Vol.2022-HPC-185 No.17
2022/7/28

セキュアなAI/HPCクラウドバースティング実現に向けた検討

滝澤 真一朗[1]　清水 正明[2]　中田 秀基[1]　松葉 浩也[2]　高野 了成[1]

概要：機密性の高い情報の安全な処理を担保しつつ計算資源への投資を最適化する手法として、オンプレミス環境を保持したうえで、必要に応じてクラウド環境を追加的に使用する**クラウドバースティング**が普及しつつある。われわれはクラウドストレージをジョブサブミッション機構として使用してユーザレベルでのクラウドバースティングを実現する CloudQ を提案し、バースト先として HPC 環境である ABCI をターゲットとした実装についてすでに報告している。本稿では、CloudQ システムの対象としてパブリッククラウド環境を使用するシステムについて報告する。パブリッククラウドを用いる際には、ABCI のような HPC 環境とは異なるセキュリティおよび利便性に関する配慮が必要である。われわれは 2 つの方針を立案し、それぞれ実装を行った。2 つの方針の得失を議論するとともに、この 2 つの実装について詳述する。

Shin'ichiro TAKIZAWA[1]　Masaaki SHIMIZU[2]　Hidemoto NAKADA[1]　Hiroya MATSUBA[2]
Ryousei TAKANO[1]

1. はじめに

研究分野における計算資源としてパブリッククラウド環境の利用が広まっている。しかし機密性が高く組織内から出すことのできない情報を処理するためには、オンプレミスにも最低限の処理環境を維持する必要がある。このため、オンプレミス環境とパブリッククラウド環境をシームレスに結合し、オンプレミスを基調としながらも、必要に応じてパブリッククラウドを使用する**クラウドバースティング**と呼ばれる手法の研究が進んでいる [1][2] [3]。

われわれはクラウドストレージをジョブサブミッション機構として使用してクラウドバースティングを実現する CloudQ を提案している。CloudQ に関しては、バースト先として HPC 環境である ABCI をターゲットとした実装についてすでに報告した [4]。

本稿では、バースト先としてパブリッククラウド環境を使用するシステムについて報告する。ABCI のような HPC 環境では、リソースは共有資源として常に存在することを前提とすることができる。一方でパブリッククラウド環境ではジョブ実行環境を常に維持することにはコストが伴うため環境の動的な制御が必要となる。また、課金管理やユーザの利便性の観点でのトレードオフを考慮する必要がある。われわれは以下の 2 つの方針を立案し、それぞれ実装を行った。

- 個々のユーザが個別に実行環境を構築する方法
- 管理者がマルチユーザ実行環境を提供する方法

本稿では、これらをそれぞれ CloudQ/S(CloudQ Single) および CloudQ/M(CloudQ Multi) と呼ぶ。われわれは、設計におけるトレードオフを議論し、この 2 つのシステムについて詳細に紹介する。

以降の本稿の構成は以下のとおりである。2 節では研究の背景として既発表の CloudQ の概要を述べるとともに、パブリッククラウドとして使用した AWS が提供する機構について説明する。3 節ではパブリッククラウド環境を使用する際に留意すべき点を整理し、2 つの方針の得失を議論する。4 節でシングルユーザアプローチでの実装について述べる。5 節でマルチユーザアプローチでの実装について述べる。6 節で関連研究について述べる。7 節で結論と今後の課題について述べる。

2. 背景

2.1 CloudQ の概要

CloudQ は、ジョブサブミッションにクラウドストレージを使用することを特徴とするクラウドバースティングの

[1] 国立研究開発法人 産業技術総合研究所
[2] 株式会社 日立製作所

ためのジョブ投入システムである。ABCI[5] を対象とした CloudQ については、すでに [4] で詳細に報告しているが、ここで概説する。

2.1.1 クラウドストレージを用いた通信

計算機にジョブを投入する際に最も問題となるのはユーザの認証である。HPC 分野では ssh が広く用いられているが、ssh による接続を許すことは一般にそのユーザのすべての権限を許可することになり、場合によっては過剰である[*1]。また、クライアントの秘密鍵に対して有効期限を指定することはできない。

独自のジョブサブミッションサーバを構築し、認証を実装することも可能ではあるが、多くの HPC システムでは外部からのネットワーク接続を許可していないため、現実的ではない。また、セキュリティ的に万全な独自サーバを構築することはそもそも非常に困難である。

そこでわれわれは、広く普及している S3 互換のクラウドストレージに着目した。S3 互換クラウドストレージはユーザ ID とトークンで認証を行う。このトークンには、有効期限を指定することができる他、サーバ側で容易に無効化することもできる。トークンによって与えられる権限は、ファイルをストレージ上に置くことだけなので、過剰な権限を付与することにもならない。このようなクラウドストレージを、ジョブをサブミッションするユーザと、HPC システム側で動作するエージェントプロセスの通信回線とすることで、ユーザレベルでのセキュアな通信が可能となる。

ここで重要な点は、データを送信する側と受信する側の双方とも、ネットワーク接続は、クラウドストレージへの外向き通信となることである。通信主体のいずれも外部に対してネットワークポートを開くことなく、データ通信が可能となる。

2.1.2 CloudQ の動作

CloudQ は、バッチキューイングシステムが動作する HPC 計算環境で、外部からのジョブサブミッションを実現する機構である。ジョブや入力データは、クラウドストレージにアップロードする。HPC 計算機のヘッドノードで、ユーザ権限で動作するエージェントが、クラウドストレージを監視し、ジョブを実行する。CloudQ の特徴は、1) クラウドストレージバケットをジョブサブミッションに用いることで、機能の豊富なクラウドストレージトークンを認証に利用すること、2) ユーザレベルで完結するため、サーバ管理者の補助が不要であること、である。

CloudQ を利用するユーザはまず、クラウドストレージにジョブサブミット用のバケットを作成する。次に、HPC 計算システムにログインし、自分の権限でエージェントを起動しておく。この際にジョブサブミット用のバケットを

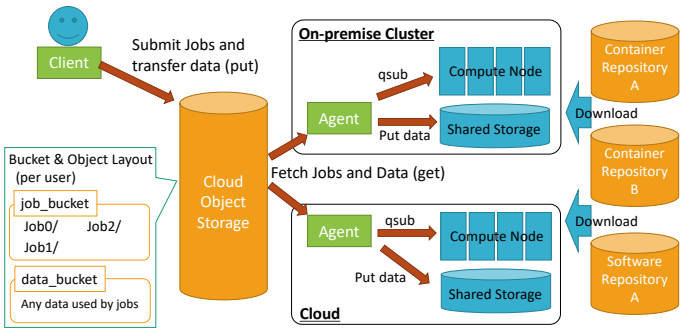

図 1 CloudQ

指定する。これらの作業は、(システムの再起動などでエージェントが停止しない限り) 一度行うだけでよい。

ジョブを投入するには、クライアントコマンドを利用する。クライアントコマンドは設定ファイルで指定されたバケットに、同様に指定されたトークンを用いて、ジョブスクリプトをアップロードする。

エージェントは定期的に指定されたクラウドストレージバケットを監視し、ジョブが存在すれば、それを引き出してジョブサブミッションシステムに投入する。その後はジョブの実行ステータスを監視し、ジョブの実行が終われば、結果として生成されたファイルをジョブバケットに書き出す。

ユーザは、クライアントコマンドでジョブの状態を確認することができる。ジョブの終了が確認できたら、クライアントコマンドを用いて結果のファイルをダウンロードする。

2.2 Amazon Web Services の機能

本稿のシステムではパブリッククラウドとして Amazon Web Services(以下 AWS) を使用し、AWS のさまざまな機能を利用している[*2]。本節ではその一部について簡単に説明する。

2.2.1 CloudFormation

AWS CloudFormation[6] は、インフラストラクチャの構成を再現可能なコードとして記述し、それを用いたインフラストラクチャ管理を行う IaC(Infrastructure as Code) を、AWS 上で実現する枠組みである。

CloudFormation では、インフラストラクチャの構成定義は JSON もしくは YAML で行う。この構成定義をテンプレートと呼ぶ。テンプレートを S3 バケットにアップロードし、CloudFormation にトリガをかけることで、インフラストラクチャを構築することができる。

CloudFormation では動作中の環境に対して、動的に変更を行うこともできる。これによって負荷に応じた動的なノードの増減が実現できる。

[*1] sshd の機能としては、authorized_keys ファイルに、個々のキーペアに対して実行可能なコマンドを制約することも可能であるが、ABCI では ssh キー管理の都合上この機能を利用できない。

[*2] Amazon Web Services およびその他の AWS 商標は，米国およびその他の諸国における Amazon.com,Inc. またはその関連会社の商標です。

2.2.2 ParallelCluster

AWS ParallelCluster[7] は、バッチキューイングシステムが設定されたクラスタを半自動的に AWS 上にデプロイする機構である。基本的にはユーザのクライアント側で動作するツールで、前述する CloudFormation を用いてデプロイを行う。バッチキューイングシステムとしては Slurm と AWSBatch がサポートされている。

ユーザは YAML 形式でコンフィギュレーションファイルを記述する。また、テキストベースのウィザードが用意されており、これに返答するだけで基本的なコンフィギュレーションファイルが生成できる。

ParallelCluster はジョブに対するオートスケールに対応している。すなわちリクエストに応じて計算ノードを起動し、不要になればシャットダウンする。このため、無駄に計算機使用料を支払う必要がない。

2.3 Amazon FSx for Lustre

Amazon FSx[8] は、AWS 上で提供されるフルマネージドな共有ファイルシステムである。バックエンドとして現在 NetApp ONTAP、OpenZFS、Windows File Server、Lustre を選択することができる。

Lustre をバックエンドとした場合には、Lustre の HSM(Hierarchical Storage Management) [9] を、S3 を二次ストレージとして指定して、使用することができる。HSM とは、高速で高価な一次ストレージと低速で安価な二次ストレージを自動的に使い分ける機能で、これを使用することでストレージの使用費用を低減することができる。

さらに、二次ストレージとして S3 を使用すると、二次ストレージ上のデータに S3 のプロトコルを用いて外部からアクセスすることが可能になる。

3. パブリッククラウドを対象としたクラウドバースティングの設計

パブリッククラウドをクラウドバースティングに使用する際に留意すべき点はセキュリティである。クラウド環境であっても、オンプレミスと同等のセキュリティ対策を施す必要がある。

その上で、ユーザ管理と課金管理を実現しなければならない。パブリッククラウド上にオンプレミス環境と同様のユーザ空間を構築するのは一般に容易ではない。オンプレミス環境でのユーザ登録、削除をパブリッククラウド側に反映する必要がある。課金管理とは個々のユーザの資源使用量に対して適切に課金を行うことである。これには個々のユーザの資源使用量を厳密にトラックする必要がある。

これらの観点から、パブリッククラウドをクラウドバースティングに用いる際には、大きく 2 つの方針が考えられる。一つは、オンプレミスと同様のマルチユーザ環境をパブリッククラウド上にも構築する方法、もう一つは個々のユーザに対して個別にパブリッククラウドを用意する方法である。ここでは、前者をマルチユーザ法、後者をシングルユーザ法と呼ぶ。

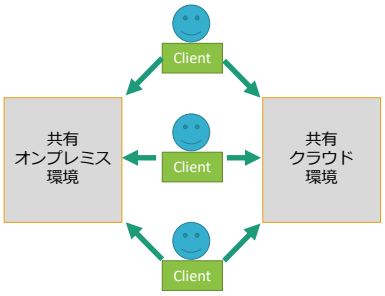

図 2 マルチユーザ法

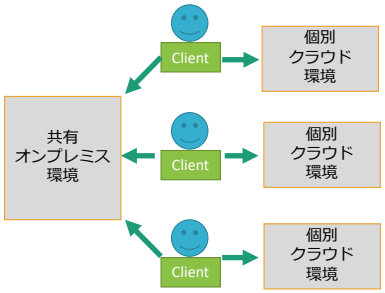

図 3 シングルユーザ法

マルチユーザ法では、オンプレミス環境と同じユーザ空間を持ったクラスタをクラウド上に構築し半永続的に維持する (計算ノードは必要に応じて増減する)。このクラウドをすべてのユーザで共有する (図 2)。ユーザ空間の同期にはアクティブディレクトリなどを用いる。課金管理に必要な情報は、クラウド上に構築したバッチスケジューラのログから取得する。セキュリティに関しては、基本的にオンプレミスと同じアプローチで実現する。いわば正攻法で防備を固めることでセキュリティを担保する。

シングルユーザ法では、個々のユーザが必要に応じてパブリッククラウド上にバッチスケジューラを持つ専用の実行環境を起動する (図 3)。個々のユーザがそれぞれ専有環境を持つため、ユーザの管理は事実上不要である。課金は管理はパブリッククラウドのアカウント単位で行う。個々のユーザに個別にアカウントを用意することで、個々のユーザの資源使用量を精密に知ることができる。セキュリティに関する考え方はやや異なる。基本的に環境を動的に生成・破棄することで、機密情報がクラウド上に存在する時間を短くする。また、ユーザごとに環境を分離することで、一人のユーザが攻撃された場合の影響範囲を限定する。ログインしての管理が不要な環境とすることで、外部からのネットワークアクセスを完全に遮断した環境とすることが可能だ。

われわれは、これらの手法の得失を検討するため、双方の実装を行った。

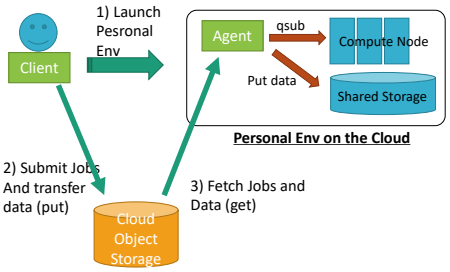

図 4 CloudQ/S の概要

4. CloudQ/S の実装

CloudQ/S は、シングルユーザ法で行った実装で、産総研が担当した。

4.1 概要

CloudQ/S の概要を図 4 に示す。CloudQ/S は個々のユーザが必要に応じて個別にバッチキューイングシステムをパブリッククラウド上に構築し、必要がなくなればシャットダウンする。システムの構築とシャットダウンは CloudQ/S が用意するスクリプトで行うため、ユーザの負荷は大きくない。

CloudQ/S は、下記の 3 つのコマンドで構成される。

- `cloudqcli`: ジョブサブミッションを行うクライアントコマンド
- `cloudqagent`: ジョブの実行を司るエージェントコマンド
- `cloudqaws`: パブリッククラウド上にバッチキューイングシステムを構築するビルダコマンド。

このうち、`cloudqcli` と `cloudqagent` については [4] で紹介した ABCI 向け CloudQ のものと基本的には同じである。

CloudQ/S のユーザは、まずジョブサブミッション用のバケットを、どこかのクラウドストレージ上に作成する。このクラウドストレージは、ターゲットとなるパブリッククラウドのものである必要はない。たとえば、ABCI の S3 ストレージを使用して AWS にジョブをサブミットすることも可能である。

次に、ビルダコマンドを用いて、パブリッククラウド上にバッチキューイングシステムを構築する。この際に、ジョブサブミッション用のバケットを指定する。ビルダコマンドで構築された環境内では、エージェントが動作している。このエージェントは定期的にジョブサブミッション用のバケットを監視する。

ユーザは、クライアントコマンドを用いてジョブを投入する。この際にも同様に、ジョブサブミッション用のバケットを指定する。エージェントはバケットにジョブが投入されたことを検知すると、バケットからジョブを取得し、バッチキューイングシステムに投入し、以降ジョブの状態を監視する。

ユーザは、クライアントコマンドを用いてジョブの実行状態を監視し、ジョブの実行が終了したら、同様にクライアントコマンドを用いて結果を取り出す。

4.2 CloudQ/S の実装

CloudQ/S は対象とするパブリッククラウドとして Amazon Web Services(AWS) を用いた。バッチキューイングシステムの構築には、2.2.2 で示した AWS ParallelCluster を用いた。ただし、AWS ParallelCluster では、われわれの望むセキュリティ制約を実現できなかったため、VPC のみ AWS CloudFormation を用いて別途作成している。

ParallelCluster ではオートスケール機能が実装されている。起動直後には、ヘッドノードとなる非常に小規模なノードだけが稼働している。ユーザがジョブを投入すると、そのジョブの実行に必要となる計算ノードを自動的に起動してくれる。また、不要になった計算ノードのシャットダウンも自動的に行なってくれる。

ParallelCluster ではヘッドノード起動時に実行するスクリプトを指定する事ができる。この機能を利用して、エージェントコードを導入する。シングルユーザ環境であるため、ユーザに関する設定を行う必要はない。ジョブの実行はデフォルトユーザの権限で行われれる。

ログ出力には、AWS のログ管理サービスである、Amazon CloudWatch Logs[10] を利用した。各ノードのシステムログ、バッチキューイングシステムデーモンのログはすべて、CloudWatch Logs に集約される。このため、環境をシャットダウンして完全に破棄したあとでも、随時ログを確認することができる。

4.3 シングルユーザ法の利点

シングルユーザ法での実装の最大の利点は、実装の容易さである。ユーザを設定する必要がないため、ParallelCluster が構築する環境をほとんど変更せずにそのまま使用する事ができる。また、ユーザの課金管理もいわばパブリッククラウド側に委譲するため、管理のためのコードを作り込む必要がない。

CloudQ/S で使用するクラウド上の環境には、基本的にログインする必要がない。エージェントとクライアントの通信は、外部のクラウドストレージを経由するため、バッチキューイングシステムに直接アクセスする必要がない。ログ情報は、CloudWatch Logs に集約されているので、ログインせずに確認することができる。したがって、外部に対してまったくオープンなポートを持たない状態で計算機クラスタを運用することができる。このため外部からの攻撃を考慮する必要がない。

また、CloudQ/S で構築される環境は状態を持たず、基

本的に使い捨てである。このため、パブリッククラウド環境を使用しない場合の課金を完全にゼロにすることができる。さらに、セキュリティアップデートを行う必要もない。ParallelCluster が使用する仮想計算機イメージは、AWS に管理されており必要なセキュリティアップデートが自動的に行われた状態となっている。このように、計算機管理に関するコストを抑えることができる。

5. CloudQ/M:オンプレミス環境と親和性の高いマルチユーザ環境

CloudQ/M は、マルチユーザ法で行った実装で、日立製作所が担当した。

日立製作所 研究開発グループでは、オンプレミスの HPC クラスタおよび AI クラスタを構築し運用している。今回、産総研との共同研究の成果である CloudQ を使ったクラウドバースティング機能の実現にあたり日立はマルチユーザがクラウド環境を共有する実装を行った。

本節では、日立における課題、実装の方針、実装の概要について示す。実装の概要を図 5 に示す。

5.1 オンプレミスのシステム概要

日立社内の HPC クラスタ、AI クラスタ共に典型的な HPC システムであり、高速共有ファイルシステム、ログインノード、計算ノード群、高速ノード間ネットワークで構成されている。HPC クラスタと AI クラスタの違いは主に GPU の有無だけであり、両方とも PBS ジョブスケジューラで運用し、社員 ID をユーザ ID とすることで課金との紐づけを容易にしている。

5.2 クラウドバースティング機能の課題

利用者は数百人規模であり、管理者にとっては利用統計や課金、セキュリティ設定の適切で効率的な管理が必要である。特に管理コストを考えるとオンプレミスと同じ実行ログが発生する PBS 利用が望ましい。またアプリケーション利用が中心の計算機に不慣れなユーザにとっては、従来の HPC/AI クラスタのアーキテクチャ概念の他に、AWS 等のクラウドアーキテクチャを理解するのは負担になる。これらの観点からオンプレミス環境との互換性を多く持つのが望ましい。逆にクラウドならではの利点も享受したい。クラウドはオンプレミス環境があふれた際のクラウドバースティング用途のため、常に多くのノードを確保しておき課金が発生するのではなく、必要で使った分だけ支払いたい。また、クラウドではメモリや CPU コアの大小、GPU の有無や種類を選べるので、ジョブにとって最適な計算ノードを確保したい。

5.3 実装の方針

一人のユーザに対してマイクラスタを生成する実装も検討したが、数百人それぞれクラスタを作った場合のオーバーヘッドの懸念、オンプレミス環境と同様に高速共有ファイルシステムに/home を置きたい、ユーザをまとめて管理したいという理由からオンプレミス環境に近い、マルチユーザの PBS ジョブスケジューラ管理とした。クラウドの利点の享受に関しては、計算ノードを常に確保しておくのではなく、ジョブが投入されたらジョブの要求リソースに従って計算ノードを生成し、ジョブが終了したら削除することでアイドル状態でのコストを削減する。また、クラウド上で Lustre ファイルシステムなどを大容量で確保し続けるのはコストがかさむため、Lustre ファイルシステムの容量は最低限とし、HSM(Hierarchical Storage Management) 機能 [9] を使って不使用時は S3 オブジェクトストレージにアーカイブすることにした。また、この S3 は https プロトコルで社内からアクセスできるため、CloudQ インタフェースで S3 経由でジョブを実行できるようにする。

5.4 実装の概要

実装は AWS 上に行った。常時起動するノードは LDAP でユーザを管理する Master node、これ以外にクラスタ毎に必要となる Login node である。Login node では PBS ジョブスケジューラが動作している。実行キューは現状では一つだけであり、また通常は計算ノードも登録されていない。これら Master node, Login node の生成はスクリプト化してあり 10 分ほどで生成できる。現在は AWS 東京リージョンで動作しているが、他のリージョンでの立ち上げも容易である。

さらに今回 S3 や PBS へのジョブ投入を監視して計算ノードを生成、起動、削除するエージェントツールを開発した。実装にあたっては AWS のリソース管理サービスである CloudFormation を利用した。理由は、今回の計算ノードの動的確保方式の場合、一時的な AWS リソース不足のためノード確保に失敗する可能性があり、CloudFormation の場合リトライ等まで行ってくれるからである。また、リソースをグループ管理できるため不要になった場合グループごと確実に削除できる。計算ノードはジョブスクリプトで指定したリソースを満たす EC2 を選択して事前に用意した AMI(Amazon Machine Image) から起動する。計算ノードはきっかけとなったジョブ専用でなく他のジョブに再利用されるが、5 分間アイドル状態だった場合シャットダウン、削除することでコストを削減する。

ファイルシステムに関しては共有利便性と性能を考えて/home を Lustre ファイルシステムとした。こちらも AWS が HSM 機能付きの FSx for Lustre のマネージドサービスを提供していたため利用した。FSx for Lustre では構築時に S3 と対応付けしておくことで S3 に透過的にアーカイブすることが可能である。今回の実装ではコスト削減のため Lustre ファイルシステムの契約容量を少なくしてい

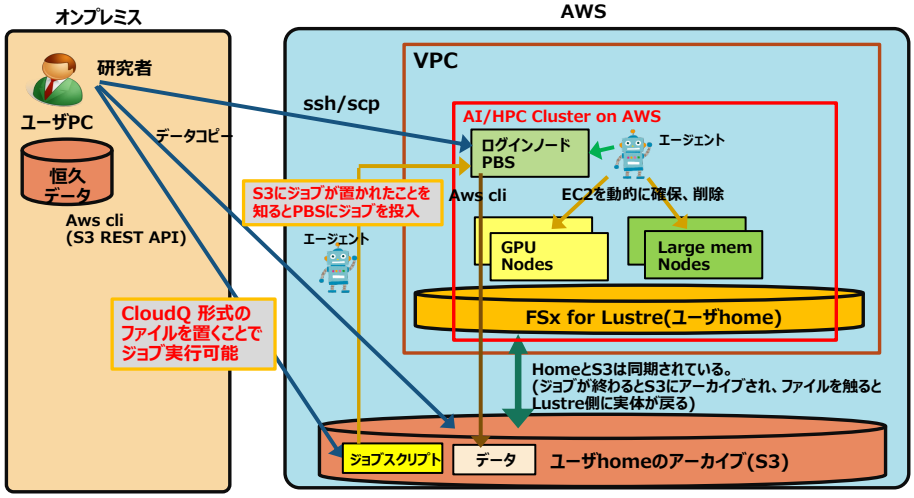

図 5 CloudQ/M の概要

るため、ジョブの終了時、ユーザの logout 時などに積極的に S3 にアーカイブしている。ファイルのメタデータは Lustre に残るためユーザやジョブがファイル操作を行なうと直ちにリコールされる。

この/home 透過な S3 オブジェクトストレージを使って、CloudQ インタフェースでジョブを投入できるようにした。S3 経由で自分の home ディレクトリにジョブスクリプトを置くと、エージェントが代わりに PBS に Submit する。ジョブの実行状態は動的に更新される manifest ファイルで確認できるほか、ジョブが終了すると結果ファイルの他に stdout, stderr も生成されるため、S3 経由でジョブ操作が完結できる。この https の REST API のみでジョブを投入できるインタフェースは、ssh/scp 等を使ってクラウドシステムにログインして操作するのに対し、制限はあるがセキュリティリスクを減らすことが可能になる。

ユーザにおいては極力 AWS の世界を意識しないで良いように設計した。管理者がユーザ ID 作成をすると、AWS 上の IAM と S3 の権限、セキュリティの設定、Linux の Posix id とのマッチングをして一括設定が行なわれる。一般ユーザは IAM ログインは行えない。

また、クラウドでは多くの計算資源を使えるメリットはあるが、ジョブスクリプトの失敗、コスト単価の見誤りなどで予定外の課金がかかることが懸念される。そこで、本実装では、クラスタ全体および各ユーザでの同時利用計算ノード上限を設定できるようにした。

現在、本システムを日立社内で実運用中である。

6. 関連研究

文献 [11] は、Slurm に用意された REST API を用いたジョブの投入を主題にしたブログである。この中で、S3 上のバケットへのファイル書き込みをトリガとして AWS Lambda を起動し、起動された関数で Slurm へ REST API を用いてジョブを投入する手法が紹介されている。直接 REST API を公開するのではなく、S3 への書き込みを経由させることによって、S3 トークンの持つ柔軟なユーザ権限制御を利用している点が CloudQ のアプローチと類似している。

7. おわりに

本稿では、AI/HPC システムのパブリッククラウドへのクラウドバースティングに向けて問題点を整理し、シングルユーザ法とマルチユーザ法がありうることを示した。シングルユーザ法の実装として CloudQ/S を、マルチユーザ法の実装として CloudQ/M について紹介した。これら 2 つのアプローチは特にセキュリティを担保する手法が全く異なり、それぞれ得失がある。管理者が重視する項目によっていずれかを選択すればよい。

本稿ではジョブスループットやレイテンシなどの性能測定を行わなかった。これは、これらの性能指標が主にエージェントの監視インターバルによって定まるため、あまり提示する意味がないと考えたためである。

CloudQ/S についてはコードを整理した後に一般公開する予定である。

参考文献

[1] Nair, S. K., Porwal, S., Dimitrakos, T., Ferrer, A. J., Tordsson, J., Sharif, T., Sheridan, C., Rajarajan, M. and Khan, A. U.: Towards Secure Cloud Bursting, Brokerage and Aggregation, *2010 Eighth IEEE European Conference on Web Services*, pp. 189–196 (2010).

[2] Guo, T., Sharma, U., Wood, T., Sahu, S. and Shenoy, P.: Seagull: Intelligent Cloud Bursting for Enterprise Applications, *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pp. 361–366 (2012).

[3] Date, S., Kataoka, H., Gojuki, S., Katsuura, Y., Teramae, Y. and Kigoshi, S.: “First Experience and Practice of Cloud Bursting Extension to OCTOPUS”, *10th Inter-*

national Conference on Cloud Computing and Services Science, CLOSER2020, pp. 448–455 (2020).
[4] 滝澤真一朗，高野了成，清水正明，松葉浩也，中田秀基，小川宏高：クラウドオブジェクトストレージを活用したメタスケジューラ，研究報告ハイパフォーマンスコンピューティング (HPC)，Vol. 2020-HPC-177, No. 5, pp. 1–8 (2020).
[5] : ABCI, https://abci.ai/.
[6] : AWS CloudFormation, https://aws.amazon.com/cloudformation/.
[7] : AWS ParallelCluster, https://aws.amazon.com/hpc/parallelcluster/.
[8] : Amazon FSx, https://aws.amazon.com/fsx/.
[9] Degrémont, A. and Leibovici, T.: Lustre HSM Project, https://wiki.lustre.org/images/4/4d/Lustre_hsm_seminar_lug10.pdf (2009).
[10] : What is Amazon CloudWatch Logs?, https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html.
[11] Bjorgaard, J.: Using the Slurm REST API to integrate with distributed architectures on AWS, https://aws.amazon.com/blogs/hpc/using-the-slurm-rest-api-to-integrate-with-distributed-architectures-on-aws/ (2021).