Vol.2022-HPC-185 No.18
2022/7/28

データ利活用に向けた高性能Kubernetes環境構築の検討

杉木 章義[1,a)]

概要：近年，様々な分野における機械学習や大規模データ解析手法の普及に伴って，従来のスパコンや学術クラウドでは吸収できないワークロードが急速に増加している．データ活用社会創成基盤 mdx はその環境改善に向けた試みであるが，それに加えて，本研究では，従来のジョブスケジューラによるバッチ環境と並行して，ソフトウェア基盤を Kubernetes に置き換え，従来のスパコン性能に近い高性能な Kubernetes 環境を学際的かつ学術利用者向けに提供可能であるかどうか検討する．本研究では，mdx の備えるハードウェア・ソフトウェア性能や機能を最大限活かしながら，k8s-configs と呼ぶ Ansible による自動構築スクリプトのプロトタイプ環境を構築し，評価を実施する．学術分野における Kubernetes の活用は，ゲノム分野等の一部研究分野で先行しているが，本研究では特定の分野に依存しない学際的なサービス提供を目標として研究を進める．

Building a High-performance Kubernetes Cluster for Cross-disciplinary Data Collections and Utilizations

1. はじめに

近年，学術研究分野においてスーパーコンピュータに代表される高性能計算（HPC）システムや既存のアカデミッククラウドでは対応できない計算機システムへの要望が増加している．その必要性は，米国においても National Discovery Cloud（NDC）の実現を訴えるホワイトペーパーで言及されており [1]，クラウド基盤，データに基づく探索，シミュレーションが 21 世紀における研究の三つの柱として掲げられている．

データ活用社会創成プラットフォーム（mdx）[2] は，日本国内におけるその状況の改善に向けた試みの一つであり，Society 5.0 に対応する産学官の連携による学際的なデータ利活用に向けて，東京大学や国立情報学研究所（NII）などを中心に 11 機関[*1]が参画し，共同で運用している．また，mdx に先行するシステムとして，産業技術総合研究所が運用する大規模 AI クラウド計算機システム（ABCI），北海道大学が運用する北海道大学ハイパフォーマンスインタークラウド，東京大学の Wisteria，大阪大学のデータ集約基盤 ONION などで，従来の HPC システムの枠組みにとらわれない計算資源提供の試みが行われている．米国においても Cameleon Cloud や CloudLab などで，既存のパブリッククラウドでは対応できない Research Cloud を実現している．

mdx に代表される最新の計算機システムは高性能な計算環境を提供するが，依然として計算機利用者としての研究者の期待と，提供者側が提供できる計算資源の間にはギャップがあり，さらに近年，そのギャップが急速に広がりつつある（**図 1**）．例えば，運用開始時点での mdx では，提供する計算資源の基本単位は仮想マシンとなっており，仮想マシン上でのアプリケーションやライブラリの導入や設定は利用者に任されている．一方で，特に機械学習・人工知能（AI）分野を中心に，PyTorch，TensorFlow，Keras などの非常に多数のフレームワークやライブラリ，データセットなどが提供されており，その環境整備に多大な時間や労力を要する．また，データの収集から解析まで行うためには，Kafaka，Spark，MySQL などのストリーム処理基盤やデータベースの整備も必要である．さらには，mdx などの計算機システムが持つ高い性能を引き出すためには，その特性に配慮したソフトウェアの導入や最適化が必要である．将来的には，NDC で指摘されているように教育プログラム及びカリキュラムの整備や，計算資源の提供のみ

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

[*1] 北海道大学，東北大学，筑波大学，東京大学，東京工業大学，名古屋大学，京都大学，大阪大学，九州大学，国立情報学研究所，産業技術総合研究所

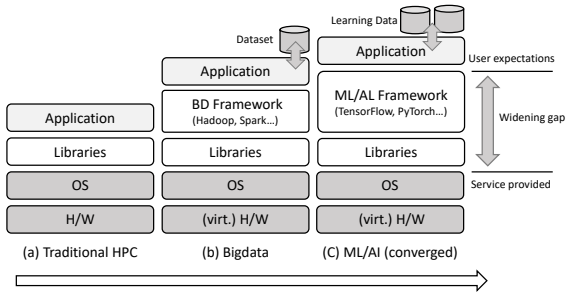

図 1 現世代の HPC システムにおける需要と供給のギャップ
Fig. 1 A widening gap between short supply and huge demand in current-generation HPC systems.

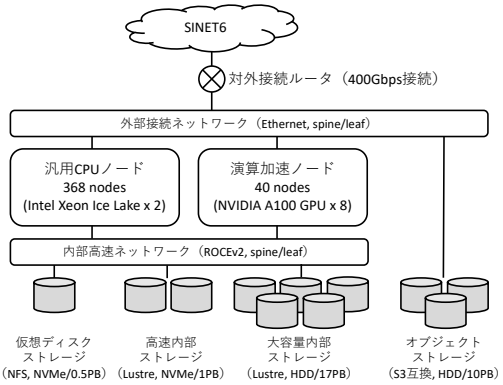

図 2 mdx のシステム概要
Fig. 2 System overview of the mdx computing platform.

に限らない研究支援サービスの拡充，それを支える人員の増強なども必要である．

本研究では，mdx を主に想定し，現世代の計算環境に最適化した Kubernetes 環境の構築を目指す．本研究は前述の全てのギャップを解消することはできず，それに向けた最初のステップである．本研究の貢献を以下に示す．

- **高性能 Kubernetes による実現可能性の探究：** mdx 上に**高性能な Kubernetes 環境**（High-Performance Kubernetes clusters: HPK）のプロトタイプを構築することで，従来のジョブスケジューラによらない HPC システム実現可能性を検討する．mdx は従来のスーパーコンピュータに近い高性能な CPU や GPU，RDMA 通信可能な高速インターコネクト，並列分散ファイルシステムを提供しており，将来的な計算機システムの検証環境として最適である．アプリケーションの整備は，Kubernetes が提供する Helm や Operator などで行う．
- **mdx に最適化した高性能 Kubernetes 環境の提供：** より直接的に，mdx に最適化した Kubernetes 環境一式を mdx の利用者に提供する．本研究の成果は，mdx におけるベストプラクティスとして活用できるのと同時に，Ansible による自動構築スクリプトも提供する．これは，mdx における**マネージド Kubernetes** の提供に相当する．

本研究の成果は，mdx が提供する仮想マシンによる標準サービスを完全に置き換えるものではなく，Kubernetes による新たなサービス層を付け加え，利用者にもう一つの選択肢として提供する．また，上記のギャップ解消に向けては，NII が提供する VCP[3] や SINET Stream[4] など他の試みも行われているが，本研究はこれらと直交するものである．利用者に対して，同種の目標に対して，複数の選択肢を提供することが重要であると考える．

本研究は，先行研究 [5] を発展させたものである．先行研究は mdx 全体のシステムと性能の概要について述べたものであり，本研究では，Kubernetes 環境のみに焦点を絞り，その内容も発展させている．また，先行研究の時点では，人手によるプロトタイプ環境の構築を行なったが，Ansible による自動環境構築も可能となっている．

2. 本研究の前提

2.1 mdx の概要

mdx は既に多くの文献 [2], [5] で解説されているが，ここでは本研究に関係する部分について説明する．

mdx のシステム概要を**図 2** に示す．mdx は大きく分けて，計算ノード，ストレージ，ネットワークの三つで構成されている．

- **計算ノード：** mdx では，**汎用 CPU ノード**，**演算加速 GPU ノード**の 2 種類が提供されている．汎用 CPU ノードとして Intel Xeon（Ice Lake 世代）の CPU を 2 ソケット搭載したサーバが 368 台提供されており，演算加速 GPU ノードでは，それに加えて NVIDIA A100 GPU を 8 機搭載したサーバが 40 台提供されている．mdx では，仮想マシンが基本提供単位となっており，それぞれ **CPU パック**と **GPU パック**が用意されている[*2]．サーバ仮想化には，VMware vSpere/ESXi が使用されており，GPU を使用する場合にデバイス・パススルーを実施するなど，高速性に配慮されている．Intel SGX の利用も一部のノードで開始された．
- **ストレージ：** プロジェクト内でのデータ共有を想定した内部ストレージとして，Lustre ファイルシステムに基づく**高速ストレージ**，**大容量ストレージ**の 2 種類が提供されている．それぞれ，NVMe，HDD で実現されており，使用可能な容量やアクセス性能が異なる．Lustre ファイルシステムでは，各 VLAN に異なるサブディレクトリを割り当てるマルチテナント機能を使用しており，プロジェクト間で分離されている．また，データの利活用に向けた外部公開も想定して大容量の S3 互換の**オブジェクトストレージ**も提供されて

[*2] 2022 年 6 月時点で，1 CPU パックは 1 仮想 CPU，仮想メモリ 1.5GB の組み合わせ，1 GPU パックは 18 仮想 CPU，仮想メモリ約 57.6GB，GPU 1 基の組み合わせ．

いる．さらには，仮想マシンのディスクイメージを格納する**仮想ディスクストレージ**も存在している．

- **ネットワーク：** SINET6 を経由してインターネットに接続する**外部接続ネットワーク**と，仮想マシン間及びストレージと高速に接続する**内部高速ネットワーク**の 2 種類で構成されている．SINET と外部接続ネットワークの接続は運用開始時点では 200Gbps であったが，SINET6 に移行する段階で 400Gbps に増速されている．SINET の L2VPN サービスによる他拠点との接続も可能であり，産業技術総合研究所 ABCI などの計算機システムと既に接続されている．インターネット外部から利用者テナント内の仮想マシンへの接続を受け付けるには，**DNAT** によるグローバル IP とプライベート IP のマッピング，**ACL** 設定によるポート開放を行う必要がある．
  内部高速ネットワークでは，ROCEv2 に基づく RDMA 可能な Ethernet 技術が採用されている．実際に RDMA 通信を行うためには，VMware 標準スイッチを使用する **Port Group** から，**PV-RDMA**，**SR-IOV** に仮想マシンのネットワーク設定を変更する必要がある．

ソフトウェア環境として，mdx では，仮想マシンへの OS やアプリケーション導入を支援するために，**仮想マシンテンプレート**と **machine-configs**[6] が提供されている．仮想マシンテンプレートでは，東京大学による推奨版として Ubuntu 20.04 が提供されており，デスクトップ版とサーバ版，CPU 対応版と GPU 対応版で 4 種類の組み合わせが存在している．仮想マシンテンプレートでは，mdx に最適化したドライバ，設定などが予め導入されている．machine-configs は複数マシンを展開するための Ansible スクリプトであり，NFS サーバ，LDAP サーバ，Jupyter Lab，OpenMPI に基づくクラスタ環境を導入する．

最後に mdx の利用申請に関して，**お試しプロジェクト**，**通常プロジェクト**，**セキュアプロジェクト**が存在している．セキュアプロジェクトは，通常プロジェクトよりもテナント間の分離を強化し，より一層セキュリティやプライバシーに配慮されるが，提供可能数に大きな限りがある．

2.2 本研究の目標

近年，従来のジョブスケジューラに基づく HPC システムに，大規模データ処理，AI・機械学習のワークロードを取り込む試みが数多く行われている．北海道大学情報基盤センターでも，人工知能対応先進的計算機システム [7] が運用されているが，下記の運用上の課題がある．

- **対話環境への対応：** 大規模データ処理や機械学習・AL 分野では，Jupyter 環境が広く用いられており，対話的な試行錯誤が行われている．対話処理はジョブスケジューラのバッチ処理との相性に問題があり，工夫が必要である．一般にジョブスケジューラには，インタラクティブジョブの仕組みがあるが，利用に制限がある．
- **計算機利用率の向上：** 多くのスーパーコンピュータでは，バッチジョブに対してノード全体を占有して提供する形態で運用されており，豊富な計算能力を提供する一方で，同時に受け入れ可能なユーザ数や利用率に課題がある．特に，NVIDIA A100 や H100 などの GPU，特にマルチ GPU，マルチノードの場合には，非常に高い性能を提供する一方で，計算資源の使い切りには課題を有する．
- **計算と I/O の資源使用比率：** 大規模データ処理，特に ETL 処理では，多数のノードに I/O 処理を分散し，並列アクセスによる高帯域を確保した方がよい可能性がある．一方で CPU や GPU 性能に関しては，それほど必要ない可能性があり，ノード占有では負担金が高額となる．
- **管理者権限を必要とするソフトウェアの導入：** 管理者権限の利用可否については，スーパーコンピュータとアカデミッククラウドの比較でよく議論される．ソフトウェアの導入については，Docker や Singularity などのコンテナ技術の採用で改善される．
- **研究成果のサービス化：** バッチ処理で，十分な研究成果が得られた場合，そのアプリケーションをサービス化して，広く一般に提供したいという要望が出る可能性がある．ジョブスケジューラでは，資源が用意され，アプリケーションが実行されるまでに時間差があり，サービス化に課題がある．

近年，多くのスーパーコンピュータでコンテナ技術が採用されているが，従来のジョブスケジューラの配下で Singularity コンテナを実行する運用がデファクトスタンダードになりつつある．一方で，本研究ではスーパーコンピュータと同等のハードウェアで，ジョブスケジューラを Kubernetes に置き換え，Kubernetes 導入による HPC システム実現の可能性を探索する．

HPC システムにおける Kubernetes の利用はゲノム分野など一部の研究領域で先行して検討されている．また，Kubernetes は NII が提供する WEKO3 や GakuninRDM のサービス構築基盤としても利用されている．さらには，AI や機械学習に対応した AI スーパーコンピュータの文脈で，民間企業を含む各機関で Kubernetes による HPC システムの運用が開始されている．しかしながら，大型計算機センターでのサービス提供には依然として課題があり，本研究での課題の解決を目指す．

3. 高性能な Kubernetes 環境の構築

本研究では，運用開始時点の mdx の調達には含まれていないコンテナおよび Kubernetes を基礎としたデータ利

活用に向けたソフトウェア基盤整備を行う．

本項目では，単に mdx 環境上で Kubernetes を稼働させるだけではなく，下記の検証を目標とする．

- mdx が備える高性能なサーバ，ネットワークおよびストレージを最大限活用し，従来のスパコン性能に近い高性能な Kubernetes 環境を実現する．
- 上記の Kubernetes 環境上で，データ収集・解析，機械学習や AI，HPC アプリケーションおよびそれらのワークロードの混在に関する研究を実施する．
- 学術的なアプリケーションコンテナの流通やコンテナ間のサービス連携による mdx を中核とした複数学術機関の連携，および学際的な研究環境の構築を実施する．
- mdx のみに限らず，将来的な学術基盤整備に関する知見を得る．

上記の目標には長期的なものも含まれているが，本論文では，これらの検証の初期の成果について報告する．

4. 実装

本研究では，mdx 上で高性能 Kubernetes 環境を自動構築するための Ansible スクリプト，k8s-configs[8] を実装した．k8s-configs は machine-configs から大きく影響を受けているが，実装は共有していない．当初は machine-configs の実装から派生する予定であったが，共有できる部分が少なく，独立したスクリプトとして記述することとした．

k8s-configs はそのまま，mdx 専用のマネージド Kubernetes 環境を提供する．mdx が提供する CPU や GPU，Lustre ストレージ，ROCEv2 ネットワークなどの各機能を最大限活用し，最適化された Kubernetes 環境を提供する．

k8s-configs は mdx のユーザポータルから手動で仮想マシンを構築し，SSH ログイン可能となるように設定された状態からの自動構築を行う．本負担は依然として利用者にとって大きいことから，将来的には，別の共同研究プロジェクトで開発された mdx REST API を使用し，完全自動化も目標とする．

4.1 仮想マシン構成

k8s-configs は利用者の環境に応じて，さまざまな規模の Kubernetes クラスタを自動構築するスクリプトであるが，仮想マシンの最小構成を**図 3** に示す．仮想マシンは Kubernetes クラスタを構成するサーバと，踏み台サーバ及びレジストリサーバの周辺サーバで構成されている．

周辺サーバでは，SSH で mdx 外部との接続性を確保する**踏み台サーバ**（`bastion`）1 台と，**プライベートコンテナレジストリサーバ**（`harbor`）1 台で構成されている．

Kubernets クラスタでは，**マスタノード**（`kube-master`）を 1 台配置し，計算ノードとなる **CPU ノード**（`kube-node`）と **GPU ノード**（`kube-gpu`）で構成されている．マスタノードは現在，小規模な利用を想定し，計算資源の有効活用の観点から 1 台のみとしているが，背後の実装で使用している k0s が複数マスタに対応しており，若干の修正で複数マスタでの冗長化に対応可能である．

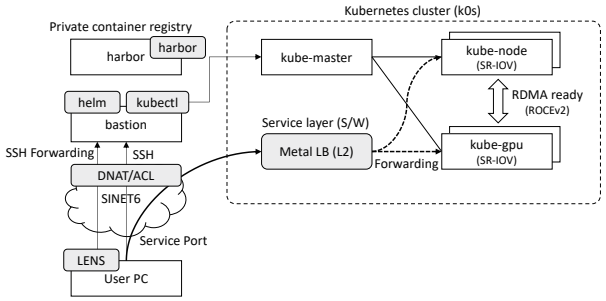

図 3 k8s-configs の仮想マシン構成
Fig. 3 Composition of virtual machines for k8s-configs.

各計算ノードは必要最小限の台数としており，必要に応じて増減させることが可能である．特に mdx では，ノード数の増減も容易であるが，CPU コア数や GPU 数の増減も仮想マシンを停止させるだけで可能であり，小規模インスタンスで検証を行い，144 仮想コアなどの mdx の各物理ノードをほぼ専有するようなインスタンスで，本格計算を行うことも可能である．

また，各計算ノードは性能面への配慮とノード間の RDMA 通信から，SR-IOV 対応の仮想マシンを使用する．マスタノード及び周辺サーバは，資源の有効利用の観点から Port Group 対応の仮想マシンを使用する．

仮想マシンテンプレートは，東京大学が提供する Ubuntu 20.04 Server 用の CPU 版および GPU 版を使用する．これらのテンプレートでは，mdx 用の各種ドライバや設定が予め導入されており，本研究では既存の成果を最大限活用し，その上に環境を構築する．

DNAT および ACL 設定によるインターネットからの外部接続性は，原則として踏み台サーバに対する SSH 接続のみとする．k8s-configs が構築する Grafana，Harbar などの各種 Web ポータルへのアクセスは，踏み台サーバを経由した SSH ポートフォワーディングで行う．Kubernetes 上で稼働する各アプリケーションサービスに対しては，後述の Metal LB を経由して，DNAT 及び ACL 設定で必要に応じて接続を割り当てる．

4.2 ソフトウェア構成

k8s-configs で使用するソフトウェア構成を**表 1** に示す．以下，主要なものについて解説する．

- **Kubernetes ディストリビューション：** 本研究では，高性能な HPC システムを目指すという観点から，各計算ノードでの CPU 使用率やメモリ使用率，使用するストレージ容量などのオーバヘッドが小さい軽量 Kubernetes ディストリビューションを使用する．軽量ディストリビューションには，k0s，k3s，MicroK8s

表 1　k8s-configs のソフトウェア構成
Table 1　List of software components used in k8s-configs.

構成要素	コンポーネント
Kubernetes	k0s
CRI	containerd（CPU ノード）
	NVIDIA GPU Operator（GPU ノード）
CSI	Exascaler File CSI Driver
CNI	Kube-router（フロントエンド用）
	NVIDIA Network Operator（ROCEv2 用）
負荷分散	Metal LB (L2 モード)
監視	Kube Prometheus stack
運用	LENS
HPC サービス	MPI Operator (Kubeflow)
ストリーミング	Spark on k8s Operator
	(Spark RAPIDS/UCX Shuffle Manager)
	Strimzi Kafka Operator
DB サービス	MySQL Operator

などのさまざまな選択肢があるが，本研究では k0s を使用した．

- **コンテナランライム（CRI）：** CPU ノードでは, k0s 標準の containered を用いるが，GPU ノードでは NVIDIA Container Toolkit を使用する．GPU 関係のコンポーネントの導入にあたっては，NVIDIA GPU Operator を使用するが，GPU ドライバは仮想マシンテンプレートで提供されるものを用いる．GPU Operator の使用により，NVIDIA A100 が提供する MIG（マルチインスタンス GPU）も Pod から使用可能となっている．
- **コンテナネットワーク（CNI）：** 外部接続ネットワークに接続するフロントエンドでは，k0s 標準の Kube-router を用いるが，Multus CNI を導入し，Pod に複数のネットワークインターフェースを割り当て，RDMA 可能な ROCEv2 による内部高速ネットワークをセカンダリとして割り当てる．導入にあたっては，NVIDIA (Mellanox) Network Operator を使用した．フロントエンドに関して，将来的には eBPF を基礎とした Cilium など，他の CNI 導入や検証も行う．
  Kubernetes 上で動作する各サービスに対するロードバランサーとしては，mdx から負荷分散機能が提供されないため，Metal LB を導入する．Metal LB は利用者テナントの未使用の IP アドレス空間を使用し，L2 モードで運用する．
- **コンテナストレージ（CSI）：** DDN 社が提供する Exascaler File CSI Driver を使用し，Pod に対して，高速ストレージ（`/fast`）および大容量ストレージ（`/large`）から Pod に対して永続ボリュームを動的プロビジョニング可能とする．前述の両ストレージに対応する Storage Class を予め定義してある．
- **NUMA 対応：** NUMA 構成を意識した Pod 間の性能分離に関しては，Topology Manager，CPU Manager 及び Memory Manager を導入する．これらは標準コンポーネントであり，ある程度 NUMA を意識したコンテナ配置が可能となっているが，Intel CPU Manager for Kubernets (`cmk`) なども導入可能である．
- **コンテナレジストリ：** プライベートコンテナレジストリは Harbor を使用した．これは将来的な運用を考慮して選択するとともに，Trivy などのコンテナイメージの脆弱性診断サービスとの連携も想定している．Harbor 導入にあたっては，各計算ノードへのプライベート証明書の導入や信頼などの設定も自動的に行う．コンテナイメージの作成にあたっては，踏み台サーバに Docker を導入する．Kaniko や Scaffold などのコンテナビルドを支援するツールは多数出ているが，利用者との親和性を重視する．
- **アプリケーションの導入，運用・監視：** アプリケーションの導入は，kubectl，Helm，Operator のいずれの方法でも可能であるが，Helm または Operator での導入を推奨する．運用・監視は，Kube Prometheus Stack で導入する Prometheus 及び Grafana が使用可能である．また，GUI による運用監視ツール LENS から k8s-configs が構築するクラスタに接続して，kubernetes を管理することも可能である．

4.3　アプリケーション構成

アプリケーションに関しては，k8s-configs から分離して，別プロジェクトとして整備や検証を進める予定である．以下に概要を説明する．

- **HPC サービス：** MPI ジョブ実行のために，Kubeflow の一部として開発されている MPI Operator を導入する．MPI ジョブは数値計算の実行や Horovod などを利用したマルチノードでの機械学習に用いられる．機械学習ジョブはギャングスケジューリングの問題となる．
  しかしながら，Kubernetes 標準のバッチ機能は非常に基本的な限られた機能しか用意されておらず，アプリケーションのコンテナ化の作業が必要であり，従来のジョブスケジューラと比較して多くのジョブ記述を要する．この作業は利用者への負担が大きく，数値計算ジョブを Kubernetes に移行しただけでは，ほとんど移行の利点は得られない．単純なラッパースクリプトの実装によって負担を大幅に軽減することも可能であるが，依然として課題がある．
  近年，バージョン 1.21 以降の Kubernetes では，Indexed Jobs や Suspended Jobs などのバッチ機能の強化が進められており，これらをもとにした HPC スケジューラの実装・評価も進める予定である．
- **ストリーミング処理サービス：** mdx ではデータの利

表 2　実験環境のインスタンス構成
Table 2　Instance configurations on experimental setup.

ノード	パック数	ディスク	接続
bastion	4 CPU パック	100GB	Port Group
harbor	4 CPU パック	160GB	Port Group
kube-master	8 CPU パック	100GB	Port Group
kube-node	18 CPU パック	150GB	SR-IOV
kube-gpu	1 GPU パック	150GB	SR-IOV

活用を主要な目標としていることから，データの収集や解析に関する支援も必要である．本研究では，データ解析では Apache Spark，データ収集では Apache Kafka を整備する．

近年の Apache Spark は Kubernetes ネイティブ実行可能となっており，各タスクが Kubernetes 上の Pod として実行される．本研究では，Google 社が中心となり開発した Spark on k8s Operator を導入する．さらに，mdx では GPU 及び UCX が利用可能となっていることから，GPU 上でデータ解析処理を行う Spark RAPIDS, UCX ファブリックを利用してデータシャッフルを行う RAPIDS UCX Shuffle Manager を導入し，mdx の性能を最大限活用する．

Apache Kafka に関しては, 導入のしやすさから Strimzi 版の Apache Kafka を使用し，既に動作している．

- **フロントエンドサービス：** データ解析のフロントエンドとして，JuypterHub や BinderHub を今後整備する予定である．
- **データベースサービス：** MySQL Operator を使用し，MySQL データベースが既に稼働している．MySQL Operator は現時点においてまだ試験運用段階であり，本番運用は推奨されないが，利用可能である．

5. 実験

5.1 実験環境

実験では，図 3 のシステム構成で**表 2** のインスタンスを使用した．

5.2 実験結果：自動クラスタ構築時間

上記の実験環境にて，k8s-configs の自動クラスタ構築時間を測定した．実験に時間を要するため，多数回測定できないが，3 回測定した結果，7 分 55 秒，7 分 52 秒，7 分 55 秒で構築が完了した．インターネットからのパッケージのダウンロードなど，変動する要素も含んでいるが，構築時間はほとんど変わらなかった．構築に要する時間は，ノード数に依存して増加することが予想される．

5.3 実験結果：GPU 性能

MPI Operator の記述例として示されている TensorFlow Benchmarks（`tf_cnn_benchmarks`）にて，GPU ノードを２台使用し，実験を行なった．パラメータは，`ResNet50` モデル，バッチサイズ 64，`fp16` 使用，Horovod による値の更新を使用した．元の記述では，CUDA 10 を使用することから，mdx で導入されている CUDA 11 を使用するよう若干修正を行なっている．また，ワーカとマスタの起動順序の問題でジョブが失敗し，実行時間が増加することから，マスタ起動時に遅延を挿入している．

表 3　TensorFlow Benchmarks の結果
Table 3　Results of TensorFlow Benchmarks.

接続	スループット
Ethernet (frontend)	1933.02 images/sec
ROCEv2 (Socket)	2232.02 images/sec
ROCEv2 (IB)	3346.78 images/sec

本実験の結果を**表 3** に示す．Ethernet の外部接続（フロントエンド）ネットワークで通信を行なった場合には，1933.02 画像/秒のスループットが得られた．内部高速（バックエンド）ネットワークを使用し，ROCEv2 で Socket 通信を行なった場合には，2232.02 画像/秒と，やや高いスループットが得られている．IB を使用し，RDMA 通信を行なった場合には，3346.78 画像/秒と大幅に性能が向上した．

ROCEv2 ネットワーク使用時の注意点として，Network Operator を構成している K8s RDMA Shared Device Plugin が，個別の Pod に異なる GID インデックスを発行しており，Pod 間で通信を行うためには，起動時に GID インデックスを指定する作業が必要である．

6. まとめと今後の予定

本研究では，データ利活用に向けて mdx 上に高性能 Kubernetes のプロトタイプ環境を構築した．研究目標は二つあり, 従来のジョブスケジューラを Kubernetes に置き換え，Kubernetes を基礎とした HPC システム実現可能性の探索と，より直接な mdx におけるマネージド Kubernetes 環境の提供である．本研究成果は，k8s-configs として Ansible による自動構築スクリプトとして提供されている．

今後の課題は多岐にわたるが，その内のいくつかについて紹介する．

- **新規コンポーネントの研究開発：** 現在の k8s-configs は実際の運用に配慮し，既存のコンポーネントを極力組み合わせ，実現している．今後，数値計算，AI・機械学習・大規模データ処理などの目的に応じて，長時間・短時間ジョブの混在など，ワークロードの混在に対応できる新規のスケジューラなどの開発が必要となる可能性があり，研究を進める．
- **セキュリティ・プライバシーの検証：** 現状の k8s-configs は mdx の性能を最大限活用し，主に性能面での検証

を重視している．今後は UID/GID の適切な設定やアクセス権の管理によるセキュリティ，プライバシーの検証を進める予定である．

Kubernetes が基礎とするコンテナ技術は OS レベルでの仮想化技術に依存しており，非常に軽量である反面，OS 名前空間の分割やシステムコール層での制御に頼っており，適切な運用には課題がある．mdx では，強い分離を一部導入するとともに，Nested VM の実行が許可されており，実際に Firecracker によるマイクロ VM が起動した．今後，Kata Containers などの導入によるマイクロ VM によるコンテナ実行の検証も進める．

Intel SGX は mdx の運用開始時点では提供されていなかったが，一部ノードでの試験提供が開始されており，これについても検証を進める．

- **アプリケーションの整備：** 本研究はボトムアップアプローチにて，環境の整備を進めてきており，マネージド Kubernetes 環境が提供できた段階にある．一方で，多くの研究者の関心は，データセットから直ちに研究が始められる，むしろトップダウンアプローチに関心があると考えられる．今後は，SINET Stream との連携も目標とし，JupyterHub をフロントエンドとし，Kafka によるデータ収集から Spark による解析までの典型的な一貫した環境を提供することを目標とする．
- **動的な Kubernetes クラスタの構築：** mdx は現在, 試験運用段階にあるが，本格運用の際にはポイント制に移行することが，アナウンスされている．k8s-configs は固定サイズの Kubernetes クラスタを作成し，多数のジョブを流すことで計算資源を最大限活用するよう設計されているが，需要に応じて動的にクラスタを作成した方がよい可能性がある．AWS における Karpernter[9] のような仕組みが必要であるが，今後検討する．
- **他拠点 Kubernetes クラスタとの連携：** 今後，他の学術機関に導入される Kubernetes クラスタや EKS, AKS，GKE などのパブリッククラウド上の Kubernetes クラスタとの環境差異の吸収は Kustomize で可能と考える．また，先行研究の成果 [10] をもとに，SPIFFE/SPIRE を活用したマイクロサービス層における学際的なフェデレーションも検討する．

謝辞 本研究は JSPS 科研費 20K11837 の助成，学際大規模情報基盤共同利用・共同研究拠点および革新的ハイパフォーマンス・コンピューティング・インフラの支援を受けている（課題番号: jh220058）．また，データ活用社会創成プラットフォーム mdx を使用して研究を実施した．

参考文献

[1] Lopresti, I. F. D., Gropp, B., Hill, M. D. and Schuman, K.: A National Discovery Cloud: Preparing the US for Global Competitiveness in the New Era of 21st Century Digital Transformation, Technical report, A Computing Community Consortium (CCC) White Paper (arXiv:2104.06953) (2021).

[2] Suzumura, T., Sugiki, A., Takizawa, H., Imakura, A., Nakamura, H., Taura, K., Kudoh, T., Hanawa, T., Sekiya, Y., Kobayashi, H., Matsushima, S., Kuga, Y., Nakamura, R., Jiang, R., Kawase, J., Hanai, M., Miyazaki, H., Ishizaki, T., Shimotoku, D., Miyamoto, D., Aida, K., Takefusa, A., Kurimoto, T., Sasayama, K., Kitagawa, N., Fujiwara, I., Tanimura, Y., Aoki, T., Endo, T., Ohshima, S., Fukazawa, K., Date, S. and Uchibayashi, T.: mdx: A Cloud Platform for Supporting Data Science and Cross-Disciplinary Research Collaborations, Technical report, arXiv:2203.14188 (2022).

[3] 国立情報学研究所：Virtual Cloud Provider. `https://cloud.gakunin.jp/ocs/`.

[4] Takefusa, A., Sun, J., Fujiwara, I., Yoshida, H., Aida, K. and Pu, C.: SINETStream: Enabling Research IoT Applications with Portability, Security and Performance Requirements, *IEEE COMPSAC'21*, pp. 482–492 (2021).

[5] 塙　敏博，中村　遼，空閑洋平，杉木章義，田浦健次朗：データ利活用に向けた仮想化プラットフォーム mdx の基本性能評価，情報処理学会研究報告 2022-HPC-183 (7)，pp. 1–9 (2022).

[6] 東京大学：machine-configs: Configuring VMs in mdx with Ansible. `https://github.com/mdx-jp/machine-configs`.

[7] 北海道大学情報基盤センター：人工知能対応先進的計算機システム概要. `https://www.iic.hokudai.ac.jp/ai/overview/`.

[8] Sugiki, A.: k8s-configs. `https://github.com/a-sugiki/k8s-configs`.

[9] Amazon.com, Inc.: Karpenter: Just-in-time Nodes for Any Kubernetes Cluster. `https://karpenter.sh/`.

[10] 杉木章義：高水準なマイクロサービス層における複数ドメインを連携させたインタークラウド HPC 環境実現の検討，情報処理学会研究報告 2021-OS-153 (9)，pp. 1–6 (2021).