

Cygnus 上での Docker Rootless Mode の利用の検討

畑中 智之^{1,a)} 建部 修見²

概要: コンテナを使用することでプログラムとその実行に必要な環境をまとめることができ、環境の差異に起因する不具合の削減や再現性の向上を期待できる。またオーケストレータを利用することで、その範囲をノード間に拡張することができる。Docker 19.03 で導入された Rootless Mode を用いると、Docker のデーモンをユーザ権限で動作させることが可能となる。ユーザ権限で Docker を動作させオーケストレータを利用することで、共有計算機環境においてもその恩恵にあずかることが期待できる。そこで、筑波大学計算科学研究センターで運用されている Cygnus を利用して Docker Rootless Mode の利用に関する検討を行った。InfiniBand の利用に関するアプローチと、HPL の実行を通した MPI アプリケーションの実行に関するアプローチを検討し、それぞれ性能を評価した。そして、パフォーマンス面で遜色ない結果を得ながら、コンテナとしてある程度の独立性を確保できることを確認した。

1. はじめに

コンテナはプログラムとその実行に必要な環境をまとめたものである。開発環境と実行環境で同じ環境を用意でき環境をコードとして書き表せるため、環境の差異に起因する不具合の削減や再現性の向上が期待できる。

Docker[1] は産業界で用いられているコンテナソフトウェアで、デファクトスタンダードになっている。dockerd と呼ばれるデーモンが常駐しており、その実行には通常ルート権限が必要である。Docker Rootless Mode[2][3] は、Docker 19.03 で導入された dockerd をユーザ権限で動作させるモードである。ユーザ権限での動作によって、共有計算機環境での Docker の利用も考えられるようになってきた。ほかのコンテナソフトウェアとして、共有計算機環境においてコンテナを利用するためにユーザ権限で動作する Singularity[4]、Charliecloud[5] などが登場している。

コンテナのオーケストレータはコンテナの運用を円滑にする。Docker Swarm Mode[6] や Kubernetes[7] はオーケストレータの一例である。ノードをまたいだコンテナの管理やオーバーレイネットワークによるコンテナのホスト名での名前解決などの機能を有し、開発者の想定した環境を準備できるため有用である。

さて、HPC 分野においてもコンテナの利用は盛んである。その研究にはクラウドなどのルート権限を持つ環境を想定したものと、共有計算機環境などのユーザ権限し

か持てない環境を想定したものがそれぞれ存在する。前者にはルート権限で動作する Docker 上のコンテナにおける InfiniBand の利用 [8] やオーケストレーションの利用について [9]、後者には Docker をいかにセキュアに扱うか [10][11] や代替のコンテナソフトウェアを評価した研究がある [12][13]。しかしながら、共有計算機環境で Docker Rootless Mode を利用する研究は進んでいない。

そこで本研究では共有計算機環境における Docker の利用を目的に、Docker Rootless Mode を筑波大学計算科学研究センターで運用されているスーパーコンピュータ Cygnus[14] で実行し、その利用に必要なことを検討した。

本論文は 8 章で構成される。まず、2 章で関連研究を、3 章で Docker Rootless Mode の利用について述べる。次に、4 章で Cygnus についてと環境設定について述べる。5 章で InfiniBand の利用に関するアプローチについて、6 章でベンチマークソフトウェアである HPL[15] の実行を通した MPI アプリケーションの実行に関するアプローチについて述べ、7 章で得られた結果について検討する。最後に、8 章でまとめと課題を述べる。

2. 関連研究

Docker[1] はデファクトスタンダードなコンテナソフトウェアである。コンテナの起動や停止などのコンテナの管理を行う dockerd の実行には通常ルート権限が必要で、セキュリティ上の理由により共有計算機環境での利用が難しい。Docker 19.03 で導入された Rootless Mode は dockerd をユーザ権限で動作させることを可能とする [2][3]。ユーザ権限での動作によって共有計算機環境での利用を考えら

¹ 筑波大学大学院システム情報工学研究科

² 筑波大学計算科学研究センター

^{a)} hatanaka@hpcs.cs.tsukuba.ac.jp

れるようになってきたが、未だ研究段階である。

Docker に Rootless Mode が導入される以前から HPC 分野でコンテナを利用する研究がされている。[10][11] はいずれも Docker の機能を制限して共有計算機環境でもセキュアに扱うことを目的とした研究である。[10] は Docker のコンテナを Slurm のジョブとして実行することを可能にしている。[11] は Docker のラッパーを介することで MPI コンテナをユーザ権限で実行できるようにしている。しかしながら、いずれも dockerd はルート権限で動作しておりセキュリティ上の問題が残る。

Singularity[4] は Sylabs が提供しているコンテナソフトウェアで、ユーザ権限で動作するように設計されている。Docker 互換を謳っていて、Dockerfile からのイメージのビルドや、Docker コンテナの実行が可能である。Charliecloud[5] はユーザ権限で動作するコンテナソフトウェアで、Docker コンテナの実行が可能としている。[12] は Singularity で MPI を実行するアプローチと性能を評価している。[13] は Singularity や Charliecloud といった共有計算機環境で用いられるコンテナソフトウェアの性能を評価している。

[8] はルート権限で動作する Docker 上で動作するコンテナで InfiniBand を利用したときの性能を評価している。[9] はクラウド環境において HPC のワークロードについて評価した研究で、Docker Swarm Mode と Kubernetes を利用するアプローチと、同環境で Ethernet と InfiniBand を利用したときの性能を評価している。

3. Docker Rootless Mode の利用について

3.1 Docker Rootless Mode の概要

Docker Rootless Mode[2][3] は dockerd をユーザ権限で動作させるモードである。fake root を実現する RootlessKit[16] と、ユーザ権限によるネットワークの操作を実現する slirp4netns[17] ないし VPNKit[18] を用いて、ユーザ権限で動作しながらあたかもルート権限を持っているかのように振る舞うことで動作する。

RootlessKit は user_namespaces(7), newuidmap(1), newgidmap(1) を使い、ユーザの UID, GID をマッピングする。プロセス内の UID=0 をホストのユーザの UID に、プロセス内の UID>0 を subuid(5) によって設定された範囲内の UID にマッピングする。GID についても同様である。これによって、プロセス内ではあたかもルート権限を持っているかのように振る舞うことができ、Docker が持つ任意のユーザによる実行をユーザ権限でも可能とする。

slirp4netns, VPNKit はともにネットワークの操作をユーザ権限で扱えるようにする。ネットワークアダプタの作成や仮想ネットワークの構築ができ、Docker のネットワーク機能をユーザ権限でも扱えるようになる。

```
1 node1
2 node2
3 node3
4 node4
```

図 1 コンテナのホスト名を用いた hostfile の例

3.2 Docker Swarm Mode の概要

Docker Swarm Mode[6] はオーケストレータのひとつで、ノードをまたいだコンテナの管理や、オーバーレイネットワークによる通信を可能とする。コンテナをサービスとして展開することで、自ノードを含むいずれかのノードでコンテナを実行させることができる。また、ノードをまたいだコンテナ間の通信にはオーケストレータが用意したオーバーレイネットワークと DNS を用いることで、コンテナのホスト名を使用した名前解決と通信を行えるようになる。なお、Docker Rootless Mode において Docker Swarm Mode を利用できることは [19] で明らかになっている。

3.3 Docker の利点

Docker をはじめとするコンテナソフトウェアが提供するコンテナを用いることの利点で大きいのは、開発者が任意の環境を用意できる点にある。共有計算機環境ではルート権限が与えられていないことが多く、必要なソフトウェアがホスト環境にインストールされていない場合は管理者に依頼する必要がある。ソフトウェアのインストールがユーザ権限で完結できる場合でも、ホスト環境に合わせた設定が必要になることが多く単純ではない。一方コンテナでは、コンテナ内に必要なソフトウェアをインストールできるため、ホスト環境のソフトウェアの有無を考慮する必要がなくなる。また、開発環境で作成したコンテナを共有計算機環境で実行することで、環境の差異に起因する不具合の削減や再現性の向上が期待できる。

さらに Docker Swarm Mode を用いると、コンテナはそれぞれが持つホスト名でノード間での通信ができるため、ノード間に関する設定がホスト環境に依存しなくなる。コンテナのホスト名は開発者が任意に設定できるため、例えば、コンテナを 4 つ用意しそれぞれ node1, node2, node3, node4 と設定することで、図 1 に示すような hostfile を用いて MPI を実行できるようになる。これはジョブスケジューラによってノードが割り当てられたあとに、適宜設定を行う方法とは対照的である。

このように Docker の利点は多く、HPC においても有用である。

4. Cygnus

4.1 Cygnus について

Cygnus は筑波大学計算科学研究センターで運用されてい

るスーパーコンピュータである [14]. CPU, GPU, FPGA を有しており, それらを活用する次世代のスーパーコンピュータの実現を目指している.

今回は Cygnus のうち 2 ノードないし 4 ノードを用いて Docker Rootless Mode の検証を行った. 表 1 は Cygnus の構成のうち検証に関連する項目を抜粋したものである.

4.2 Cygnus の環境設定

Docker Rootless Mode においてネットワーク操作を行うには slirp4netns ないし VPNKit を使用できるが, 今回は slirp4netns を用いる. よって Docker Rootless Mode を利用するために必要な環境設定は Docker, RootlessKit, slirp4netns の導入と, subuid(5), subgid(5) の設定である. そのうち, Docker, RootlessKit, slirp4netns はユーザ権限で準備できるため, 表 2 のとおりに導入した. 一方, subuid(5), subgid(5) の設定はそれぞれ /etc/subuid, /etc/subgid を設定するためルート権限を必要とした.

5. InfiniBand に関する予備実験

HPC 分野において InfiniBand を利用することは重要で, Docker Rootless Mode におけるコンテナ利用についても例外ではない. しかしながら, コンテナ内から MPI を利用するには InfiniBand デバイスの扱いを考慮する必要性が生じる. そして, デバイスの扱いはルート権限を要する操作も存在するため, Docker Rootless Mode で動作するコンテナ内から InfiniBand を扱えるかは不明瞭である. そこで, Docker Rootless Mode において InfiniBand を利用できるかを次節以降の手順で検証した.

5.1 方針

Docker には各種デバイスを扱うために --device オプションが用意されている. Docker Rootless Mode においても InfiniBand を扱うために --device オプションを使用するのは自然である. また, --device オプションを使用すると /dev/infiniband がコンテナ内から見えるようになるため, これを扱う方針で検証する.

さて, /dev/infiniband をコンテナ内で直接扱うことになるため, コンテナ内にも OFED が必要になると考えられる. 一方, コンテナは通常カーネル部分の操作は行えないため, OFED のユーザ空間部分のみのインストールを行うこととする.

検証は 2 ノードでコンテナを立ち上げ qperf を実行することとする. それぞれのコンテナは Docker Swarm Mode によるオーバーレイネットワークで接続し, コンテナのホスト名による通信を試みる. 評価には RDMA のテストである rc_rdma_read_bw, rc_rdma_write_bw をコンテナとホストそれぞれでテストしてパフォーマンスを比較することで行う.

5.2 コンテナの設計

コンテナは CentOS をベースに, MLNX_OFED のユーザ空間部分と OpenSSH をインストールしたものを作成する. MLNX_OFED はコンテナ内でも InfiniBand を扱うため, OpenSSH は検証の過程で使用するためインストールしている. エントリーポイントは sshd をフォアグラウンドで動作する設定とする.

表 3 は, コンテナ環境と, 比較対象のホスト環境を表す. 実験の都合でコンテナ環境とホスト環境で異なる部分が存在するが, 結果に大きな影響はないと考えている.

5.3 検証

検証のため, インタラクティブに操作できるデバッグノードを使用する. また, あらかじめ開発環境で作成したコンテナを docker save コマンドで tar にまとめ, Cygnus に転送している.

まず, 各ノードで dockerd を Rootless Mode で立ち上げ, 作成したコンテナイメージを docker load コマンドでロードする. 次に, Docker Swarm Mode によるクラスタを構築し, オーバレイネットワークを作成する. そして, 各ノードでコンテナを docker run コマンドで実行する. このとき --device オプションで /dev/infiniband を, --network オプションでオーバーレイネットワークを, --hostname オプションでコンテナのホスト名を指定する. これでコンテナは sshd がフォアグラウンドで動作している状態になる.

コンテナが立ち上がったら docker exec コマンドで一方のコンテナに入り, qperf のサーバを起動する. 次に, コンテナ内で ssh してもう一方のコンテナに入り, qperf のクライアントを起動しパフォーマンステストをする. このとき InfiniBand の本数は指定せずに 1 本のみ使用した. またホスト名にコンテナのホスト名を指定することで正常にテストを行えた.

qperf によるテストを終了したあと, 後片付けを行い終了する.

5.4 結果

qperf によって得られた結果を表 4 に示す. コンテナ環境とホスト環境ともに 95[Gb/sec] から 96[Gb/sec] の性能が得られており, 遜色ない結果となった. また双方の結果は, 検証において InfiniBand HDR100 を 1 本のみ使用していることを考慮すると妥当といえる.

検証からコンテナのホスト名による名前解決と InfiniBand による通信を確認できた. このことから, OFED を導入したコンテナを用意し, 実行時に --device オプションを使用して /dev/infiniband を追加することで InfiniBand を利用できるといえる.

表 1 Cygnus のノードの構成 (検証に関連する項目を抜粋)

OS	CentOS 7.7.1908
CPU	Intel(R) Xeon Gold 6126 Processor (12C/2.6GHz) ×2
Memory	192GiB (16GiB DDR4-2666 ECC RDIMM ×12)
Network	InfiniBand HDR100 ×4

表 2 Docker と Docker Rootless Mode に用いるソフトウェア

Docker	master-dockerproject-2020-03-31, build 6e98ebc8
RootlessKit	0.7.0
slirp4netns	0.4.2

表 3 InfiniBand の検証に用いるコンテナとホストの環境

	コンテナ	ホスト
OS	CentOS 8.1.1911	CentOS 7.7.1908
MLNX_OFED	5.0-1.0.0.0	4.7-3.2.9.0

表 4 qperf の実行結果

	コンテナ	ホスト
rc_rdma_read_bw	95.5 [Gb/sec]	95.5 [Gb/sec]
rc_rdma_write_bw	96.0 [Gb/sec]	95.8 [Gb/sec]

表 5 HPL の主要なパラメータ

	2 ノード	4 ノード
Ns	98,304	147,456
NBs	384	384
Ps	6	8
Qs	8	12

表 6 HPL の検証に用いるコンテナとホストの環境

	コンテナ	ホスト
OS	CentOS 8.1.1911	CentOS 7.7.1908
MLNX_OFED	5.0-1.0.0.0	4.7-3.2.9.0
OpenMPI	4.0.3	4.0.3
GCC	8.3.1	8.3.1
Intel(R) MKL	2020.0.166	2019.5.281
HPL	2.3	2.3

6. HPL による評価

HPL は分散メモリコンピュータ用の Linpack ベンチマークである [15]. Docker Rootless Mode における MPI アプリケーションの実行に関するアプローチの検証と、コンテナでの実行によるパフォーマンスへの影響の評価を目的に、次節以降の手順で HPL による評価を行った。

6.1 方針

Docker Swarm Mode によるクラスタを用意し、InfiniBand を利用して HPL の性能を評価する。InfiniBand の利用は 5 章で検証した方法に従う。HPL はオーバーレイネットワーク上に構築したコンテナ上で実行する。このとき、MPI の実行時に指定するホスト名はコンテナのホスト名を用いる。また、コンテナのひとつで mpirun を実行し、MPI 経由で HPL を実行する。

コンテナは、コンテナ内から InfiniBand と MPI を利用するために、コンテナ内に OFED のユーザ空間部分を導入し、UCX に対応した MPI をビルド、インストールしたものを作成する。MPI の実行時に利用できる InfiniBand の本数はホスト環境に依存するためパラメータとして扱うこととする。

検証は 2 ノードと 4 ノードで行う。HPL の実行は表 5 に示すパラメータを用いて各 3 回試行し、最良値を取ることとする。

6.2 コンテナの設計

コンテナは CentOS をベースに、MLNX_OFED のユーザ空間部分、OpenSSH、OpenMPI、Intel(R) Math Kernel

Library をインストールし、HPL をビルドしたものを作成する。MLNX_OFED はコンテナ内でも InfiniBand を扱うため、OpenSSH と OpenMPI は MPI を利用するため、Intel(R) Math Kernel Library は HPL 内の演算に利用するためにインストールしている。なお、OpenMPI は UCX を用いるためソースコードからビルド、インストールする。エントリポイントは sshd をフォアグラウンドで動作する設定とする。

表 6 は、コンテナ環境と、比較対象のホスト環境を表す。実験の都合でコンテナ環境とホスト環境で異なる部分が存在するが、結果に大きな影響はないと考えている。

6.3 検証

あらかじめ開発環境で作成したコンテナを docker save コマンドで tar にまとめ、Cygnus に転送している。

まず、各ノードで dockerd を Rootless Mode で立ち上げ、作成したコンテナイメージを docker load コマンドでロードする。次に、Docker Swarm Mode によるクラスタを構築し、オーバーレイネットワークを作成する。そして、各ノードでコンテナを docker run コマンドで実行する。このとき --device オプションで /dev/infiniband を、--network オプションでオーバーレイネットワークを、--hostname オプションでコンテナのホスト名を指定する。また、HPL のパラメータを示す HPL.dat と MPI に用いる hostfile はノード

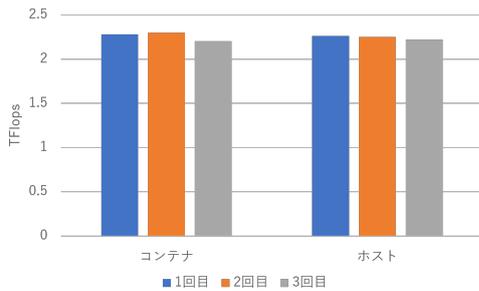


図 2 HPL を 2 ノードで実行した結果

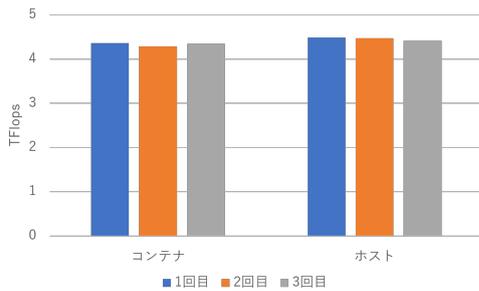


図 3 HPL を 4 ノードで実行した結果

数に応じて変更できるように--volume オプションで指定する。これでコンテナは sshd がフォアグラウンドで動作している状態になる。

コンテナが立ち上がったら docker exec コマンドを用いて mpirun に HPL のバイナリを渡して実行し、標準出力から HPL の結果を得る。このとき、利用する InfiniBand の本数を指定するために mpirun の環境変数として UCX_MAX_RNDV_RAILS を設定した。

HPL の実行が終了したあと、後片付けを行い終了する。

6.4 結果

2 ノードでの実験結果を図 2 に示す。コンテナ環境での最良値は 2.2985[TFlops]、ホスト環境での最良値は 2.2608[TFlops] であった。ホスト環境と比較したコンテナ環境の性能比は約 102[%] と、遜色ない結果が得られた。

4 ノードでの実験結果を図 3 に示す。コンテナ環境での最良値は 4.3635[TFlops]、ホスト環境での最良値は 4.4883[TFlops] であった。ホスト環境と比較したコンテナ環境の性能比は約 97[%] と、遜色ない結果が得られた。

検証では MPI アプリケーションの実行に用いられる hostfile にコンテナのホスト名を用いた。この hostfile で MPI アプリケーションを実行できたことから、ホスト環境に依存せずにアプリケーションを実行できたといえる。

7. 検討

5 章の結果より、Docker Rootless Mode において InfiniBand を利用できること、遜色ない結果が得られパフォーマンスへの影響を考慮する必要がないこと、Docker Swarm

Mode によるオーバレイネットワークを構築することでノード間でもコンテナ名で利用できることが判明した。6 章の結果より、Docker Swarm Mode によるオーバレイネットワークを構築し InfiniBand を利用して MPI アプリケーションを実行できること、パフォーマンスに大きく影響を与えないことが判明した。

Docker の--device オプションと Docker Swarm Mode を組み合わせる方法は、Docker Rootless Mode において InfiniBand を利用するアプローチとして利用すると判断できる。MPI アプリケーションについても、利用するノードでコンテナを立ち上げ、コンテナのひとつで docker exec コマンド経由で mpirun を実行するアプローチが有効であった。

さて、Docker Rootless Mode と Singularity を比較すると、Software Defined Network (SDN) を構築できるか否かに違いがある。これは、例えば MPI を実行するときにコンテナ上で MPI を動かすか、MPI 上でコンテナを動かすかの違いとなる。単純に MPI を実行する場合は SDN を構築しない Singularity のほうが簡潔で、ノードにまたがり複雑なことをする場合は SDN を構築できる Docker Rootless Mode のほうが有利と考えられる。各種デバイスの扱いは、コンテナ内にドライバを導入しデバイスファイルを扱う必要がある点はどちらも同じである。Docker は各種デバイスを扱うために--device オプションを使用する必要があり、Singularity はデフォルトで/dev がバインドされるためオプションの指定が不要である点の違いとなるが、大きな差にはならないと考えられる。

Docker Rootless Mode とユーザ権限しか持たない環境においても Docker コンテナを実行できる点は、開発者が望む環境で実験でき、さらに再現性にも優れることから重要といえる。さらに Docker Swarm Mode を利用することで、複数のノード間においても適用できるため応用できる範囲は広がる。一方、InfiniBand の利用のためコンテナ内に OFED のインストールしたことや、ホスト環境に依存するパラメタが残ることはコンテナの独立性の観点で望ましくないため、コンテナの独立性を保ちつつホスト持つ特性を活かすことが次の課題となってくる。

8. まとめと課題

本研究では筑波大学計算科学研究センターで運用されている Cygnus 上で Docker Rootless Mode を実行し、その利用に必要なことを検討した。その結果、Cygnus 上で Docker Rootless Mode を使用でき、コンテナから InfiniBand を扱えることを確認できた。また qperf と HPL を実行し、InfiniBand と計算性能ともに遜色ない結果が得られた。Docker Swarm Mode の利用によってホスト環境のホスト名に依存せず、コンテナのホスト名を用いることで実行できることも確認した。

一方、OFED のインストールを行ったことやホスト環境に依存するパラメタが残ったことで、コンテナの独立性に課題が残る結果となった。また、個々のケースに応じた効率的なコンテナの作成と実行についても今後の課題である。

謝辞 本研究の一部は、筑波大学計算科学研究センターの学際共同利用プログラム (Cygnum), 国立研究開発法人新エネルギー・産業技術総合開発機構 (NEDO) および富士通研究所との共同研究の助成を受けたものです。

参考文献

- [1] Docker: Empowering App Development for Developers | Docker, Docker Inc. (online), available from <https://www.docker.com/> (accessed 2020-05-28).
- [2] Docker: Run the Docker daemon as a non-root user (Rootless mode) | Docker Documentation, Docker Inc. (online), available from <https://docs.docker.com/engine/security/rootless/> (accessed 2020-06-03).
- [3] Docker: Docker Engine release notes | Docker Documentation, Docker Inc. (online), available from <https://docs.docker.com/engine/release-notes/#19030> (accessed 2020-06-03).
- [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] Priedhorsky, R. and Randles, T.: Charliecloud: Unprivileged Containers for User-defined Software Stacks in HPC, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, ACM, pp. 36:1–36:10 (online), DOI: 10.1145/3126908.3126925 (2017).
- [6] Docker: Docker Swarm overview | Docker Documentation, Docker Inc. (online), available from <https://docs.docker.com/swarm/overview/> (accessed 2020-06-03).
- [7] Kubernetes: Production-Grade Container Orchestration - Kubernetes, The Kubernetes Authors (online), available from <https://kubernetes.io/> (accessed 2020-06-03).
- [8] Chung, M. T., Le, A., Quang-Hung, N., Nguyen, D. and Thoai, N.: Provision of Docker and InfiniBand in High Performance Computing, *2016 International Conference on Advanced Computing and Applications (ACOMP)*, pp. 127–134 (2016).
- [9] Beltre, A. M., Saha, P., Govindaraju, M., Younge, A. and Grant, R. E.: Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms, *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, pp. 11–20 (2019).
- [10] Azab, A.: Enabling Docker Containers for High-Performance and Many-Task Computing, *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 279–285 (2017).
- [11] de Bayser, M. and Cerqueira, R.: Integrating MPI with Docker for HPC, *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 259–265 (2017).
- [12] Sande Veiga, V., Simon, M., Azab, A., Fernandez, C., Muscianisi, G., Fiameni, G. and Marocchi, S.: Evaluation and Benchmarking of Singularity MPI containers on EU Research e-Infrastructure, *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, pp. 1–10 (2019).
- [13] Torrez, A., Randles, T. and Priedhorsky, R.: HPC Container Runtimes have Minimal or No Performance Impact, *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, pp. 37–42 (2019).
- [14] 筑波大学計算科学研究センター: スーパーコンピュータ 筑波大学計算科学研究センター Center for Computational Sciences, (オンライン), 入手先 <https://www.ccs.tsukuba.ac.jp/supercomputer/#Cygnum> (参照 2020-06-01).
- [15] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary: HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, Innovative Computing Laboratory (online), available from <https://www.netlib.org/benchmark/hpl/> (accessed 2020-05-27).
- [16] rootlesskit: GitHub - rootless-containers/rootlesskit: kind of Linux-native "fake root" utility, made for mainly running Docker and Kubernetes as an unprivileged user, rootless-containers (online), available from <https://github.com/rootless-containers/rootlesskit> (accessed 2020-06-08).
- [17] slirp4netns: GitHub - rootless-containers/slirp4netns: User-mode networking for unprivileged network namespaces, rootless-containers (online), available from <https://github.com/rootless-containers/slirp4netns> (accessed 2020-06-08).
- [18] vpnkit: GitHub - moby/vpnkit: A toolkit for embedding VPN capabilities in your application, moby (online), available from <https://github.com/moby/vpnkit> (accessed 2020-06-08).
- [19] 畑中智之, 建部修見: ユーザ権限における Docker オーケストレーションの構築と撤去, 技術報告 3, 筑波大学大学院システム情報工学研究科, 筑波大学計算科学研究センター (2019).