Vol.2017-HPC-162 No.23
2017/12/19

AIクラウドでのLinuxコンテナ利用に向けた性能評価

佐藤 仁[1,a)]　小川 宏高[1]

概要：ユーザ権限で動作するLinuxコンテナの一つであるSingularityに対して，演算性能，ネットワーク通信性能，メモリバンド幅性能，ストレージI/O性能などAI/ビッグデータ処理を模したワークロードや，分散深層学習フレームワークの一つであるChainerMNのワークロードを産総研AIクラウド上で実行し，予備的な性能評価を行った．その結果，Singularityを介して実行した場合においてもベアメタルな環境での実行と遜色のない性能が得られることがわかり，アプリケーションの実行に高い性能が求められるスーパーコンピュータのような共有計算機環境においても，ユーザ権限で動作するLinuxコンテナの利用が現実的であることがわかった．

1. はじめに

近年，AIに関する研究開発が，自動車の自動運転，製造業，ロボット，医療，創薬，金融などの様々な分野で広く盛んに行われている．とりわけ，多層構造のニューラルネットワーク (ディープニューラルネットワーク，DNN) を用いた機械学習手法である深層学習 (ディープラーニング) が注目されており，画像認識，音声認識，映像認識，自然言語処理などの幅広い応用が見込まれている．深層学習の基本的な概念は1980年代から存在しているものの，多段のDNNにより大量のデータから莫大な計算を行い特徴量を抽出する必要があったためこれまで目立った進展はみられなかったが，2010年代に入り，アルゴリズム (Algorithm Theory) の進展とともに，大量のビッグデータ (Big Data) を蓄えるストレージ技術や，それらのデータに対して高速に処理する計算能力 (Computation) が大幅に進展したため，比較的短時間で高精度な学習が実現されるようになり，三位一体となってブレイクスルーがもたらされている状況である．このため，従来はビッグデータ処理の基盤であったクラウドにも高性能計算由来のGPUアクセラレータの導入が積極的に行われており, また，従来は高性能計算によるシミュレーション基盤であったスーパーコンピュータにもデータ同化などビッグデータ処理の重要性が謳われるようになっており，これらの基盤を融合して設計されたAIクラウド [1], AIスーパコンピュータ [2] も登場しはじめている．

AIやビッグデータ処理においても高性能計算由来の技術が必要になる一方で，既存のクラウドとスーパーコンピュータのソフトウェアスタックには依然として大きな隔たりがある．とりわけ，分散深層学習においては，多くの場合，簡便なプログラミングインターフェースを提供しつつも高い性能を達成するために，Pythonを介して，CUDAやCuDNNなどGPUアクセラレータを活用して高速な演算を行うためのライブラリや，OpenMPIやNCCL2などGPUやInfiniband に最適化された通信を行うためのMPIライブラリ，さらに，Redis，LevelDBなどデータアクセスのためのKey Value Storeなどを実行する必要があり，複雑な依存関係がある多数のソフトウェアコンポーネントをインストールする必要がある．しかし，従来のスーパーコンピュータのような共有計算機環境では，通常，一般ユーザーに対してroot権限は与えられないため，任意のOSの選択ができない点や，rpmやaptなどのパッケージインストーラによる簡便なソフトウェアのインストールができない点など，ユーザのソフトウェア配備の生産性や再現性を阻害していることが大きな問題となっている．

Linuxコンテナは上記の問題を大幅に緩和することができる有望な技術である．現在，デファクトで広く用いられているDocker [3] は，クラウドでの利用に特化しており実行時にroot権限を必要としデーモンとして計算機上に常駐するため，スーパーコンピュータのような共有計算機環境での利用に向かないものの，近年，Singularity [4], Shifter [5], [6], CharieCloud [7] などユーザ権限で動作する共有計算機環境向けのLinuxコンテナが登場し始めている．その一方で，このようなユーザ権限で動作するLinuxコンテナをAIクラウド上でAI/ビッグデータ処理のワークロードを適用した場合の性能は明らかではない．

我々は，ユーザ権限で動作するLinuxコンテナの一つで

[1] 国立研究開発法人産業技術総合研究所
[a)] hitoshi.sato@aist.go.jp

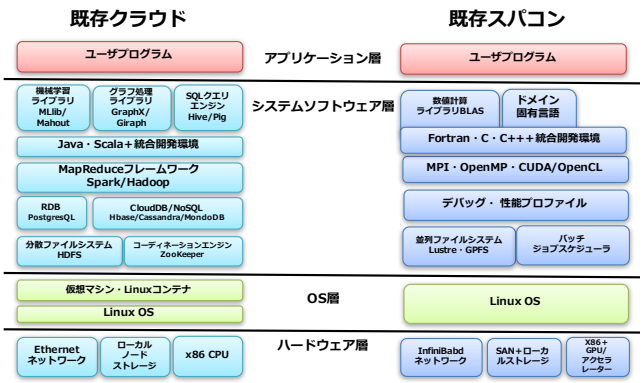


図 1　既存のクラウドとスーパーコンピュータのソフトウェアスタック

ある Singularity に対して，演算性能，メモリバンド幅性能，ネットワーク通信性能，ストレージ I/O 性能など AI/ビッグデータ処理を模したワークロードや，分散深層学習フレームワークの一つである ChainerMN [20] のワークロードを産総研 AI クラウド [8] 上で実行し，予備的な性能評価を行った．その結果，Singularity を介して実行した場合においてもベアメタルな環境での実行と遜色のない性能が得られることがわかり，アプリケーションの実行に高い性能が求められるスーパーコンピュータのような共有計算機環境においても，ユーザ権限で動作する Linux コンテナの利用が現実的であることがわかった．

2. AI クラウド

我々は，人工知能技術の研究開発を推進するため，ビッグデータ処理基盤であるクラウドと高性能計算によるシミュレーション基盤であるスーパーコンピュータとを融合した次世代の大規模計算環境である AI クラウドの設計・研究開発を進めている．この節では，AI クラウドに向けて既存のクラウドとスーパーコンピュータ上でのソフトウェアスタックの問題点を指摘し，AI クラウドのソフトウェアエコシステムの創出する上で必要な点を述べる．

2.1 クラウドとスーパーコンピュータのソフトウェアスタックの隔たり

AI クラウドでは既存のクラウドやスーパーコンピュータの両方の技術要素が必要であるが，一方で，既存のクラウドとスーパーコンピュータではソフトウェアスタックに関して大きな隔たりがある．大まかな概要を図 **1** に記す．

アプリケーション層

- クラウドではジョブの実行にインタラクティブな操作が必要であったり，ウェブを介してサービスを提供する必要があったりする．
- スーパーコンピュータではスケジューラを介したバッチジョブによる実行が多い．

システムソフトウェア層

- クラウドではデータ解析など頻繁にプログラムを書き換えるようなユースケースに特化し，Java, Scala など生産性が高いプログラム言語が広く使われているものの，高速化には向かない．分散データベースの利用が多く，利用用途に応じたマルチテナントな資源割当や REST API を介した資源制御を行う傾向がある．
- スーパーコンピュータでは Fortran, C, C++をはじめ MPI，OpenMP, CUDA/OpenCL などのようなマシンの性能を活かせるプログラム言語やランタイムが広く使われているものの，プログラムが難しく生産性が低い．ただし，BLAS などの数値演算をはじめとするカーネル処理は頻繁にプログラムを書き換える必要がない．また，しばしば数千・数万コア規模の計算機向けにデバッグ・性能チューニングが必要となる．スケジューラを介して資源割当を行うことが多い．

OS 層

- クラウドでは柔軟な資源管理を目的として仮想マシンや Linux コンテナなどにより資源の仮想化が行われることが多い．
- スーパーコンピュータではプログラムの高速性を重視し，資源の仮想化が行われずベアメタルな利用が多い．

ハードウェア層

- クラウドではウェブサーバー由来のコモディティな x86 サーバーや REST API を前提とした分散ストレージを採用していることが多い．
- スーパーコンピュータでは高速化に特化し，Infiniband のような超広帯域・低遅延なネットワーク，GPU のような演算アクセラレータ，バーストバッファのような I/O アクセラレータなど，新しい技術を積極的に採用していることが多い．

AI クラウドでは既存のクラウドやスーパーコンピュータの両方の技術要素を満たすソフトウェアエコシステムをいかに創出するかが重要な課題となる．

2.2 AI クラウドのソフトウェアスタック

我々は平成 29 年 6 月に産総研 AI クラウド [8] の運用を開始した．この運用を通じて，現状で我々が必要である AI クラウドのソフトウェアスタックの要求を図 **2** に記す．

アプリケーション層

- 機械学習，深層学習，グラフ処理などのフレームワークを簡単にバッチ処理やインタラクティブ操作で利用できる必要があり，また，場合によってはウェブを介してサービスを提供する必要がある．

システムソフトウェア層

- データ解析など頻繁にプログラムを書き換えるようなユースケースに特化し，Python, R などのスクリプト言語，Jupyter Notebook などのフレームワークを

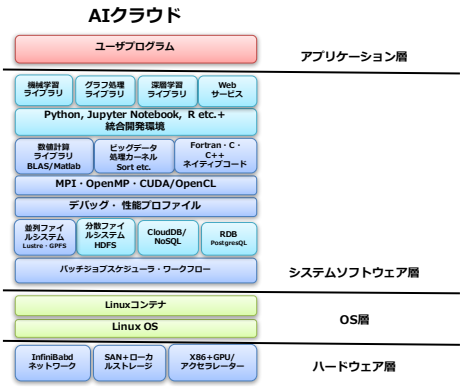


図 2　AI クラウドのソフトウェアスタック

利用しつつも，性能が必要な場合も多く，HPC 由来の BLAS などの数値演算カーネルや Sort, PrefixSum, Merge, PrefixSum などのビッグデータ処理カーネルを簡単に呼び出せる必要がある．また，場合によっては MPI や GPU などを用い並列処理による高速化を行う．

- 大規模なパラメータサーベイを行うため，ジョブの長時間実行やワークフローをサポートする必要がある．
- 画像，映像，テキストなどのデータに対する細粒度な I/O が多く，場合によっては秘匿性のあるデータへのアクセスが必要となる．

OS 層

- 多数のコンポーネントで構成された複雑なソフトウェアスタックの簡便に配備するために Linux コンテナが必要となる．ただし，プログラムの高速性も重視される．

ハードウェア層

- スーパーコンピュータ由来の Infiniband のような超広帯域・低遅延なネットワーク，GPU のような演算アクセラレータを採用しつつもコモディティなコンポーネントで構成されたサーバが用いられる．

分散深層学習をはじめとして AI/ビッグデータ処理のワークロードでは高性能計算由来の技術要素が必要であり，AI クラウドがスーパーコンピュータに類似した構成を採用することも選択肢の一つとして考えられる．一方で，AI/ビッグデータ処理では，複雑な依存関係がある多数のソフトウェアコンポーネントをインストールする必要があるのに対し，従来のスーパーコンピュータのような共有計算機環境では，通常，一般ユーザーに対して root 権限は与えられないため，任意の OS の選択ができない点や，rpm や apt などのパッケージインストーラによる簡便なソフトウェアのインストールができない点など，ユーザのソフトウェア配備の生産性や再現性を阻害していることが大きな問題となっている．

2.3　共有計算機環境上での Linux コンテナの利用と問題点

Linux コンテナは，アプリケーションとアプリケーションが必要とする実行環境をイメージとしてパッケージ化し，ホスト計算機の OS リソースから隔離された空間でプロセスを実行するための仮想化技術である．Linux コンテナを用いることにより柔軟にアプリケーションの配備と実行を行うことが可能になる．現在，Linux コンテナでは Docker [3] がデファクトで用いられており，NVIDIA GPU へ拡張した NVIDIA Docker [9] や NVIDIA Docker に特化したコンテナイメージを配布する NVIDIA GPU Cloud [10] なども登場してきている．しかし，Docker はクラウドでの利用に特化しており実行時に root 権限を必要としデーモンとして計算機上に常駐するため，共有計算機環境で任意のユーザに Docker の実行を許した場合には，

- 任意のユーザが任意のストレージ領域へアクセス可能になる
- 任意のユーザが計算機に対して任意の設定を行うことが可能になる

などのセキュリティ上の問題が発生するため，スーパーコンピュータのような共有計算機環境での利用に向かない．

3.　Singularity

近年，2.3 節で述べた欠点を解消し，ユーザ権限で動作する共有計算機環境向けの Linux コンテナ [4], [5], [6], [7] が登場し始めている．ここでは，その中でも，現状で利用事例が比較的多い Singularity [4] に着目する．

3.1　Singularity の概要

Singularity は，米国ローレンス・バークレー国立研究所を中心にオープンソースで開発されている Linux コンテナの一つである．Singularity は 2.3 節で述べた欠点をラップトップなど個人が root 権限を持つ環境で構築したコンテナイメージを共有計算機環境ではユーザ権限のみで実行することで解決している．具体的には，Singularity 向けに作成されたコンテナイメージのマウント，カーネル内で必要な名前空間の作成，コンテナ内へのホストの path の作成などを root 権限が必要な操作を SUID されたファイルを介して行うことで限定し，常駐デーモンを用意せず，資源管理をスケジューラに委ねることにより実現している．

Singularity の動作イメージを図 **3** に記す．まず，ユーザはラップトップなど個人が root 権限を持つ環境で Singularity コンテナのイメージを構築する必要がある．Singularity で Linux コンテナのイメージを構築するためには，sandbox や空のイメージファイルからインタラクティブにソフトウェアをインストールすることで作成する方法の他，Recipe と呼ばれるソフトウェアのインストール手順を記述したファイル (Singularity とする）から構築する方法，

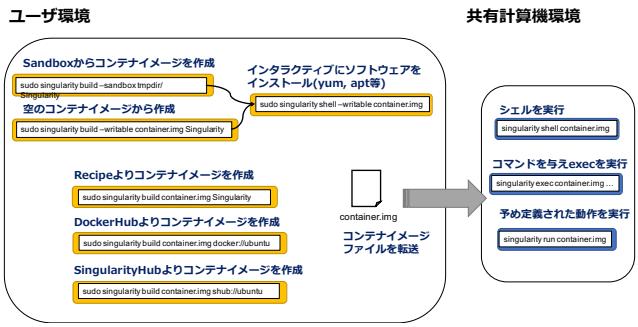


図 3　Singularity の動作イメージ

DockerHub [11] 上にホストされている Docker コンテナのイメージを利用して構築する方法，SingularityHub [12] と呼ばれる Singularity コンテナのイメージをホストしている Web サービスを利用して構築する方法がある．次に，構築された Singularity のコンテナイメージをファイル (container.img とする) として共有計算機環境へ転送する．最後に，共有計算機環境で Singularity コマンドによりコンテナイメージのファイルを用いてアプリケーションを実行する．Singularity では，インタラクティブな操作を行う shell モード，プロセスを置き換えて実行する exec モード，Recipe ファイルに記述された動作を行う run モードの 3 種類の実行方法をサポートしている．

3.2　HPC アプリケーションの実行

HPC アプリケーションを GPU アクセラレータや Infiniband ネットワークを用いて実行する際は各種デバイスドライバを必要とする．一方で，コンテナイメージにこれらのデバイスドライバをインストールすると，コンテナイメージの汎用性が損なわれることが問題となる．GPU の場合，ホスト側のデバイスドライバを Singularity コンテナ内にマウントすることで解決しており，実際，–nv オプションを用いることで，/.singularity/libs 以下に libcuda.so や libnvidia-ml.so をはじめとするホスト側のライブラリがコンテナ側へマウントされる．Infiniband の場合も同様のメカニズムで適用することができる．具体的には，Singularity を実行する際に，Infiniband のドライバをロードするように設定ファイル (/etc/singularity/init) に記述することで行うことができる．

HPC アプリケーションを MPI を用いて実行する場合は，コンテナ内に MPI をインストールし，コンテナ外から MPI を起動することで行うことができる．ただし，この際にコンテナ内とコンテナ外の MPI のバージョンなどの構成を同一にする必要がある．Python を介して MPI を起動する場合も同様で，コンテナ内にインストールされている MPI に対して mpi4py をコンテナ内にインストールし，コンテナ外から MPI を起動することで行うことができる．

表 1　AAIC の計算ノードのスペック

CPU	Intel Xeon E5-2630L v4 1.80GHz (10 cores, HT-enabled) × 2
GPU	NVIDIA Tesla P100 × 8
Mem	256 GiB
SSD	Intel DC S3510 480 GB × 1

4.　実験

ユーザ権限で動作する共用計算機環境向けの Linux コンテナを AI/ビッグデータ処理のワークロードを適用した場合の性能を明らかにするための実験を行った．具体的には，ユーザ権限で動作する Linux コンテナの一つである Singularity に対して，演算性能，メモリバンド幅性能，ネットワーク性能，ストレージ I/O 性能など AI/ビッグデータ処理を模したワークロードや，分散深層学習フレームワークの一つである ChainerMN のワークロードを産総研 AI クラウド上で実行し，予備的な性能評価を行った．

4.1　実験環境

実験は全て産総研 AI クラウド AAIC 上で行った．**表 1** に計算ノードのスペック，**図 4** に計算ノードの内部構成を示す．1 つの NUMA ノードにつき CPU が 1 ソケット, 128 GiB のメモリ, 4 台の GPU が属し，一方の NUMA ノードにはローカルストレージとして SSD が属し，もう一方の NUMA ノードには EDR Infiniband HBA が属している．計算ノード間は EDR Infiniband により Director Switch を介して Full-bisection Fat Tree 構成で接続されている．共有ファイルシステムは GPFS v4.2 であるが，今回の実験では直接は使用していない．

計算ノードの OS は CentOS 7.3 で構成され，Linux のカーネルは v3.10.0 である．使用した Singularity のバージョンは v2.4 であり，実験の都合上 4.6 節のみ v2.3.2 を用いている．Linux コンテナイメージの OS は Ubuntu16.04 とした．各種ベンチマークプログラムのビルドには OS のデフォルトの gcc コンパイラを用い，ベアメタル環境では v4.8.5，Linux コンテナ環境では v5.4.0 であった．また，glibc はベアメタル環境では v2.17, Linux コンテナ環境では v2.23 であった．両環境とも CUDA v8.0.61.2, CuDNN v6.0.21, NCCL v2.0.5 を用い，MPI は OpenMPI v2.1.1 を用いた．

4.2　演算性能

演算性能を計測するために BLAS(Basic Linear Algebra Subprograms) ライブラリの GEMM(General Matrix Multiplication) カーネルを GPU と CPU を対象に実行した．GPU 上での実行では，CUDA Toolkit 8.0.61 に付属する cuBLAS [13] を用いて半精度 (FP16)，単精度 (FP32)，倍精

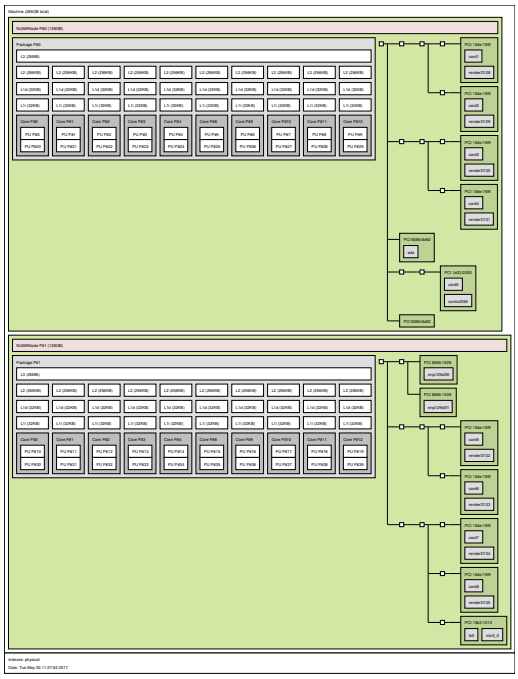


図 4　AAIC の計算ノードの内部構成

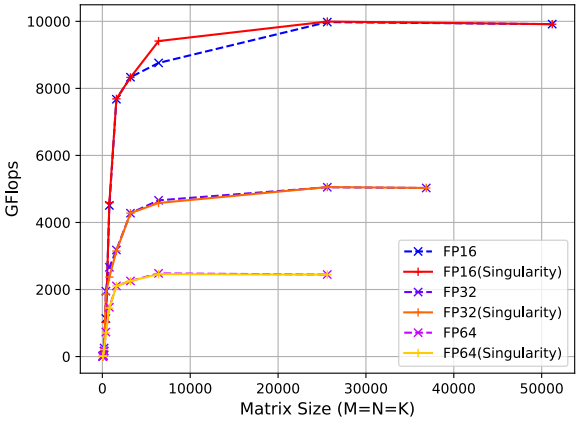


図 5　GEMM(GPU) の性能

度 (FP64) をベアメタルな計算ノード上で実行したものと Singularity の exec サブコマンドを介して実行したものとを比較した．CPU 上での実行では，Intel Math Kernel Library 2017.1.132 の CBLAS [14] を用いて単精度 (FP32)，倍精度 (FP64) をベアメタルな計算ノード上で実行したものと Singularity の exec サブコマンドを介して実行したものとを比較した．**図 5** に GPU 上の結果を示す．精度によらずベアメタル環境での実行でも Singularity を介した実行でもほぼ性能上の遜色がないことが伺える．同様に，**図 6** に CPU 上の結果を示す．行列のサイズが小さい場合に性能上のばらつきが発生するものの概ね性能上の遜色がないことが伺える．

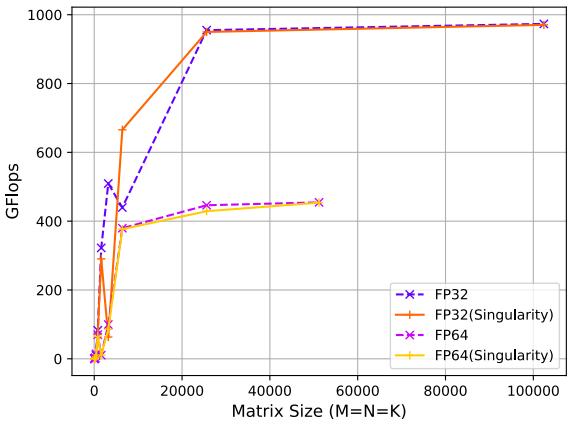


図 6　GEMM(CPU) の性能

4.3　メモリバンド幅性能

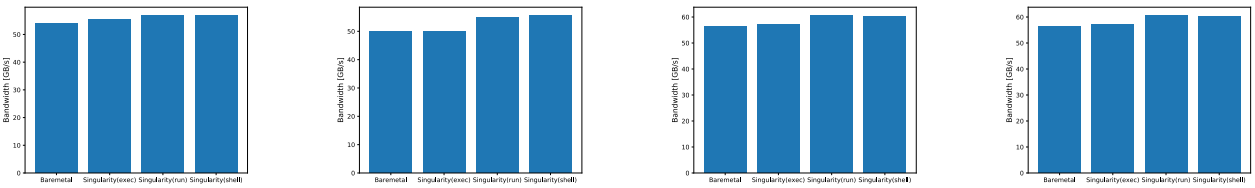
メモリの帯域幅を計測するために STREAM ベンチマークを実行した．今回は，バージニア大学のサイトより入手したコード [15] をベアメタルな計算ノード上で実行したものと Singularity の 3 種類のサブコマンド exec, shell, run を介して実行したものとを比較した．**図 7** に結果を示す．Singularity を介して実行した場合でもほぼ遜色のない性能であることが伺え，場合によっては Singularity を介して実行したほうが若干高い性能を示すことを確認した．これは，プログラムの実行に関して，Linux コンテナが必要最小限の最適なソフトウェアのみを含み余分なオーバーヘッドを少なくすることができるためであると考えている．

4.4　ネットワーク通信性能

ネットワーク通信性能を計測するために OSU Micro-Benchmarks [16] を実行した．今回は，MVAPICH のウェブサイトから取得した OSU Micro-Benchmarks v5.3.2 を用いて，ホスト計算機間及び GPU デバイス間のバンド幅，遅延，AllReduce の性能を 2 台のベアメタルな計算ノードを用いて実行したものと Singularity を介して 2 台の計算ノードを用いて実行したものとを比較した．**図 8** に結果を示す．**図 8(c)** において Singularity を介して実行した場合に良好な性能がみられたが，その他の場合においてはほぼ同等の性能を示した．

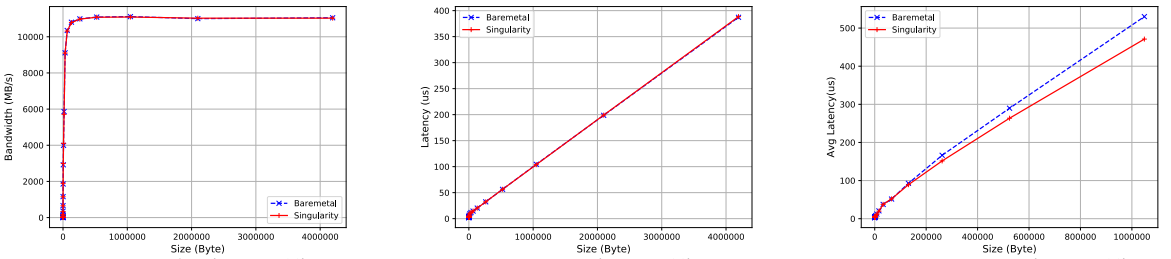
4.5　ストレージ I/O 性能

ストレージ I/O の性能を計測するために fio ベンチマーク [17] を実行した．今回使用した fio のバージョンは v2.19 である．まず，計算ノードのローカルストレージ (SSD) に対してブロックサイズを 128KB として I/O のスループットを計測し，ベアメタルな計算ノード上で実行したものと Singularity の exec サブコマンドを介して実行したものとで比較した．結果を**図 9** に記す．READ，WRITE

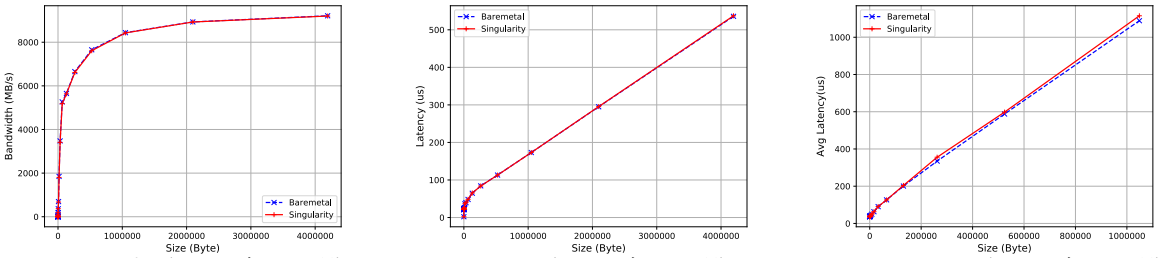


(a) Copy (b) Scale (c) Add (d) Triad

図 7 STREAM の性能

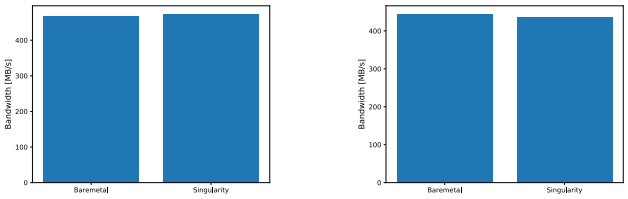


(a) バンド幅（ホスト間） (b) 遅延（ホスト間） (c) AllReduce（ホスト間）



(d) バンド幅（GPU デバイス間） (e) 遅延（GPU デバイス間） (f) AllReduce（GPU デバイス間）

図 8 OSU Micro Benchmarks の性能



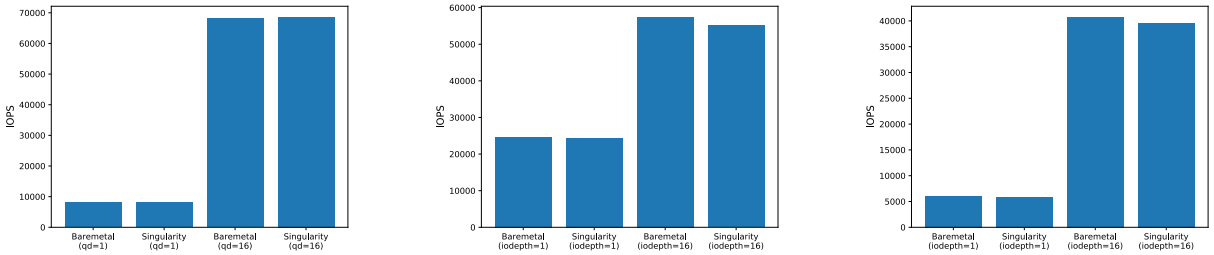
(a) READ (b) WRITE

図 9 fio の性能 (I/O スループット)

ともにほぼ同等な性能を示すことが伺える．次に，計算ノードのローカルストレージ (SSD) に対してブロックサイズを 4KB としてランダム I/O することにより IOPS を計測し，ベアメタルな計算ノード上で実行したものと Singularity の exec サブコマンドを介して実行したものとで比較した．この際，READ，WRITE の I/O の他に，READ と WRITE の I/O を 70%と 30%の比率で混合させた READ/WRITE のワークロードも実行し，iodepth を 1 と 16 と設定した．結果を図 **10** に記す．こちらも，READ，WRITE，READ/WRITE ともにほぼ同等な性能を示すことが伺える．

4.6 分散深層学習

分散深層学習の性能を計測するために ImageNet のデータセット [18] に対して ResNet-50 [19] による学習を行った．今回は，ChainerMN [20] を用いて，8 台の計算ノード (64GPU) を用いてベアメタルな環境で実行したものと Singularity の exec サブコマンドを介して実行したものとを比較した．ChainerMN は v1.0.0 を使用し，内部的に Chainer v2.1.0, Cupy v1.0.3，mpi4py v2.0.0 を使用している．また，Python は v3.6.1 を使用した．設定したパラメータは [21] に記載されているものと同等であり，バッチサイズを GPU あたり 32 (トータルで 2048)，学習率を 30 epoch 毎に 0.1 倍とし，Momentum SGD (momentum=0.9) の最適化を用い，Weight decay を 0.0001 として 100 epoch の学習を行った．図 **11** に結果を示す．他のユーザのジョブが実行されている共有計算機環境で実験を行ったため，ベアメタルな計算ノード上で実行した場合と Singularity を介して実行した場合とで振る舞いの違いがみられるものの，実行時間の点では大きな差異がみられないことが伺える．



(a) READ (b) WRITE (c) READ(70%)/WRITE(30%)

図 10 fio の性能 (IOPS)

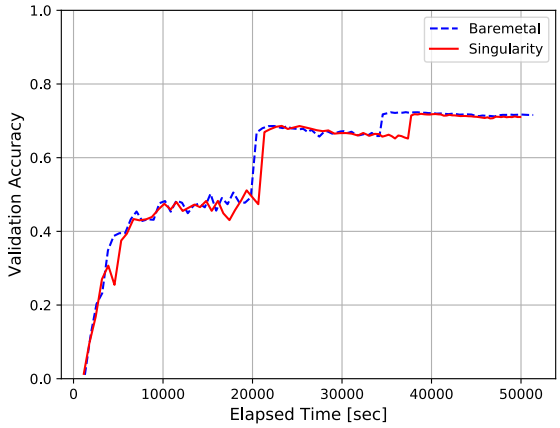


図 11 分散深層学習の性能

4.7 議論

演算性能，メモリバンド幅性能，ネットワーク性能，ストレージ I/O 性能など AI/ビッグデータ処理を模したワークロードをベアメタルな環境と Singularity を用いた Linux コンテナ環境で実行して性能を比較した．実験を通じて，概ね，Singularity を用いた Linux 環境でもベアメタルな環境と遜色のない性能を示し，アプリケーションの実行に高い性能が求められるスーパーコンピュータのような共有計算機環境でも，ユーザ権限で動作する Linux コンテナの利用が現実的であることがわかった．特に，今回の実験では，ベアメタル環境と Linux 環境で異なる OS(CentOS7.3 と Ubuntu16.04), 異なるコンパイラのバージョン (gcc v4.8.5 と v5.4.0), 異なる glibc のバージョン (glibc v2.17 と v2.23) など大幅に異なるソフトウェアを用いても問題なく実行することを確認した．また，AI/ビッグデータ処理に焦点をあて，ネットワーク通信性能やストレージ I/O 性能においてもベアメタル環境と Linux コンテナ環境で遜色のない性能を示すことが確認できた．4.6 節の分散深層学習の実験では性能にばらつきが発生してしまったが，各種マイクロベンチマークの結果から Linux コンテナ仮想化に由来するものであるとは考えておらず，他のユーザのジョブが実行されている共有計算機環境で実験を行ったこともあり，他の要因であると考えている．

スーパーコンピュータのような共有計算機環境では，通常，様々な要因によりソフトウェアのアップデートが限定的であり，state-of-the-art な最新のソフトウェアが利用できなかったり，システムがソフトウェアのバージョンを勝手に変えてしまうことによりユーザのアプリケーションが動作しなくなったりなど，ユーザのソフトウェア配備の生産性や再現性など阻害されることが問題となっていたが，このような問題は，ユーザ権限で動作する Linux コンテナ技術を用いることによりアプリケーションの性能低下することなく大幅に緩和できると考える．とりわけ，AI/ビッグデータ処理では，アプリケーションの実行に高い性能が求められる一方で，複雑な依存関係がある多数のソフトウェアコンポーネントをインストールする必要があるため，ユーザ権限で動作する Linux コンテナ技術は有望なアプローチである．

5. おわりに

本稿では，ユーザ権限で動作する Linux コンテナの一つである Singularity をスーパーコンピュータに類似した共有計算機環境である AI クラウド上へ適用し，AI/ビッグデータ処理を模したワークロードや分散深層学習のワークロードを実行することにより予備的な性能評価を行った．その結果，概ね，Singularity を用いた Linux 環境でもベアメタルな環境と遜色のない性能を示すことがわかった．特に，デファクトで利用されている Docker コンテナのイメージを利用することができ，かつ，カスタマイズしたコンテナイメージをユーザ権限のみで共有計算機上で実行できる点はユーザ側や運用管理者側にとってソフトウェア配備の生産性の向上や再現性の担保に大きく寄与すると考えられる．今後は，安定性の向上やセキュリティーイシューの改善など更なるエンジニアリングによるプロダクション環境での利用に向けた取り組みが求められる．

謝辞 この研究の一部は，NEDO 次世代人工知能・ロボット中核技術開発，及び，JSPS 科研費 26540050 の一環で実施した．

参考文献

[1] 小川宏高，　松岡聡，　佐藤仁，高野了成，滝澤真一朗，谷村勇輔，三浦信一，関口智嗣：AI 橋渡しクラウド— AI Bridging Cloud Infrastructure (ABCI) — の構想，情報処理学会研究報告 Vol.2017-HPC-160 No.28，pp. 1–7 (2017).
[2] 松岡聡，遠藤敏夫，　額田彰，三浦信一，野村哲弘，　佐藤仁，實本英之，Drozd, A.：HPC とビッグデータ・AI を融合するグリーン・クラウドスパコン TSUBAME3.0 の概要，情報処理学会研究報告 Vol.2017-HPC-160 No.29，pp. 1–6 (2017).
[3] Docker: Docker - Build, Ship, and Run Any App, Anywhere, https://www.docker.com/.
[4] Kurtzer, G. M., Sochat, V. and Bauer, M. W.: Singularity: Scientific containers for mobility of compute, *PLoS ONE*, Vol. 12, No. 5, pp. 1–20 (online), DOI: 10.1371/journal.pone.0177459 (2017).
[5] Canon, R. S. and Jacobsen, D.: Shifter : Containers for HPC, *Cray User Group 2016*, pp. 1–8 (2016).
[6] Benedicic, L., Cruz, F. A. and Schulthess, T. C.: Shifter : Fast and consistent HPC workflows using containers, *Cray User Group 2017*, pp. 1–11 (2017).
[7] Priedhorsky, R., Randles, T. C. and Randles, T.: Charliecloud: Unprivileged containers for user-defined software stacks in HPC Charliecloud: Unprivileged containers for user-defined software stacks in HPC, *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. p1–10 (online), available from ⟨http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-16-22370⟩ (2017).
[8] 産総研人工知能研究センター：産総研人工知能クラウド (AAIC), http://www.airc.aist.go.jp/en/info_details/computer-resources.html.
[9] NVIDIA: NVIDIA Docker, https://github.com/NVIDIA/nvidia-docker.
[10] NVIDIA: NVIDIA GPU Cloud, https://ngc.nvidia.com.
[11] Docker: Docker Hub, https://hub.docker.com/.
[12] Sochat, V.: Singularity Hub, https://singularity-hub.org/.
[13] NVIDIA: cuBLAS, https://developer.nvidia.com/cublas.
[14] Intel: Intel Math Kernel Library, https://software.intel.com/en-us/mkl.
[15] McCalpin, J. D.: STREAM: Sustainable Memory Bandwidth in High Performance Computers, https://www.cs.virginia.edu/stream/.
[16] Panda, D. K.: OSU Micro Benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/ (2001-2017).
[17] Axboe, J.: Flexible I/O Tester (FIO), git://git.kernel.dk/fio.git.
[18] Fei-Fei, L.: ImageNet, http://www.image-net.org/.
[19] He, K., Zhang, X., Ren, S. and Sun, J.: Deep Residual Learning for Image Recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778 (online), DOI: 10.1109/CVPR.2016.90 (2016).
[20] Akiba, T., Fukuda, K. and Suzuki, S.: ChainerMN: Scalable Distributed Deep Learning Framework, *arXiv.org e-Print archive*, pp. 1–6 (online), available from ⟨http://arxiv.org/abs/1710.11351⟩ (2017).
[21] 秋葉拓哉：ChainerMN による分散深層学習の性能について，https://www.cs.virginia.edu/stream/.