Vol.2021-HPC-180 No.20
2021/7/21

microburst: クラウドネイティブ環境を起点とした異種混合HPCアプリケーション開発と展開の検討

杉木 章義[1,a)]

概要：本研究では，パブリッククラウドを起点として HPC アプリケーションの初期開発を行い，開発が進行した段階で，複数の研究機関が提供する大型計算機システムへアプリケーションを水平展開する開発手法，microburst を提案する．本研究のアプローチは次の通りである．(1) 提供機関全体で単一のクラスタを作成せず，小規模な研究ユニットごとにクラスタを構成する．(2) 単一コードから複数の異種混合アーキテクチャ用のコードを生成することを想定する．(3) スポットインスタンスの活用，インスタンスの一時停止や再開など，費用低減のための工夫を最大限行う．(4) コード開発環境，ジョブスケジューラ，ワークフロー等に関して，従来 HPC で広く活用されていたものを利用せず，各クラウドベンダが提供するクラウドネイティブなサービスがあれば，それを活用する．(5) パブリッククラウドから大型計算機システムへ展開するのは小規模なコード，バイナリ，コンテナのみとし，大容量データの転送は研究機関からクラウドへの一方向のみとし，費用を低減する．具体的には，Amazon Web Services (AWS) に特化したプロトタイプ環境を構築し，その可能性と今後の課題を明らかにする．

microburst: Towards Bursting Multi-architecture HPC Applications down from a Cloud-native Environment

1. はじめに

HPC (High Performance Computing) 分野では，パブリッククラウドの活用が検討されている．多数のインスタンスによる豊富な計算資源が即時に手に入るという点で，クラウドは魅力的な一方で，当初の仮想化技術によるオーバヘッド，Ethernet ベースのインターコネクトの弱さから，採用があまり進んでこなかった．

近年，この状況は急速に改善しつつあり，パブリッククラウドであっても，各学術機関に導入されている大型計算機システムに近い性能が得られるようになっている．例えば，Amazon Web Services (AWS) では，2020 年 12 月に最大 4.5GHz で動作する Cascade Lake 世代のカスタム Xeon Scalable プロセッサを搭載した M5zn インスタンスが登場し，最大 100Gbps の ENA (Elastic Network Adapter)，さらには MPI や libfabric に対応し，これまでの通信遅延を大幅に削減した EFA (Elastic Fabric Adapter) [1] による高速なインターコネクトが利用可能となっている．また，Microsoft Azure や Oracle Cloud Infrastructure (OCI) などの他のクラウドでも，同等の高性能インスタンスや，InfiniBand が早期から利用可能となっている．

また，近年のパブリッククラウドでは，多種多様なアーキテクチャやプロセッサ，機械学習や大規模データ解析などのより上位層のサービスが利用できるという，大型計算機システムにはない魅力もあり，研究者の関心が急速に高まっている．AWS においても，AMD EPYC プロセッサを採用したインスタンスや，64bit Arm ベースの Graviton2 プロセッサ（Neoverse N1 世代）[2] を採用したインスタンスが利用可能となっている．GPU ももちろん利用可能であり，執筆時点で 8 基の NVIDIA A100 や，AMD Radeon Pro V520 が利用可能である．さらには，Xilinx の FPGA や，AWS Inferentia のカスタム機械学習推論プロセッサ，AWS Braket による量子コンピューティング環境も提供されている．

HPC 分野におけるパブリッククラウドの利用は魅力的な一方で，いくつか解決すべき課題がある．初めの費用の最適化では，一般的に HPC では豊富な計算資源量と長時

[1] 北海道大学情報基盤センター
Information Initiative Center, Hokkaido University, Sapporo 060–0811, Japan
[a)] sugiki@iic.hokudai.ac.jp

間計算を必要とすることから，同一ワークロードをそのままの計算機構成でパブリッククラウド上で実行した場合には，大幅に高額な費用となる．そのため，計算機（インスタンスタイプ）やストレージ容量のサイジングや，問題設定の工夫などによる現実的な範囲への費用の落とし込みが必要である．上記の作業には，アプリケーションに対する深い造詣と，パブリッククラウドに関する豊富な経験の双方を必要とする．

二つ目の文化的ギャップの解消では，従来からの HPC 技術と Web を基礎としたクラウド技術では，同じような並列分散の問題に対して異なるソフトウェアスタックが用いられている [3]．Amazon EC2 などの計算機インスタンスを活用し，従来のジョブスケジューラに基づくクラスタ環境をクラウド上に構築することも可能であるが，それだけでは，AWS Lambda などのサーバレスコンピューティングや，Istio や AppMesh などのマイクロサービス，さらには SageMaker などの機械学習サービス，DynamoDB や Aurora などの大規模データベースや，Redshift や Kinesis，EMR などの大規模データ解析といった高水準なサービスの成果を活用することができない．

本研究では，HPC アプリケーション開発の初期段階においてパブリッククラウドを利用し，その後，大規模な計算資源を必要とする場合には，各大学や各研究機関が提供する複数の大型計算機システムにアプリケーションを水平展開する “microburst” と呼ぶアプローチを提案する．研究の初期段階においては，豊富な高水準サービスや，最新世代のハードウェアなど，パブリッククラウドの利点を最大限活用し，研究期間の後半では，スーパーコンピュータをはじめとする学術機関が提供する大型計算機システムの豊富な計算資源による費用対効果を活用する．前者では，パブリッククラウドを起点としたクラウドネイティブの視点から，従来の HPC システム環境全体を捉え直し，ソフトウェアやサービスの再構築を行う．後者では，大量の計算資源を消費し，大きな費用を必要とする部分をパブリッククラウドから極力排除することで，経済性の改善を目指す．

2. 関連研究

HPC 分野の研究開発の初期段階において，パブリッククラウドを活用し，その後，各大学や研究機関の大型計算機システムに展開するというアイデアは，著者らが初めてではない．最近，HPCwire Japan [4] のパネルディスカッションなどで活発な議論が行われている．本研究の特徴は，クラウドネイティブの視点に立ち，その際の設計指針を示している点であり，具体的なシステム構成も示している．

パブリッククラウドは HPC 分野において，さまざまに活用されており，その全てを示すことは難しい [5], [6]．また，Nextflow [7] などフレームワークの対応から，バイオインフォマティクスなどの特定の分野での活用が先行しており，その浸透は一律ではない．

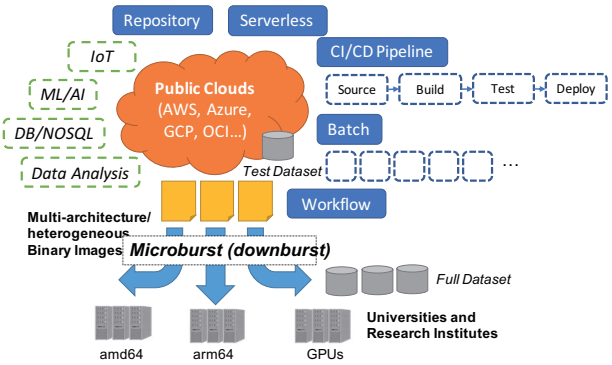

図 1　研究開発環境 “microburst” の概要
Fig. 1　Overview of “microburst” development architecture

パブリッククラウドの活用に関しては，学術界と比較して，産業界が先行していると推測される．調達上の課題もあるが，各学術機関において大型計算機システムが提供されており，豊富な計算資源が比較的低廉な負担金で利用可能である点も大きい．また，Rescale，Azure CycleCloud，Penguin Computing，Adaptive Computing，NIMBIX，Gompute，Sabalcore，UberCloud，Xtreme-D などの HPC クラウドを提供する企業やサービスも多数存在している．これらの企業では，既存の AWS や Microsoft Azure，Google Cloud Platform (GCP) などを HPC 用にカスタマイズし，研究者に対して使いやすくする，または HPC クラウドを直接自身で提供するなどの工夫を行なっている．これらに対して，本研究では，AWS が提供するクラウドを起点に開発された現時点でのネイティブなサービスを直接使用し，プロトタイプ環境を構築することで，その可能性と課題を明らかにする．

3. 提案

本研究では，パブリッククラウドを起点として HPC アプリケーションの初期開発を行い，開発が進行した段階で，複数の研究機関が提供する大型計算機システムへアプリケーションを水平展開する開発手法，“microburst” を提案する（図 1）．

本研究のアプローチは次の通りである．

- **小規模な研究ユニットでの利用：** 本研究では，研究者個人や数名からなる研究グループなどの小規模な研究ユニットを対象とする．パブリッククラウドは小規模なものから，いかなるスケールの計算機クラスタも構成できるのが一つの特徴であり，また，仕様調達上の困難さによる問題も極力回避する．本利用は従来の大型計算機システムの占有ノード利用に対応し，計算資源が不足する場合には，リソースを追加できることから，ジョブ待ち時間も大幅に短縮する．
- **複数・異種混合アーキテクチャを対象とした開発：**

近年のパブリッククラウドの一つの特徴は，Intel や AMD，Arm や NVIDIA など，さまざまなアーキテクチャのハードウェアが利用できる点である．また，大型計算機システムへの導入前に先行して，新しい世代のハードウェアで検証することも可能である．さらには，FPGA や AWS Inferentia などの機械学習推論プロセッサ，Amazon Braket などの量子コンピューティングも利用可能である．本研究では，複数・異種混合アーキテクチャでの開発や展開を当初より想定する．

- **細粒度の停止による費用の最適化：** 近年のパブリッククラウドでは，1 時間単位から秒単位や分単位の課金へと移行している．そのため，短期間でのクラスタ構築や削除，計算機のサスペンドやレジュームによる一時停止の効果が大きい．1 ヶ月未満の短期間の利用であれば，1 ヶ月単位や年単位で負担金が請求される従来の大型計算機システムと比較して，計算のための費用が抑えられる可能性もある．
  また，スポットインスタンスなどの中断される可能性があるが低廉なインスタンスを活用することで，費用を数分の一に抑えることが可能である．
  さらには，AWS Lambda や Step Functions などのサーバレスでイベント駆動で起動や課金されるサービスを利用すれば，利用停止中の課金を当初より回避することが可能である．
- **従来の代替としてのクラウドネイティブなサービスの活用：** 本研究では，従来の HPC 環境で用いられていたジョブスケジューラやワークフローエンジンをそのままクラウド環境上で利用せず，パブリッククラウドでクラウドネイティブな視点で開発された代替可能なサービスがあれば，それを利用する．これにより，同じ並列分散の問題を対象としながら，技術的な乖離が進む HPC とクラウドの融合を再び試み，既存の HPC 環境への現時点のクラウド技術のバックポートを目標とする．
- **大容量データ転送の削減：** パブリッククラウドの HPC 利用において，計算機利用に加えて，大きく費用がかかるポイントとなるのが，データ転送料金やその保管のためのストレージ容量料金である．従来の HPC のクラウド利用では，クラウドバースティングの文脈で利用が検討されることが多く，大きな課題となった．本研究では，クラウド側で利用するのは小規模な試験データセットのみとし，本格的な実データを用いた計算は，後日，各研究機関の大型計算機システムで実施することとする．また，データを転送する場合も，大きくは各研究機関からクラウドへの一方向のみとし，データ転送による課金の問題を回避する．

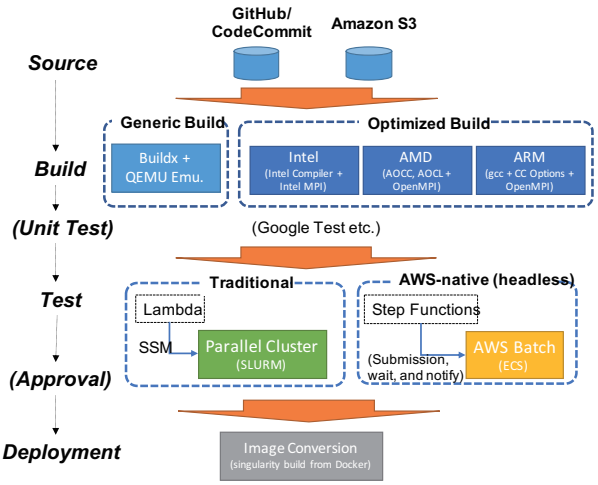

図 2　研究開発パイプラインの概要
Fig. 2　CI/CD pipeline for HPC applications

4. 実装

本研究では，特定の HPC アプリケーションを対象とした完成された研究開発パイプラインではなく，アプリケーションに応じて汎用的に組み合わせ可能な部品を構築し，各機能の検証を行う．

また本研究では，AWS を対象としてプロトタイプ環境の構築を行う．AWS を採用した理由は，HPC 分野に限らない一般的なパブリッククラウド利用で，現時点において最も高いシェアを確保していること [8]，AWS は最大規模の多種多様なサービスを提供しており，パブリッククラウドの可能性を探索するには都合がよいことなどがある．また，2019 年から最大 100Gbps の EFA や AMD EPYC インスタンス，2020 年から 64bit Arm の Graviton2 インスタンスが導入されたことも大きい．

AWS の HPC 利用に関しては，AWS 自身により Well-Architected フレームワークの HPC Lens [9] として，導入のガイドラインが公開されている．Well-Architected フレームワークは意図的に導入者に問いかけを行い，改善を促す高水準なドキュメントとして記述されており，本研究では具体的なシステムを構築する．

4.1 研究開発パイプラインの概要

本研究の研究開発パイプラインの概要を図 **2** に示す．コード開発において，近年，一般的に用いられる CI/CD（Continuous Integration，Continuous Delivery または Deployment）を HPC においても積極的に導入する．

以降，パイプラインの詳細をソース開発，ビルド，テスト，展開フェーズに分割して説明する．

4.2 ソース開発フェーズ

研究開発の起点となるソース開発フェーズでは，従来

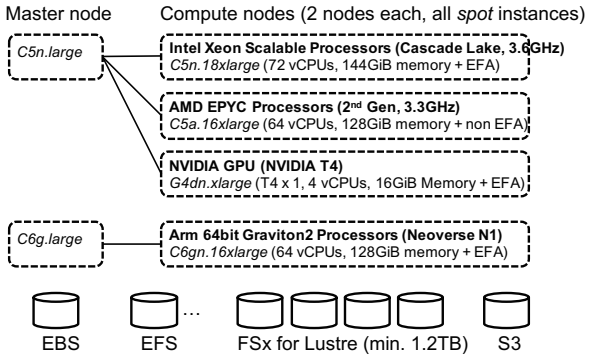

図 3 Parallel Cluster 環境
Fig. 3 Parallel Cluster

の HPC 環境との親和性から，AWS が提供する Parallel Cluster [10] を使用する．Parallel Cluster では，SLURM や AWS Batch に基づく，EC2 インスタンスを活用した計算機クラスタを 15 分程度の短時間で構築可能である．また，ソースコードの記述やコンソール利用には，AWS が独自に提供する開発環境 Cloud9 を利用することも可能である．

本研究の Parallel Cluster 環境では，**図 3** のプロトタイプ環境を構築した．標準の Parallel Cluster では，ログインノードと計算ノードが同一アーキテクチャでなければならないという制約があり，x86 系と Arm 系で 2 つのクラスタを提供する．x86 系クラスタでは，Intel，AMD，NVIDIA GPU のアーキテクチャごとに，3 つのキューをそれぞれ設けたマルチキュー方式としている．各インスタンスは，それぞれ `c5n.18xlarge`，`c5a.16xlarge`，`g4dn.xlarge` を選択し，各々，最大 2 台起動するように設定した．上記は EFA に対応する最小のインスタンスから選択している．AMD EPYC は現時点においては，EFA に対応せず，最大 20Gbps の帯域となっているが，近い将来，改善することが期待される．本論文では，Intel の vCPU 数とメモリ容量に近いインスタンスを選択した．Arm 系のクラスタでは，`c6gn.16xlarge` の 64bit Arm Graviton2 インスタンスを最大 2 台使用する．こちらは 2020 年 12 月に導入された C6gn インスタンスから EFA に対応している．

各計算ノードは，ジョブが投入され，実行が終了してから一定時間が経過するまで起動する．本研究では全てスポットインスタンスを使用しているが，Parallel Cluster では，他の AWS サービスが対応しているスポットフリート機能 [11] は使えない．スポットフリート機能は，vCPU 数やメモリ容量などの性能が近い他のインスタンスもスポット実行の候補に含めることで，実行の可能性を高め，中断の可能性を下げる手法である．一方で，スポットフリートを用いない場合には，同じインスタンスタイプを使用することとなり，実行結果の再現性が高まる．Parallel Cluster はそもそもオープンソースとして公開されており，利用者が独自の改良や拡張を加えることが可能である．

上記のクラスタでは，Parallel Cluster が標準でサポートする Amazon EFS (Elastic File System) や FSx for Lustre も使用可能としている．EFS は NFSv4 ベースの共用ファイルシステムであり，事実上無限に拡張可能なストレージ容量のみで課金されるという利点がある．FSx for Lustre はオープンソース版の Lustre をベースとしており，高い並列 I/O 性能が期待され，一時的なスクラッチ利用が想定されている．

Parallel Cluster では，計算ノードは自動的に停止するが，ログインノードはクラスタを停止しても起動したままとなる．EC2 でログインノードのインスタンスも手動で停止し，NAT に割り当てられた Elastic IP も解放すれば，さらに課金を抑えることができる．

4.3 ビルドフェーズ

以降の CI/CD パイプラインでは，AWS 独自の CodePipeline を使用する．特に CodePipeline に依存した機能は使用しておらず，汎用的なパイプラインで代替可能である．

パイプラインへの入力は，(1) ソースコードの場合は GitHub や CodeCommit などのリポジトリ，(2) `tar.gz` 形式や `zip` 形式などの tar ボールの場合は，Amazon S3 で行う．リポジトリ内容の更新や S3 へのアップロードを契機として，パイプラインの実行を開始する．各ステージでは，AWS の CodeBuild を使用し，Docker コンテナの push には ECR (Elastic Container Registry) を使用する．

- **汎用ビルドフェーズ**：汎用ビルドフェーズでは，Dcoker Buildx と QEMU を利用して，クロスビルドにより，マルチ CPU アーキテクチャ対応イメージを構成する．Buildx は同一の Dockerfile から複数 CPU に対応したイメージを作成し，同一のイメージタグで異なる CPU イメージを提供することが可能である．しかしながら，Docker コンテナでは基本的にさまざまな環境で汎用的に実行できることを想定しており，HPC 分野で期待する特定プロセッサへの最適化などが十分ではない．
- **最適化ビルドフェーズ**：最適化ビルドフェーズでは，汎用ビルドフェーズの代替として，各ベンダや世代のプロセッサに最適化した Docker コンテナを構築する．Intel プロセッサでは，Intel oneAPI HPC Toolkit を使用して，Intel コンパイラと Intel MPI の組み合わせで Docker コンテナを構築する．AMD プロセッサでは，AOCC と AOCL を使用してビルドを行い，AMD 拡張の Clang/LLVM と OpenMPI の組み合わせとなる．Arm の Graviton2 では，GCC と OpenMPI の組み合わせを用いるが，AWS が提供する Graviton2 推奨 GCC コンパイルオプション [12] を採用する．しかしながら，Amazon Linux 2 では，GCC7 と GCC10 の

双方が利用可能だが，Fortran は GCC7 版しか標準で提供されていなかった．今回は時間制約の関係から，GCC7 で使用可能な最適化オプションでコンパイルする．また，有償の Arm Allinea Studio も導入していない．

現状の Docker のマルチアーキテクチャイメージの仕様では，amd64 や arm64 などといった大まかな分類であり，各プロセッサの型番や世代などを区別しないことから，イメージのタグ付けによって工夫を行う．

CodeBuild では，ビルド後に単体テストを行い，テスト結果やテスト網羅率のレポートを GUI 上に出力することも可能である．CodeBuild は JUnit などの XML 形式レポートであれば，GUI の画面上に統合可能である．C/C++のアプリケーションであれば，GoogleTest などの導入が考えられる．

4.4 テストフェーズ

テストフェーズでは，Docker コンテナ化された HPC アプリケーションをクラスタ環境で並列実行し，テストを行う．本研究では下記の二つを実装する．

- **Parallel Cluster (SLURM)**：SLURM 環境でテストする場合には，前述の Parallel Cluster 環境をそのまま使用する．ジョブの投入には，AWS Lambda の Lambda function から SSM（Systems Manager）を経由して Send Command 命令でコマンドを送信する．ジョブの経過を確認する際には，別サービスであるワークフロー実行エンジンの Step Functions をさらに組み合わせる．Parallel Cluster で SSM を利用可能とするためには，SSM エージェントをログインノードにインストールし，IAM のロールおよびポリシーを調整する．本研究では，Lambda から SSM コマンドが送信できることまで確認している．
- **AWS Batch**：前述の SSM や SSH でログインノード上でコマンド実行する手法は，やや間接的な連携となる．AWS Batch はクラウド上でフルスクラッチで開発されたバッチサービスであり，他の AWS サービスとの緊密な連携が可能となる．本研究では，AWS が提供する Step Functions のサンプル記述を参考に，Step Functions から AWS Batch でマルチノードによる並列実行を行い，実行結果を確認するステージを作成した（図 4）．

  しかしながら，AWS Batch の利用には多少の制約がある．まず，マルチノードジョブはスポットインスタンスとの組み合わせでは実行できず，オンデマンド実行のみとなる．これは単一ノードジョブやアレイジョブでは問題とはならず，AWS Batch の背後では ECS (Elastic Container Service) で実装されていることから，オンデマンド実行でありながら，多少，起動や停止が効率的である．本研究では，オンデマンド実行用のキューとスポット実行用のキューを別に用意している．AWS Batch はヘッドレスサービスであり，ジョブが実行されない場合には課金されないサービスであることから，複数のキューを用意しても問題はない．また，EFA を使用した MPI 実行には多少の記述や設定の煩雑さを伴う．特に各コンテナ起動の冒頭で MPI の Machine File をコンテナ内部で生成する処理が必要であり，本処理はテンプレート化可能でありながら，煩雑さを伴う．

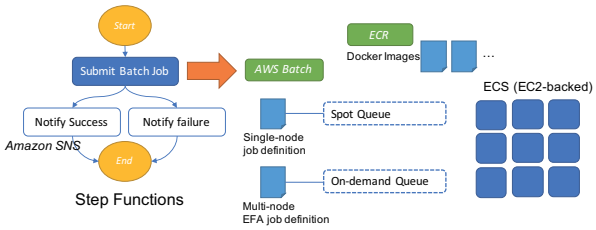

図 4　AWS Batch へのジョブ投入ワークフロー
Fig. 4　Workflow for job submission onto AWS Batch

CodePipelline では，テストの実施後，リリース前に電子メールなどによる人手の承認フェーズを挿入することも可能である．

4.5 展開フェーズ

テストが成功すれば，各大学や各研究機関の大型計算機システムへ HPC アプリケーションを水平展開する．その際の選択肢として，(1) ソースコードをリリースし，実行環境で再コンパイルを行う，(2) バイナリリリースを行う，(3) コンテナとしてリリースするの三つの選択肢がある．

最後のコンテナ展開にあたり，多くの大型計算機システムでは，Singularity を採用していることから，Docker から Singularity へのイメージ変換ステージを作成した．x86 用と Arm 用の二つの変換ステージを用意し，並列ビルドを実施する．

また，多くの HPC アプリケーションでは，一般的に常時サービスとして稼働していないことから，CI/CD の Continous Deployment（継続デプロイ）というよりは Continous Delivery（継続配信）の形態となる．特に，大阪大学の SQUID [13] などの最近の大型計算機システムではセキュリティを向上させる目的から，ログインの際に人手による操作を必要とする多要素認証（MFA）を導入しており，パブリッククラウドからの push 型ではなく，大型計算機システムからの pull 型の配信やデプロイを必要とする．

5. 予備実験

今回，パブリッククラウドの性能面の問題を評価の中心とせず，主に機能的な検証や費用面での検討を中心に，予備的な評価を実施する．コンテナ内で稼働する HPC アプ

表 1　Parallel Cluster の構築・削除時間
Table 1　Creation and Delete of Parallel Cluster

構成	構築時間	削除時間
MQ (Intel, AMD, Nvidia)	16 分 27 秒	8 分 33 秒
SQ (Arm)	15 分 39 秒	9 分 44 秒

リケーションとしては，[14] を参考に OSU ベンチマークと NAS Parallel Benchmark の二つを使用している．OSU ベンチマークは EFA のレイテンシ評価を目的として採用し，NPB はより一般的なアプリケーション評価を目的とする．各所要時間の計測は 3 回の平均としている．

5.1　Parallel Cluster 環境の構築

AWS 上での Parallel Cluster 環境の構築や削除にはやや時間を要し，その際の時間や費用面でのオーバヘッドの考慮が必要である．そのことを検証した結果を**表 1** に示す．本表では 4.2 節の Paralell Cluster 環境を構築しており，x86 (amd64) 系のマルチキュー（`MQ`），Arm (arm64) 系の単一キュー（`SQ`）双方のクラスタで実験を行なっている．構築と削除は主にマスタノードのみで行われることから，マルチキューと単一キューで所要時間はそれほど大きく変わらない．

マスタノードで使用してる各インスタンスは 1 時間あたり，本論文の執筆時点で `c5n.lage` では 0.108 USD，`c6g.large` では 0.068 USD のオンデマンド料金が発生する．一方の計算ノードは，3 ヶ月間のスポット料金の設定履歴から `c5n.18xlarge` で 3.888 USD のオンデマンド料金に対して 1.1590 USD，`c5a.16xlarge` で 2.464 USD に対して 1.0302 USD，`g4dn.xlarge` で 0.526 USD に対して 0.1578 USD のオンデマンド料金が 1 時間，1 ノードあたり発生する．AMD Graviton2 の `c6gn.16xlarge` では，2.7648 USD に対して 1.0818 USD である．

計算ノードは自動的に停止するため，ジョブの実行時間に大きく依存する．一方のマスタノードは稼働したままとなるため，クラスタの削除やマスタノードの一時停止の検討が必要である．頻繁な構築や削除は必ずしも費用の削減に繋がらないため，工夫が必要である．

5.2　Pipeline 実行時間

Parallel Cluster 環境での開発の後，各学術機関の大型計算機システムへ Docker または Singularity のコンテナを展開する．そのための CI/CD パイプラインの実行時間を**表 2** に示す．表 2 では，所要時間に揺らぎが生じるテストフェーズを実験の都合から省略し，ソース・ビルド・ステージングの 3 層としている．本実験では，汎用ビルドを使用し，Singulariry イメージに変換しているが，全体で 32 分程度の時間を要している．

かかる費用に関しては，CodeBuild では Linux のビルド

表 2　Code Pipeline の実行時間
Table 2　Execution Time for Code Pipeline

フェーズ	処理内容	処理時間
Source	Amazon S3	1 秒
Build	Docker Buildx (amd64, arm64)	22 分 17 秒
Staging	Singularity Build (amd64, parallel)	6 分 20 秒
	Singularity Build (arm64, parallel)	10 分 17 秒
全体		32 分 37 秒

表 3　マルチ CPU プラットフォームイメージの比較
Table 3　Multi CPU platform Images

	amd64	arm64
イメージサイズ	913.55 MB	870.50 MB

表 4　最適化ビルドイメージの比較
Table 4　Optimized Build Images

	Intel	AMD	Graviton2
ビルド時間	37 分 23 秒	5 分 48 秒	7 分 36 秒
イメージサイズ	8527.15 MB	714.98 MB	941.59 MB

1 分あたり 0.005 USD を必要とするが．現在，1 ヶ月 100 分までの無料枠が与えられている．

5.3　最適化ビルドの検討

最後に最適化ビルドイメージに関する評価を**表 3** と**表 4** に示す．Intel プロセッサでは，全ての機能を統合した Intel oneAPI を使用しているため，ビルド時間及びイメージサイズともに大きくなっている．これは，一般的に Docker で用いられている開発用イメージと配布用イメージを分離するマルチステージビルドを用いれば，大幅に改善可能であると考えているが，oneAPI では一般的に開発環境と実行環境が融合しているため，分離が難しい．実際，Intel 公式の One API HPC Kit イメージは圧縮後で 7.12 GB となっている．その他の AMD と Graviton2 では，汎用イメージとほぼ同等のサイズとなっている．それぞれの実行時間の改善効果については，今後の課題とする．

6.　議論

- **性能可搬性と再現性**：AWS では，Nitro System と EFA による独自の仮想化技術およびインターコネクトを採用しており，AWS 上での評価結果がどの程度，他の大型計算機システムでの性能可搬性や再現性があるのかという問題がある．例えば，EFA では，同一 Placement Group 内で単一フローが最大 10Gbps，複数フローの総計で 100Gbps となっている．
  本制約は，単一ノードに閉じた OpenMP ジョブや，パラメータサーベイなどのアレイジョブでは問題とならず，研究を AWS で完結させる方法もある．また，Azure や OCI などのベアメタルと Infiniband をベー

スとしたインスタンスを当初より使用する方法もある．

- **費用の最適化とベンダーロックイン**：学術機関の大型計算機システムとパブリッククラウドでは，利用に関する費用のコストモデルが異なり，同一の計算資源量を消費した場合に，一般的にパブリッククラウドの方が高額となる．スポットインスタンスの活用，細粒度の起動や停止，インスタンス性能やストレージ容量などのサイジング，サーバレスなどの計算機以外のサービスの活用により，初めて同水準とすることができる．これらの最適化を行うほど，そのノウハウは特定のパブリッククラウドサービスで固定的となり，ベンダーロックインにも繋がる．また，AWS には ARN (Amazon Resource Name) や IAM といった全サービス共通の独特の概念があり，パブリッククラウドへの移行や，デバッグの妨げとなることがある．
- **複数アーキテクチャスパコンとの比較**：近年，研究者の多様な要求から，複数アーキテクチャで構成された大型計算機システムを導入する学術機関が増加している．特に，大阪大学の SQUID/OCTOPUS などでは，共通のポイント制度を導入し，相互に融通可能となっている．これらの大型計算機システムで当初より研究開発すれば，複数アーキテクチャ対応のアプリケーションを低廉な利用負担金で開発することができる．しかしながら，ハードウェアの世代や型番は，数年に一度のサイクルの調達時に固定され，相互にトレードオフがある．

7. おわりに

本研究では，パブリッククラウドを起点として HPC アプリケーションの初期開発を行い，開発が進行した段階で，複数の研究機関が提供する大型計算機システムへアプリケーションを水平展開する開発手法 “microburst” を提案した．

今回はサービスの組み合わせによる機能的な検証に留まったが，今後は費用や性能面での評価の充実や．実アプリケーションでの評価を実施する予定である．また，現在は主に手動で環境を構築していることから，CloudFormation などのテンプレートによる環境構築の自動化を行い，他の研究者にも機能を提供していく予定である．

謝辞 本研究は JSPS 科研費 20K11837「高性能計算技術とマイクロサービス化技術の融合に関する研究」の助成を受けたものです．

参考文献

[1] L. Shalev et al.: A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC, *IEEE Micro*, Vol. 40, No. 6, pp. 67–73 (2020).
[2] Amazon Web Services, Inc.: AWS Graviton Processors. https://aws.amazon.com/jp/ec2/graviton/.
[3] Bechman, R.: Exascale and Big Data Convergence, *SC'15 BDEC Workshop* (2015).
[4] HPCwire Japan: HPC in the Cloud –クラウドにおける HPC の未来– (2021).
[5] M. A. S. Netto et al.: HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges, *ACM Computing Survey*, Vol. 51, No. 1, pp. 1–29 (2018).
[6] G. Guidi et al.: 10 Years Later: Cloud Computing is Closing the Performance Gap, *ICPE'21*, ACM, pp. 41–48 (2021).
[7] P. D. Tommaso et al.: Nextflow enables reproducible computational workflows, *Nature Biotechnology*, Vol. 35, pp. 316–319 (2017).
[8] Gartner, Inc.: Gartner Says Worldwide IaaS Public Cloud Services Market Grew 37.3% in 2019. https://www.gartner.com/en/newsroom/press-releases/.
[9] Amazon Web Services, Inc.: High Performance Computing Lens – AWS Well-Architected Framework. https://docs.aws.amazon.com/ja_jp/wellarchitected/latest/high-performance-computing-lens/.
[10] Amazon Web Services, Inc.: Parallel Cluster. https://aws.amazon.com/jp/hpc/parallelcluster/.
[11] Amazon Web Services, Inc.: Amazon Elastic Compute Cloud – Spot Fleet. https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/spot-fleet.html.
[12] Amazon Web Services, Inc.: GitHub – Getting started with AWS Graviton. https://github.com/aws/aws-graviton-getting-started.
[13] 大阪大学サイバーメディアセンター：大型計算機システム SQUID. http://www.hpc.cmc.osaka-u.ac.jp/squid/.
[14] Rao, C.: Using Elastic Fabric Adapter (EFA) to scale HPC workloads on AWS, *AWS re:Invent 2019 (CMP408-R)* (2019).