のクラウドデータセンターの資源に対して，仮想クラウドを構築し，アプリケーションテンプレートによりアプリケーション環境構築・運用を支援する．以降で基盤ソフトウェアの VCP とアプリケーションテンプレートについて説明する．さらに，利便性および実用性向上のための課題について述べる．

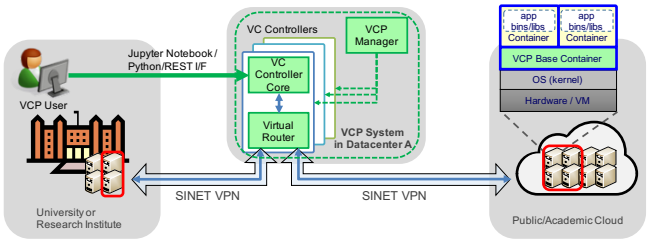

図 2　VCP の利用形態.

2.1 基盤ソフトウェア VCP

VCP は，SINET に接続された 1 つ以上のクラウドの物理計算機や仮想計算機等の資源上にアイソレートされた仮想クラウドを構築し，提供する．VCP では，軽量な仮想化を実現する Docker コンテナを用いることで，低性能オーバヘッドで容易にアプリケーション環境を配備できる．また，仮想クラウド内ネットワークの設定支援や異なるクラウド API の差異を吸収するサービスインタフェースを提供する．本インタフェースを用いることで，利用者は様々なクラウドに対して同様の操作で必要なノード構成の仮想クラウドの構築，停止，削除や，仮想クラウド内のノード（VC ノード）の追加・削除が行える．

2.1.1 VCP による仮想クラウドの提供

図 2 に，VCP の利用形態を示す．VCP は，主に VCP マネージャ（VCP Manager），仮想クラウド（VC）コントローラ（VC Controller），仮想ルータ（Virtual Router）で構成される．VCP マネージャは仮想クラウドを管理する VCP の利用者／グループに対して VC コントローラを配備する．VCP の利用者はそれぞれ異なる仮想プライベートネットワーク（VPN）を利用するため，VC コントローラも利用者毎に用意する．各利用者は，割り当てられた VC コントローラに対して VCP サービスインタフェースを介して仮想クラウドの構築を指示する．また，仮想ルータは大学等学術研究機関と複数商用クラウドデータセンター間のネットワークのハブ的な役割を担うとともに，外部ネットワークへのゲートウェイ機能および BGP による経路制御機能も備えている．クラウド間ネットワークは SINET5 の L2VPN を前提としているが，L2VPN 環境がない，また

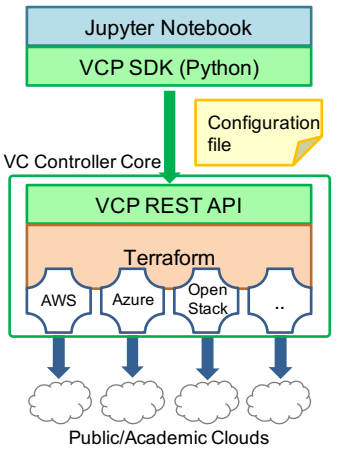

図 3　VCP サービスインタフェース.

は高性能ネットワークが必要ないアプリケーションを利用する利用者に対して，仮想ルータでは IPSec トンネルを用いたクラウド間 VPN を構築して利用できるようにしている．仮想ルータには，仮想スイッチ Open vSwitch（OVS）[14] とルーティングソフトウェア Quagga[15] を用いて実装している．

オンデマンドクラウド構築サービスでは，様々なクラウドで容易に仮想クラウドが構築できるようにすること，および様々なアプリケーションで公開されている既存の Docker イメージを利用できるようにすることが求められる．よって，図 2 の右側に示しているように，VCP による仮想クラウドへの VC ノード起動ではクラウドで確保した仮想／物理計算機（VM/BM）に対してベースコンテナとアプリケーションコンテナを二階層で配備して利用することを前提としている．このような利用形態は，Docker-in-Docker と呼ばれており，多少の性能劣化が懸念されるものの，柔軟なアプリケーションライブラリの配備を可能にする．ベースコンテナ層ではネットワーク設定や起動した VC ノードの死活監視を行う．アプリケーションコンテナ層では，目的のアプリケーションで必要とされるコンテナを配備する．ベースコンテナは各 VM または BM に 1 つ配備されるものであり，アプリケーションコンテナはベースコンテナ上に複数配備することができる．

2.2　VCP サービスインタフェース

VCP では，異なるクラウドプロバイダの API の差異を吸収するため，VC コントローラの内部でオープンソースの Terraform[16] を用いており，利用者に対しては VCP REST API と Python ベースの開発キット VCP SDK を提供している．図 3 にこれらのサービスインタフェースの概要を示す．

VCP REST API では，YAML 形式で記述された仮想クラウドの構成ファイル（Configuration file）を POST メソッドで取得すると，その構成ファイルに従って Terraform 経由で各クラウドへの資源確保等の制御を可能にする．Terraform では，各クラウドにおける VM やネットワークスイッチ，コンテナ等の資源を制御するモジュールを Provider と呼び，AWS（Amazon Web Services），GCP（Google Cloud Platform），Microsoft Azure，OpenStack 等の主要な Provider を予め提供されている．また，Terraform はプラグインベースで実装されており，Terraform の提供する API を実装することで独自の Provider を開発することができる．

```
# 仮想クラウド環境の初期化
vc = VcpSDK("vcp_accesskey", "my_vc_name")
# 作成する VC ノードの spec 情報をクラウド名と flaver"small"で作成
spec = vc.spec.find("aws", "small")
# spec で指定した VC ノードを起動
nodes = vc.unit.create("instance_name", spec)
# vc で管理するすべての VC ノードの削除
# vc.cleanup()
```

図 4　VCP SDK を用いたコード例.

```
### VCC 設定 ###
vcc:
    host: VC Controller IP
    name: VCC Name
### AWS の設定 ###
aws:
    access_key: "AWS のアクセスキー"
    secret_key: "AWS のシークレットキー"
    private_network: "default"
### さくらのクラウドの設定 ###
sakura:
    token: "さくらのクラウドのトークン"
    secret: "さくらのクラウドのシークレット"
    private_network: "default"
    :
```

図 5　VCP SDK 設定ファイルの例.

仮想クラウドの制御を容易にするため，Python ベースの開発キット VCP SDK も提供している．図 4 に VCP SDK を用いたコード例を示す．図 4 は，AWS に `small` フレーバの VC ノードを 1 つ起動する例を示している．利用者は，仮想クラウドの初期化（2 行目），VC ノード構成（`spec`）の指定（4 行目），VC ノードの起動（6 行目）の 3 行でクラウドに VC ノードを起動することができる．`spec` の指定箇所で，`aws` の代わりに `azure` と入力すると，Azure 上に VC ノードを起動することができる．フレーバには，各クラウドに対して予め `small`，`medium`，`large` のパラメータセットを定義しており，クラウド利用に不慣れなユーザにも容易に操作できるようにしている．また，ここで定義しているパラメータは上書き可能であるため，利用者が必要に応じてインスタンスタイプやディスクサイズ等の詳細な性能を定義することもできる．また，`cleanup()` というメソッドでその仮想クラウドが管理しているすべてのノードを削除することができる．ただし，AWS のアクセストー

クンなど制御に必要な情報は，図 5 に示すような YAML 形式の設定ファイルに予め格納しておく必要がある．これらの情報から，VCP SDK で構成ファイルを生成し，VCP REST API で VC コントローラに制御の要求を送信する．

2.3 アプリケーションテンプレート

アプリケーションテンプレートは，研究・教育目的で広く利用されているアプリケーションの構築・運用手順をテンプレート化したものを Jupyter Notebook 形式で提供する．これにより，クラウドでのアプリケーション構築に不慣れな利用者に対してクラウドの利活用を支援する．現時点では，以下のアプリケーションテンプレートの開発を進めており，引き続き利用者コミュニティと協力しながら開発を進めていく．

HPC テンプレート　OpenHPC[17] で提供されている科学技術計算のための計算ライブラリ群およびキューイングシステムからなるクラスタ環境を構築と，構築したクラスタでの基本的な性能評価が行える．

LMS テンプレート　moodle[18] を用いた学習管理システムを構築，ダウンタイムを短くすることを目的とした BlueGreenDeployment[19] という手法に基づくアップデート作業 [8] のための手順を記している．

VDI テンプレート　Guacamole[20] を用いた講義・演習用の仮想デスクトップ基盤（VDI）の構築と，スケールアウト／ダウンの手順を示している．

ゲノム解析テンプレート　Galaxy ワークフローツール [21] によるゲノムデータ解析環境をクラウド上に構築するものである．

2.4 利便性，実用性向上のための課題

アプリケーションテンプレートの開発とオンデマンドクラウド構築サービスの試験運用を進めるにあたり，利便性や実用性を向上させるためには以下のような技術的な課題があることが明らかになった．

VC コントローラの管理　VC コントローラは，同一の研究グループなど VPN にアクセス可能な複数の利用者で共有することになる．VC コントローラの管理者は，どの利用者に VC コントローラへのアクセスを許可し，どの利用者がどのような操作を行ったのか，管理する必要がある．また，このための管理者，利用者双方の手続きが煩雑になると利便性も下がってしまう．

仮想クラウドモニタリング　仮想クラウドを利用する際，クラウドでは確保する VC ノードの性能に比例して単位時間あたりの利用料が高くなるため，対象アプリケーションを運用するために十分な性能の VC ノードを利用することが求められる．そのため，どのプログラムがどの資源をどの程度消費するのかを的確に把握する必要があるが，個々の利用者がモニタリングツールをインストールするのは負担が大きい．

クラウドでの安全なデータ管理　SINET の VPN と各クラウドでのアイソレーション機能を利用することで安全に仮想クラウドを構築することができるが，起動したアプリケーション環境や VPN に接続された他の計算機における脆弱性等により，情報漏えいの危険にさらされる可能性がある．よって，クラウドで機密性の高い情報を扱うアプリケーションでは，ディスクボリューム自体を暗号化したより安全なデータの管理が求められる．

新規クラウドプロバイダへの対応　VCP では Terraform を用いて複数のクラウドプロバイダに対応しており，現状では AWS，Azure，GCP，さくらのクラウド，OpenStack に対応している．今後，より多くのクラウドプロバイダに対応するには，第三者により VCP 対応のクラウドプロバイダが開発できることが求められるが，Terraform でのプロバイダ開発だけでなく VCP REST API や VCP SDK でのコード修正も必要となり，簡単ではない．

アクセラレータの利用　機械学習や高性能計算分野では，低消費電力で高速な計算を行うために，GPU と CPU を併用する手法が用いられる．クラウドでも，主要なクラウドでは GPU が利用可能なインスタンスが既に提供されている．また，最近では FPGA をアクセラレータとして利用可能なインスタンスの提供も始まっている．しかしながら，これらは Docker コンテナから利用することができないため，このままでは VCP から利用することはできない．

3. VCP の機能拡張

オンデマンドクラウド構築サービスの利便性，実用性向上を目的とし，以下の機能を開発および検証を行った．

(1) VC コントローラ利用者管理機能
(2) 仮想クラウドモニタリング機能
(3) 安全なデータ管理支援機構
(4) クラウドプロバイダ対応モジュールのプラグイン機構
(5) VCP での GPU インスタンス利用検証

3.1 VC コントローラ利用者管理機能

VC コントローラ（VCC）の管理者に対し，その VCC の利用者の管理を支援するための機能として，管理者用 Web ユーザインタフェース（Web UI）と利用者の認証認可機構を開発した．図 6 に VCC 利用者の管理機能の概要を示す．管理者用 Web UI では，学術認証フェデレーション「学認（GakuNin）」[22] による認証認可を行う．学認は，日本国内の大学等と NII が連携して構築・運用している認証フェデレーション基盤であり，各機関がフェデレーションのポリシを信頼し合うことで相互の認証連携を実現している．

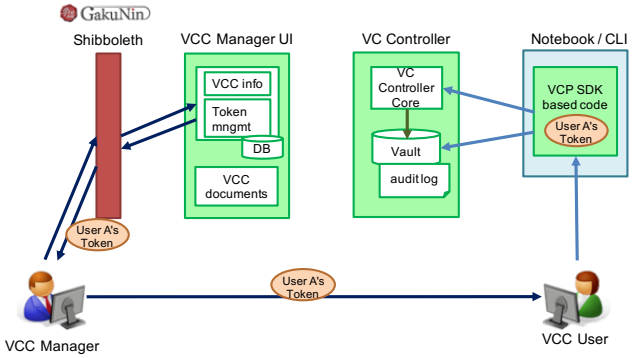

図 6　VC コントローラ利用者管理機能の概要.

本サービスの提供対象者は国内の大学等研究機関であるため，学認を利用することで独自の ID 管理が不要になったり，利用者はシングルサインオンが可能になるなどの利点がある．

ただし，個々の利用者に対する VCC 利用時の認証認可は学認ベースにするとプログラムからの認証が容易でないため，利用者に対してはアクセストークンベースでの認証認可を行う．VCC 管理者は，管理者用 Web UI から VCC 管理者が各ユーザに対して期限付きアクセストークンを発行する手続きを行い，発行したトークンを VCC 利用者に配布する．管理者による発行したトークンの削除，期限延長は適宜行うことができる．アクセストークンには，HashiCorp Vault（Vault）[23] を用いている．Vault は，トークンごとに秘匿情報を管理するための暗号化された Key Value Store（KVS）を提供する．VCC では，各利用者に対して別々のアクセストークンを発行して利用者を識別する．VCC 利用者は，配布されたアクセストークンを利用して VCC にアクセスする．VCC では，そのアクセストークンが有効なものかどうかを検証し，その利用者の要求の認証認可を行う．

3.2　仮想クラウドモニタリング機能

仮想クラウドのモニタリングを支援するため，VC コントローラにモニタリング機能を実装した．仮想クラウドのモニタリングでは，a) 各ベースコンテナでの VC ノード内の資源利用状況に関する情報（メトリクス情報）を取得し，b)VCC ごとにベースコンテナで取得したメトリクス情報を一元的に収集・管理し，c) その結果の可視化を行う．

モニタリング機能の実装では，a) では cAdvisor[24]，b) は Prometheus[25]，c) は Grafana[26] を用いた．既存のオープンソースソフトウェアを利用することで，汎用的なデータ形式と API をもつモニタリングシステムが短期間で実装することができる．

cAdvisor は Google が開発している Docker コンテナのモニタリングツールであり，コンテナ単位でメトリクス情報を取得することが可能である．各ベースコンテナ内に配備した cAdvisor により，VC ノード全体の利用状況だけでなく，その上で起動している複数のアプリケーションの利用状況もそれぞれ監視できる．

Prometheus は pull 型のモニタリングシステムであり，各ベースコンテナ内の cAdvisor からメトリクス情報を定期的に取得して管理する．VCC では VC ノードの起動状況を把握しており，また Prometheus と cAdvisor も連携しているため，Prometheus を用いた．

メトリクス情報の可視化では，Prometheus 自体も可視化ツールを提供しているが，より直感的な GUI を提供する Grafana を用いた．Grafana はデータソースとして Prometheus を指定することができ，ダッシュボードのカスタマイズも可能である．

3.3　安全なデータ管理支援機構

VCP の利用者に対して安全なデータ管理を支援するため，3.1 節で VC コントローラに導入した Vault を各 VCC の利用者の秘匿情報一時管理 KVS として利用できるようにした．これにより，VCP に関する操作と各アプリケーション環境の構築・運用に関する操作において次のように Vault を利用することができる．

VCP では，クラウドのアクセストークン等を含む設定情報を図 5 に示す設定ファイルや REST API で用いる構成ファイルに直接書き込む必要があるが，情報漏えいの恐れがある．よって，利用者はアクセストークン等の秘匿情報の代わりに Vault のキーを入力すれば，VCC 側でそのキーに紐づく秘匿情報を Vault から取得してクラウドに対する制御が行えるようにした．VCP では，アクセストークンごとに KVS の格納スペースを切り分ける HashiCorp Vault の Cubbyhole と呼ばれる方式の利用を前提としている．

アプリケーション環境の構築・運用に関する操作では，ssh やディスクボリューム暗号化に関する鍵情報等を Vault に格納して利用することができる．VCP では，VC ノード上でのアプリケーション環境構築・運用手順はスクリプトや Jupyter Notebook 内のコードで記述し，実行されることを想定している．しかし，上記のような秘匿情報を設定ファイルやコード自体に埋め込むことは VCP 操作時同様に情報漏えいの恐れがある．また，随時パスフレーズを入力するような利用方法では負担が大きい．そこで，VCC の機能として Vault を提供することとし，VCP に関する操作以外でも利用できるようにした．

2.3 節で紹介した Moodle を用いた LMS テンプレートでは，Moodle で個々の学生の成績など機密性の高い情報を管理するため，クラウドで確保するディスクボリューム全体を LUKS（Linux Unified Key Setup）で暗号化する手順を紹介している．これは，万が一クラウド側でそのディスクが盗難等にあった場合でも，そのままでは読み出せないようにしている．また，暗号化の鍵情報自体は VCC 側に

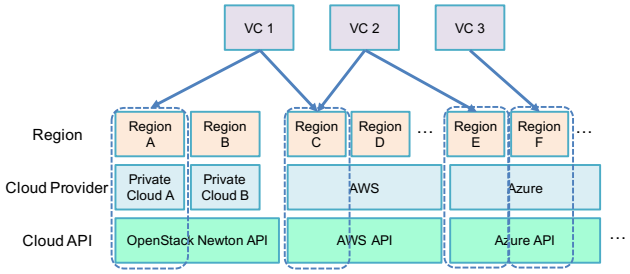

図 7　クラウド API，クラウドプロバイダ，リージョンの関係．

あるため，鍵とディスクを同時に盗まれる可能性は低い．

3.4　クラウドプロバイダ対応モジュールのプラグイン機構

VCP コードのリファクタリングを行い，第三者による VCP 対応クラウドプロバイダ対応モジュールの開発が容易に行えるようにした．図 7 にクラウド API，クラウドプロバイダ，リージョンの関係を示す．Cloud API 層では，異なるクラウド API を表している．Cloud Provider 層では，直下の Cloud API を提供するクラウドプロバイダを表している．Region 層では，直下のクラウドプロバイダで利用可能なリージョン情報が定義されている．各仮想クラウド（VC）では，矢印が指し示す破線の四角で囲まれたリージョン，クラウドプロバイダ，クラウド API の組み合わせを利用することを表している．

新たなクラウドプロバイダを VCP で利用可能にするためには，図 7 の Cloud API 層と Cloud Provider 層で必要とされるコードやスキーマファイルからなるプラグインモジュールを実装または定義すればよい．必要なコードおよびスキーマファイルは以下の通りである．

Cloud API 層　VCP が規定している SPI（Service Provider Interface）に従って，コアとなるプラグインモジュールを開発する．また，Terraform 用テンプレートファイルを規定する．

Cloud Provider 層　各クラウドプロバイダで利用可能なインスタンスの性能情報パラメータと YAML スキーマで定義する．また，YAML 形式で対象クラウドにおける `small/medium/large` の各フレーバのデフォルトパラメータ値を定義する．さらに，対象クラウドプロバイダに紐づく Cloud API を指定する．

VCP SDK　図 7 には含まれていないが，VCP SDK のクラスの追加も必要となる．Cloud Provider 層で規定した性能情報パラメータの setter/getter メソッドを必要に応じて定義する．

本プラグイン機構を用いて，新たに GCP と OpenStack ベースのプライベートクラウドのためのプラグインモジュールを開発し，正常に動作することを確認した．GCP プラグインの追加では，上記の 3 つの項目に関する実装・定義を行った．プライベートクラウドプラグインの追加では，既存の Cloud API が利用可能であるため，Cloud API 層以外の 2 項目について実装・定義を行った．

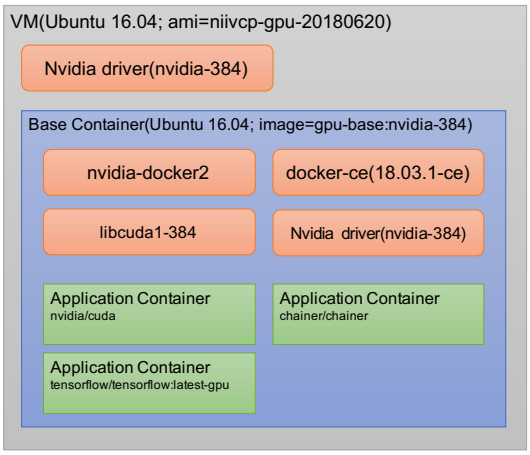

図 8　GPU 対応 Docker-in-Docker 構成．

3.5　VCP での GPU インスタンス利用検証

本研究では，Docker-in-Docker 方式での GPU インスタンスへのベースコンテナおよびアプリケーションコンテナの配備の検証と，VCP への適用を行った．

コンテナ上で GPU を利用するには，NVIDIA 社が開発した NVIDIA Docker（nvidia-docker）[27] を利用する方法がある．これにより，GPU 利用環境の分離や再現性のある実行環境の構築が容易になった．また，Open Container Initiative（OCI）[28] で規定された仕様に準拠して再実装された nvidia-docker2 により，docker-compose や swarm など通常の Docker で利用可能なツール群との親和性も高くなった．しかしながら，NVIDIA Docker による Docker-in-Docker 形式の利用方法は十分に検証されていない．

VCP では，VC ノードを起動するにあたり各クラウドのインスタンス用のマシンイメージとベースコンテナイメージを予め用意している．GPU コンテナに対応するため，GPU コンテナ対応マシンイメージおよびベースコンテナイメージの作成と，GPU 対応ベースコンテナ上でアプリケーションコンテナを用いたの機械学習プログラムの実行を確認した．図 8 に，構築した Docker-in-Docker の構成を示す．灰色の四角は VM，青色の四角はベースコンテナ，緑色の四角はアプリケーションコンテナを表しており，階層的に配備されている．また，橙色の丸四角はインストールされているライブラリを表しており，nvidia-docker2 等はベースコンテナ内にインストールされている．Nvidia driver は VM またはベースコンテナのいずれかにインストールされていればよいが，ドライバをインストールする OS のカーネルバージョンは，ホスト VM とコンテナとで一致しなければならないという制約は残る．

アプリケーションコンテナでは，深層学習フレームワークの Chainer[29] や TensorFlow[30] に対応した nvidia-doker2

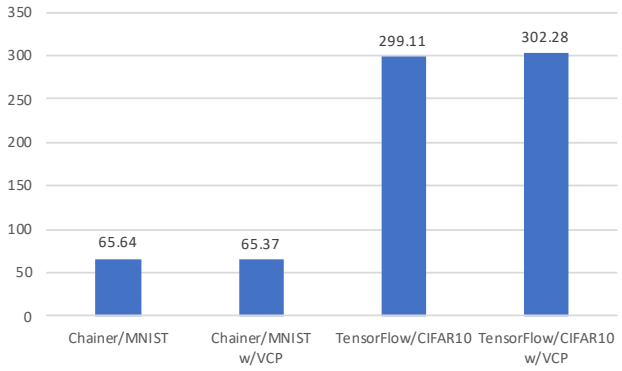

図 9　AWS における学習時間の比較.

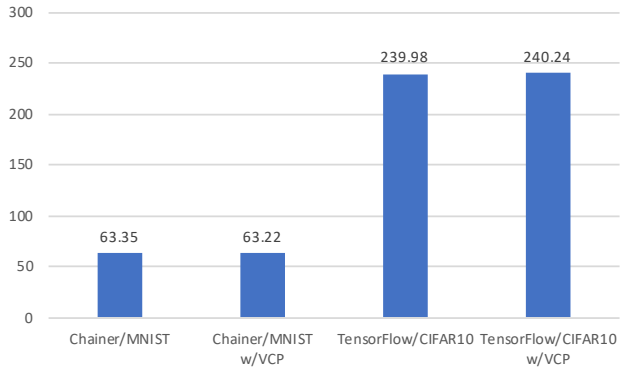

図 10　Azure における学習時間の比較.

コンテナを配備し，その上で正常に機械学習処理が行えることを AWS および Azure で確認した．また，これらのマシンイメージおよびベースコンテナイメージを VCP 側に登録して必要なパラメータを指定することで，GPU 対応ベースコンテナ配備した VC ノードを AWS および Azure で VCP から起動できることを確認した．

4. GPU 対応 VC ノードの基本性能

nvidia-docker2 を VCP で Docker-in-Docker 形式で実行する際の性能オーバヘッドを調査するため，Chainer および TensorFlow を用いた評価実験を AWS および Azure で行った．実験では，AWS と Azure では利用できる GPU が異なるため，AWS および Azure において VC ノードを起動して利用する場合（ベースコンテナあり）と，通常のインスタンスをそのまま利用する場合（ベースコンテナなし）でそれぞれ比較する．起動した各ノードでは，必要なアプリケーションコンテナを配備して，それぞれの環境で機械学習プログラムを実行する．

表 1，表 2 に本実験で用いた計算機とアプリケーションの構成を示す．各クラウドでは，VCP の有無に関わらず同じ構成の計算機を利用することとした．また，アプリケーションの構成は Chainer，TensorFlow ともすべての環境で同じ組み合わせのものを用いた．Chainer では MNIST[31]，TensorFlow では CIFAR10[32] のデータセットを用いた機械学習サンプルプログラムが提供されている．MNIST は 28×28 画素の手書き数字からなるデータセットである．Chainer のサンプルプログラムを用いて，600 イタレーション ×20 エポック分の学習時間を測定した．CIFAR10 は 32×32 画素の画像が飛行機等の 10 種類のクラスに分類されているデータセットである．TensorFlow サンプルプログラムを用いて，1 万ステップの学習に要する時間を測定した．

図 9，図 10 に AWS および Azure において各学習に要した時間をを示す．いずれも 10 回の平均値であり，各学習時間の標準偏差は 1.0 未満であった．AWS，Azure での学習時間の差は，利用している GPU の世代が異なることによる．また，w/VCP は Docker-in-Docker 構成の VC ノードの実行結果である．Chainer と MNIST を用いた実験の結果では，AWS，Azure とも VC ノードの方が学習時間が僅かに短かく，VCP によるオーバヘッドは誤差の範囲であることが分かる．一方，TensorFlow と CIFAR10 を用いた実験の結果では，VC ノードの方が AWS では 1.06%，Azure では 0.11%学習時間が長くなっていたが，許容できることが確認できた．

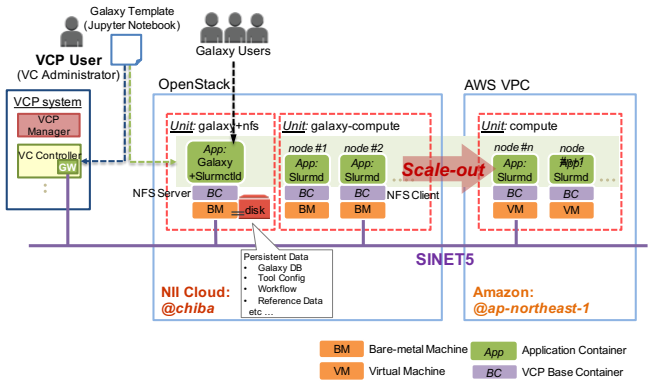

図 11　Galaxy スケールアウト実験環境.

5. Galaxy ゲノム解析環境スケールアウト実験

VCP の実用性を示すため，Galaxy とクラスタ計算機を用いたゲノム解析環境をオンプレミス（NII 所内クラウド）環境から商用クラウドの AWS に SINET5 を介してスケールアウトする実験を SC17 の展示会場で実施した．NII 所内クラウドは OpenStack ベースのベアメタルクラウドであり，VCP から制御することができる．よって，オンプレミス環境と商用クラウド環境を VCP を用いて構築する．

図 11 に実験環境を示す．VCP のユーザは，図 12 に示す Jupyter Notebook の手順に従って以下の作業を行い，Galaxy 環境のスケールアウトを実証した．

1. VCP を用いてオンプレミス環境にフロントエンド用の VC ノード 1 台と計算用 VC ノード 2 台を起動する．フロントエンドノードは NFS サーバの機能を担うため，適切なサイズのデータボリュームをアタッチして

表 1　実験で用いた計算機の構成.

実験環境	AWS, AWS-VCP	Azure, Azure-VCP
OS	Ubuntu 16.04.4 LTS	Ubuntu 16.04.4 LTS
カーネル	v. 4.4.0-128-generic	v. 4.13.0-1018-azure
CPU	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz	Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz
仮想 CPU 数	4	6
メモリ	59 GiB	55 GiB
インスタンス	p2.xlarge	Standard_NV6
GPU	Tesla K80 × 1	Tesla M60 × 1
NVIDIA ドライバ	v. 384.130	v. 384.130

表 2　各環境で用いたアプリケーションの構成.

	Chainer	TensorFlow
コンテナ	chainer/chainer	tensorflow/tensorflow
イメージ	:v4.2.0-python3	:1.8.0-gpu-py3
OS	Ubuntu 16.04.4 LTS	Ubuntu 16.04.4 LTS
Python	v. 3.5.2	v. 3.5.2
CUDA	v. 8.0.61	v. 9.0.176
cuDNN	v. 6.0.21	–

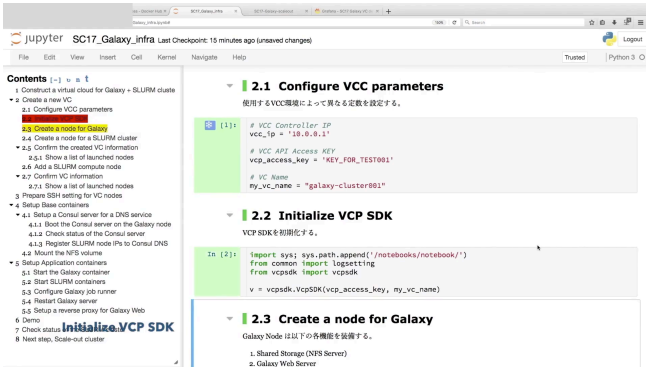

図 12　Galaxy スケールアウト実験用 Jupyter Notebook のスナップショット.

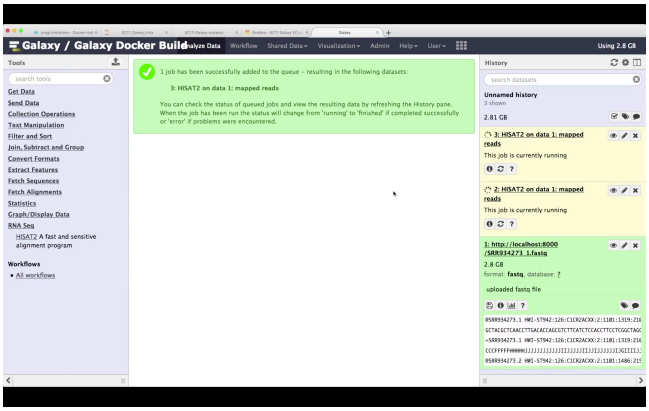

図 13　Galaxy ワークフローツールからのツール（ジョブ）の実行時のスナップショット.

おく．起動した VC ノードは，利用者からは通常の計算ノードとして見える．

2. 起動したノード群の上に，アプリケーションコンテナの配備を必要な設定を行っていく．フロントエンドノードでは，Galaxy ワークフローツールとバッチキューイングシステム SLURM のコントローラのイメージを配備する．計算ノードでは，SLURM の計算ノード用イメージを配備し，オンプレミスの Galaxy 環境を構築する．

図 14　オンプレミス環境のみ利用した際のモニタリング結果.

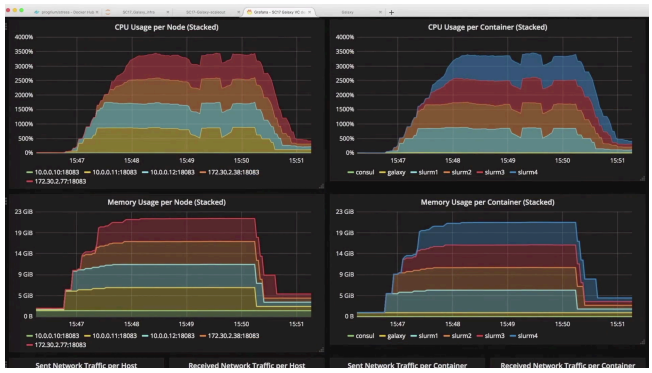

図 15　オンプレミス環境から AWS へスケールアウト後のモニタリング結果.

3. 構築した図 13 の Galaxy ポータルにアクセスし，Galaxy から投入されたジョブが各計算ノードで実行できることを確認する．図 14 は，オンプレミス環境で 2 つのジョブを実行した際のモニタリング結果を表している．左側はベースコンテナのメトリクス情報，右側はアプリケーションコンテナのメトリクス情報であり，アプリケーションコンテナ内でジョブを実行しているために同様の結果となっている．
4. オンプレミス環境の計算資源が足りなくなったという前提で，SINET5 から直接接続された AWS の VPC（Virtual Private Cloud）内に VCP を用いて計算用

VC ノードを 2 台起動する．

5. 2. での計算ノードの設定と同じ手順で，AWS 内の VC ノードに計算ノード用のアプリケーションコンテナを配備する．計算ノードの NFS サーバは，オンプレミスのものを利用する．
6. 3. と同様に Galaxy から 4 つのジョブを投入し，オンプレミスと AWS の 4 つの計算ノードで各ジョブが実行されることを確認する．図 15 は，スケールアウト後の環境で 4 つのジョブを実行した際のモニタリング結果を表している．AWS の計算ノードはオンプレミス環境の NFS サーバを利用しているため，ピークに達するまでに時間に多少の遅延があるものの，1 つの SLURM クラスタとして機能していることが確認できる．

6. 関連研究

InterCloud[33] や Contrail[34] などのインタークラウドに関する研究プロジェクトでは，複数クラウドのフェデレートして SLA ベースのブローカリングサービスを提供する．本研究が対象とするサービスでは，利用者が指定したクラウド上の資源の利用を容易にし，アプリケーション環境構築を支援することを目的としており，ブローカリングサービスやアロケーション手法に関する研究はスコープ外である．

異なるクラウドを制御するためのツールとして，AWS CloudFormation[35]，OpenStack Heat[36]，Apache Libcloud[37] がある．これらは，異なるクラウドの API の差異を吸収することを目的としている．我々は，クラウド API の差異の吸収には Terraform を用いており，利用者向けには VCP REST API と，クラウドでの環境構築に不慣れな利用者に対しても使いやすい VCP SDK を提供している．RightScale[38] では，ボタンひとつでクラウド上にアプリケーション環境を構築する商用サービスを提供している．アプリケーション環境構築を支援する点で類似性があるが，我々は Jupyter Notebook を用いたホワイトボックス的アプローチをとっており，環境設定のカスタマイズやエラー発生時の原因究明がしやすいなどの利点がある．

インタークラウド環境を提供する試みとして，GTS[39]，Skyport[40]，CYCLONE[41] がある．GTS（GÉANT Testbed Service）は，分散する OpenStack ベースの計算資源や SDN スイッチ，およびその間のネットワークを統合するためのソフトウェアスタックを開発している．Skyport は，彼らの開発しているデータ解析プラットフォーム AWE/Shock 上のプログラム実行環境を Docker を用いて配備する．これらは対象とする計算基盤が特定されている点で本研究と異なる．CYCLONE は本研究同様，アプリケーション環境を複数クラウド資源上に構築するシステムである．アプリケーション環境の配備では，本研究では軽量な Linux コンテナを利用するのに対し，CYCLONE では VM イメージを用いている．

7. まとめ

オンデマンドクラウド構築サービスの提供に向け，基盤ソフトウェア VCP と Jupyter 形式の研究・教育アプリケーションの構築・運用手順書の開発を進めている．本稿では，VCP の利便性向上とより実用的なアプリケーションテンプレートの開発を支援するため，VCP を改良して (1) 利用者の管理機能，(2) モニタリング機能，(3) 安全なデータ管理支援機構，(4) クラウドプロバイダ対応モジュールのプラグイン機構の開発を行うとともに，(5)VCP での GPU インスタンス利用検証を行った．実験から，VCP で GPU インスタンスを利用することが可能であり，その性能も通常の利用形態と比較して遜色ないことを確認した．また，VCP を用いてゲノム解析環境をオンプレミス環境から SINET5 を介して商用クラウドへスケールアウトする実験について紹介し，実用性を示した．今後は，VC コントローラの監査機能の拡充や FPGA 等の他のアクセラレータへの対応，アプリケーションテンプレートやハンズオンパッケージの拡充などを行っていく．

謝辞 本研究にご協力いただいた株式会社アスケイドの那須野淳様，羽鳥文子様，増山隆様，数理技研の小泉敦延様に深く感謝いたします．

本研究の一部は，JST CREST「ビッグデータ統合利活用のための次世代基盤技術の創出・体系化」JPMJCR1501 の助成を受けたものである．

参考文献

[1] GICTF: インタークラウドのユースケースと機能要件，*GICTF White Paper*, pp. 1–31 (2010).
[2] SINET, `https://www.sinet.ad.jp/` (accessed on 01-07-2018).
[3] S. Urushidani and S. Abe and K. Yamanaka and K. Aida and S. Yokoyama and H. Yamada and M. Nakamura and K. Fukuda and M. Koibuchi and S. Yamada : New Directions for a Japanese Academic Backbone Network, *IEICE Transactions on Information and Systems, E98.D (3)*, pp. 546–556 (2015).
[4] Kurimoto, T., Urushidani, S., Yamada, H., Yamanaka, K., Nakamura, M., Abe, S., Fukuda, K., Koibuchi, M., Ji, Y., Takakura, H., and Yamada, S.: A fully meshed backbone network for data-intensive sciences and SDN services, *Proc. ICUFN2016*, pp. 909–911 (2016).
[5] 学認クラウド，`https://cloud.gakunin.jp/` (accessed on 01-07-2018).
[6] S. Yokoyama and Y. Masatani and T. Ohta and O. Ogasawara and N. Yoshioka and K. Liu and K. Aida: Reproducible Scientific Computing Environment with Overlay Cloud Architecture, *Proc. 9th IEEE Cloud*, pp. 774–781 (2016).
[7] Takefusa, A., Yokoyama, S., Masatani, Y., Tanjo, T., Saga, K., Nagaku, M. and Aida, K.: Virtual Cloud Service System for Building Effective Inter-Cloud Applica-

tions, *Proc. IEEE CloudCom2017*, pp. 296–303 (2017).
[8] 浜元信州，横山重俊，竹房あつ子，合田憲人，桑田義隆，石坂徹：Moodle 運用における Jupyter Notebook の活用，*MoodleMoot Japan 2018* (2018).
[9] Docker, `https://www.docker.com/` (accessed on 01-07-2018).
[10] Jupyter Notebook, `http://jupyter.org/` (accessed on 01-07-2018).
[11] Masatani, Y.: Collaboration and automated operation as literate computing for reproducible infrastructure, `https://conferences.oreilly.com/jupyter/jup-ny/public/schedule/detail/59995` (accessed on 01-07-2018) (2017).
[12] 政谷好伸，谷沢智史，横山重俊，吉岡信和，合田憲人：インフラ・コード化の実践における IPython notebook の適用，信学技報, SC2015-6，pp. 27–32 (2015).
[13] NII Cloud Operation Team, `https://github.com/NII-cloud-operation/` (accessed on 01-07-2018).
[14] Open vSwitch, `http://openvswitch.org/` (accessed on 01-07-2018).
[15] Quagga Routing Suite, `https://www.quagga.net/` (accessed on 01-07-2018).
[16] Terraform, `https://www.terraform.io/` (accessed on 01-07-2018).
[17] OpenHPC, `https://openhpc.community/` (accessed on 01-07-2018).
[18] Moodle, `https://moodle.org/` (accessed on 01-07-2018).
[19] BlueGreenDeployment, `https://martinfowler.com/bliki/BlueGreenDeployment.html` (accessed on 01-07-2018).
[20] Apache Guacamole, `https://guacamole.incubator.apache.org/` (accessed on 01-07-2018).
[21] Galaxy, Data intensive biology for everyone, `https://galaxyproject.org/` (accessed on 01-07-2018).
[22] 学術認証フェデレーション，`https://www.gakunin.jp/` (accessed on 01-07-2018).
[23] HashiCorp Vault, `https://www.vaultproject.io/` (accessed on 01-07-2018).
[24] cAdvisor, `https://github.com/google/cadvisor` (accessed on 01-07-2018).
[25] Prometheus, `https://prometheus.io/` (accessed on 01-07-2018).
[26] Grafana, `https://grafana.com/` (accessed on 01-07-2018).
[27] NVIDIA Docker, `https://github.com/NVIDIA/nvidia-docker` (accessed on 01-07-2018).
[28] Open Container Initiative, `https://www.opencontainers.org/` (accessed on 01-07-2018).
[29] Tokui, S., Oono, K., Hido, S. and Clayton, J.: Chainer: a Next-Generation Open Source Framework for Deep Learning, *Proc. Workshop on Machine Learning Systems (LearningSys) in NIPS*, pp. 1–6 (2015).
[30] TensorFlow, `https://www.tensorflow.org/` (accessed on 01-07-2018).
[31] LeCun, Y., Cortes, C. and Burges, C. J.: The MNIST Database of handwritten digits, `http://yann.lecun.com/exdb/mnist/` (accessed on 01-07-2018).
[32] Krizhevsky, A.: The CIFAR-10 dataset, `https://www.cs.toronto.edu/~kriz/cifar.html` (accessed on 01-07-2018).
[33] Buyya, R., Ranjan, R. and Calheiros, R. N.: InterCloud: Utility-oriented Federation of Cloud Computing Environments for Scaling of Application Services, *Proc. ICA3PP'10*, Springer-Verlag, pp. 13–31 (2010).
[34] M. Coppola and P. Dazzi and A. Lazouski and J. Jensen and I. Johnson and P. Kershaw and P. Mori and F. Martinelli: The CONTRAIL approach to Cloud Federations, *Proc. Science - International Symposium on Grids and Clouds 2012 (019)*, pp. 1–14.
[35] AWS CloudFormation, `https://aws.amazon.com/cloudformation/` (accessed on 01-07-2018).
[36] OpenStack Heat, `https://wiki.openstack.org/wiki/Heat` (accessed on 01-07-2018).
[37] Apache Libcloud, `https://libcloud.apache.org/` (accessed on 01-07-2018).
[38] RightScale, `https://www.rightscale.com/` (accessed on 01-07-2018).
[39] GTS (GÉANT Testbed Service), `https://www.geant.org/Services/Connectivity_and_network/GTS/` (accessed on 01-07-2018).
[40] W. Gerlach and W. Tang and K. Keegan and T. Harrison and A. Wilke and J. Bischof and M. D'Souza and S. Devoid and D. Murphy-Olson and N. Desai and F. Meyer: Skyport - Container-Based Execution Environment Management for Multi-Cloud Scientific Workflows, *Proc. IEEE DataCloud '14*, pp. 25–32 (2014).
[41] D. Gallico and M. Biancani and C. Blanchet and M. Bedri and J. F. Gibrat and J. I. A. Baranda and D. Hacker and M. Kourkouli: CYCLONE: A Multi-cloud Federation Platform for Complex Bioinformatics and Energy Applications (Short Paper), *Proc. IEEE CloudNet*, pp. 146–149 (2016).