Vol.2020-HPC-174 No.1
2020/5/13

大規模計算クラスタにおけるユーザ利便性向上

中田 秀基[1,a)]　滝澤 真一朗[1]　小川 宏高[1]

概要：大規模な計算クラスタの多くでは、計算ノードがバッチスケジューリングシステムで管理されているため、事前にアドレスを確定する事ができない、計算ノードに直接外部からネットワーク接続できない、計算ノードから外部ネットワークへの接続が制限される、といった制約がある。このため、ネットワーク接続を前提とする Jupyter Notebook などの近代的なプログラム実行環境の利用が難しい。われわれは、これらの制約を回避して、ユーザの手元の計算機から大規模クラスタの計算ノード上の Jupyter Notebook を簡便に利用する方法を確立した。ユーザの計算機環境に小規模なプログラムをインストールすることが許されるケースと、ユーザ計算機には全く依存しない代わりにクラウド上のサービスが利用できるケースの、2 つのシナリオを対象として検討を行い、いずれの場合においても容易に利用できる環境を実現した。

キーワード：クラスタ, Jupyter notebook

Improvement of User-friendliness of Large-scale Computation Clusters

Abstract: Computer nodes of large-scale computation clusters are typically allocated dynamically by batch scheduling systems and do not accept a direct connection from outside of the clusters. Furthermore, direct connections from the computation nodes to the outside of the clusters are often prohibited. This restriction prevents users from using modern user-friendly programming environments, such as Jupyter notebook, that depends on network connection between the computation nodes and the users terminal. We report our efforts on solving this problem using some tricks with ssh tunneling in two scenarios; 1) with small installation on the users' terminal, 2) with a service on a cloud and without any program installation on the users' terminal,

Keywords: Cluster, Jupyter notebook

1. はじめに

データ解析の分野では、実行した結果をプログラマが見た上で、インタラクティブにプログラムを発展させていく手法が重要である。そのようなプログラム開発・実行環境として Jupyter Notebook が広く利用されている。Jupyter Notebook は、クライアントの端末環境として Web ブラウザを用い、計算環境はサーバとして分離されている。したがって、サーバ環境としてリモートの計算機を利用することも比較的容易に実現できる。

しかし、ABCI[1] などの大規模な計算クラスタの多くでは、計算ノードがバッチスケジューリングシステムで管理されているため事前にアドレスを確定する事ができない、計算ノードに直接外部からネットワーク接続できない、計算ノードから外部ネットワークへの接続が制限される、といった制約がある。

このため、ネットワーク接続を前提とする Jupyter Notebook などの利用は難しい。これは Jupyter Notebook に固有の問題ではなく、Visual Studio Code の Remote Extension などでも同様である。

本稿の目的は、大規模計算クラスタに固有の制約を回避して、Jupyter Notebook を簡便に利用する手法を確立することである。ここでは、ユーザ端末の自由度に応じて 2 つのケースを考えた。1 つは、ユーザ端末に小規模なプログラムをインストールすることが許されるケースである。もう 1 つは、ユーザ計算機には全く依存しない代わりにクラウド上のサービスが利用できるケースである。

本稿の構成は以下の通りである。2 節で、前提となる環境と Jupyter Notebook および JupyterHub について説明

[1] National Institute of Advanced Science and Technology AIST, Tsukuba, Ibaraki 305–8560, Japan
[a)] hide-nakada@aist.go.jp

する。3 節で、提案手法を説明する。4 節で議論を行う。5 節でまとめと今後の課題について述べる。

2. 背景

2.1 大規模クラスタ計算機環境

大規模なクラスタ計算機では利用効率の向上とセキュリティ担保のため、通常の計算機環境にはない制約があることが一般的である。

- バッチキューイングシステム
  大規模クラアスタ計算機の計算機ノードはバッチキューイングシステムで管理されており、ユーザのリクエストに応じて、空いているノードが自動的に割り当てられる。したがってユーザがノードを選定することはできず、利用する計算機のアドレスを事前に指定することはできない。
- 外部からのネットワーク接続
  一般に計算ノードはグローバルな IP アドレスを持たない。グローバル IP アドレスをもつ場合でも、セキュリティ上の問題で外部からのルーティングを行わないのが一般的である。
- 内部からのネットワーク接続
  さらに、計算ノードから外部ノードへの接続も制限することが一般的である。これは何らかの方法で計算ノードが悪意のあるユーザに利用される事があったとしても、外部に対する攻撃の踏み台になることを防ぐためである。

2.2 Jupyter notebook

Jupyter Notebook[2] は、Web ベースのインタラクティブなプログラム環境で、主にデータ解析の分野で広く用いられている [3]。もともと Python をターゲットとして開発されていたが、Web ブラウザとの通信部分とプログラム実行環境エンジン (カーネルと呼ぶ) を分離することで、さまざまな言語をサポートできるようになった。現在では R, Julia, Ruby を始め非常に多数の言語がサポートされている [4]。

図 1 に実行の様子を示す。ノートブックは複数のセルから構成され、セル単位で評価を行う。グローバル変数はセル間で共有される。描画ライブラリもサポートされており、計算結果を可視化した上で次の操作を決定することができる。ユーザからの GUI 入力を行うことも可能で、Python ではウィジェットを使うことで GUI ベースのインタラクティブなアプリケーションを構成することも可能である。

また、JupyterLab と呼ばれる強化版も存在し、プラグインなどが活用できるようになっている。

2.3 JupyterHub

JupyterHub[5] は、Jupyter Notebook を複数のユーザに対して提供するためのフロントエンドアプリケーションである。PAM や OAuth などの認証機構を用いてユーザの認証を行い、そのユーザのアカウントで Jupyter Notebook をローカルで起動する。また Kubernetes や Yarn を用いてリモートでノートブックを起動することもできる。

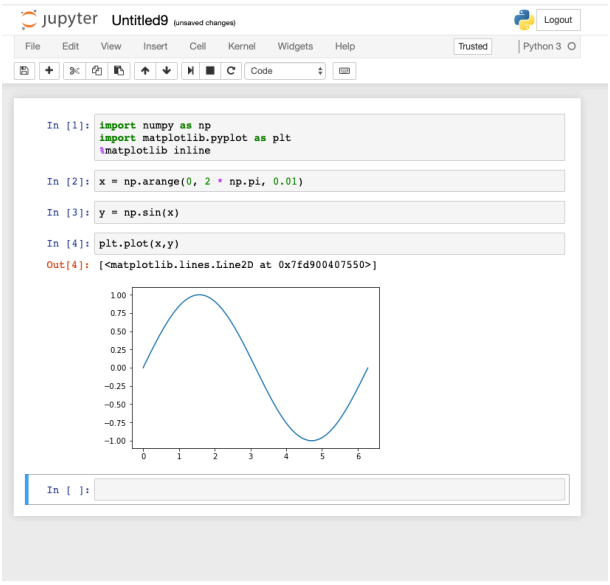

図 1 Jupyter Notebook

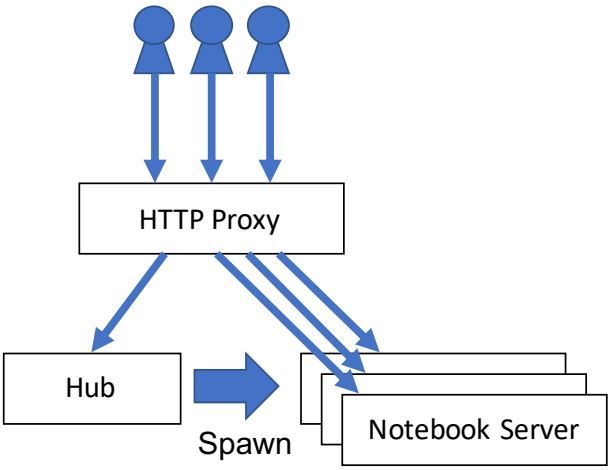

図 2 JupyterHub の概要

図 2 に JupyterHub の基本的な動作を示す。JupyterHub はハブ本体のプロセスと HTTP プロキシプロキシから構成される。プロキシは、起動時にはデフォルトですべての入力をハブに転送するように設定されている。ユーザがアクセスすると、ハブは適切な認証を行い、そのユーザ専用の Notebook のプロセスを起動する。さらにプロキシにリダイレクトの設定を行い、以後そのユーザからのアクセスを Notebook プロセスにリダイレクトするようにする。

Notebook プロセスを起動・監視するコードは、Spawner と呼ばれるクラスとして抽象化されており、これを差し替えることでプロセスの起動場所や起動方法を変更することができる。独自の Spawner クラスを実装するには、

start、stop、poll の 3 つのメソッドを実装する必要がある。start メソッドは Notebook プロセスを起動し、その Notebook のアドレスとポートを返す。stop はノートブックを停止する。poll は、プロセスの動作状況を確認しデータベースを更新する。

この際 Notebook プロセスからは 2 つの接続が必要となる。1 つは通常の Jupyter Notebook の場合と同様にノートブックそのものを提供する HTTPS 接続、もう 1 つは、Notebook プロセスの状態を管理するための接続である。後者は Notebook プロセスから Hub へのコールバック接続として実現される。

2.4 SSH ControlMaster

SSH はコネクションを多重化し、リモートサイトのポートをクライアント側にリダイレクトしたり、クライアント側のポートをサーバノードから到達できる任意のホスト/ポートにリダイレクトすることができる。この機能は Port Forwarding と呼ばれる。

Port Forwarding は一般にはセッション開始時に指定するが、ControlMaster と呼ばれる機能 [6] を用いると、既存のセッションに対して動的に Port Forwarding を追加、削除することができる。

ControlMaster は Unix ドメインソケットを指定されたパス (ControlPath) に作成する。このソケットを指定して下のようにすると、フォワードを動的に追加する事ができる。

```
$ ssh -O forward -L 8080:localhost:80\
      -S SOCKET SERVER
```

同様に、下のように cancel を指定することで、フォワードを停止することができる。

```
$ ssh -O cancel -L 8080:localhost:80\
      -S SOCKET SERVER
```

3. 手法

ここでは、ユーザ端末の自由度に応じて 2 つのケースを考えた。1 つは、ユーザ端末に小規模なプログラムをインストールすることが許されるケースである。もう 1 つは、ユーザ計算機には全く依存しない代わりにクラウド上のサービスが利用できるケースである。それぞれのケースに対して 2 つの手法を提案する。

3.1 通常の手法

提案手法を示す前に、通常の手法でバッチキューイングシステムを用いるクラスタ上で Jupyter Notebook を用いる方法をまとめる (図 3)。

(1) ユーザはクライアントからログインノードに ssh でログインする。

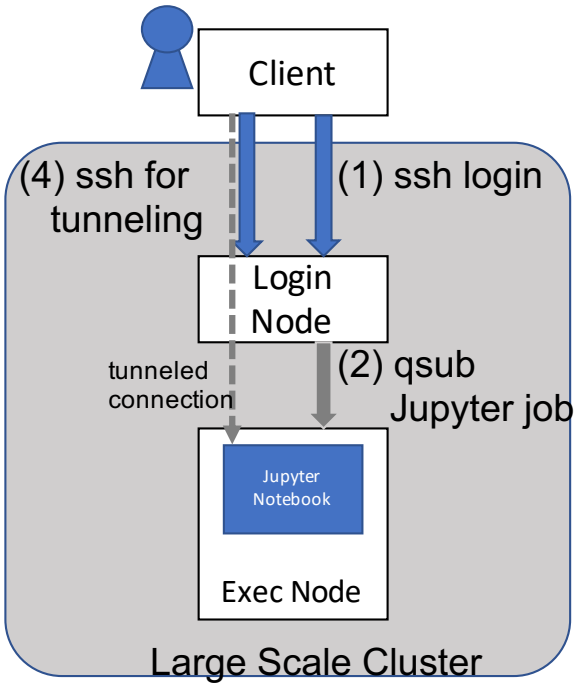

図 3 従来の手動による手法

(2) qsub もしくは qrsh を用いて、Jupyter Notebook を実行ノード上で起動する。

(3) 起動が確認できたら、実行ノードの IP アドレスと Jupyter Notebook が起動したポートを取得する。

(4) クライアントノードから別の SSH セッションを張ってトンネリングを行い、実行ノードに接続する。

(5) トンネリングを指定したローカルポートをブラウザで指定して開く。

手順としては複雑なものではないが、IP アドレスを目視で確認し、それに応じてトンネリングのためのコマンドを修正する必要がある点が、特に煩雑である。

3.2 自動ポートフォワード

提案手法の 1 つ目は、自動ポートフォワードである。これは小さなスクリプトをユーザの PC にインストールし、前項で述べた手続きを自動化するものである。

このスクリプトはサーバ上のスクリプトを ssh を経由して起動する。サーバ側のスクリプトは qrsh を用いて資源を取得し、その資源上で Jupyter Notebook を起動する。このスクリプトは、資源の IP アドレスを取得し、Jupyter Notebook が作成する JSON ファイルを監視することでポート番号を取得し、これらの情報を標準出力に書き出す[*1]。クライアント側のスクリプトは、書き出された情報を用いて新たに SSH のセッションを起動して Jupyter

[*1] 一つのノードが複数のジョブで共有される可能性があるため、デフォルトポート以外のポートで Jupyter Notebook が起動することがある。

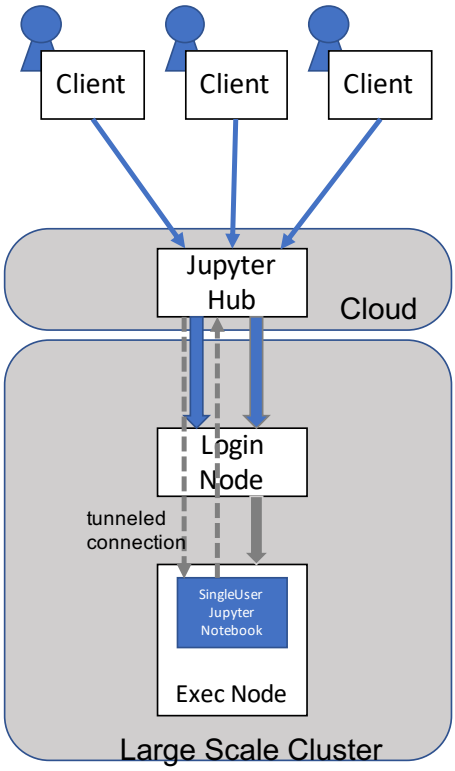

図 4　JupyterHub を用いた手法の概要

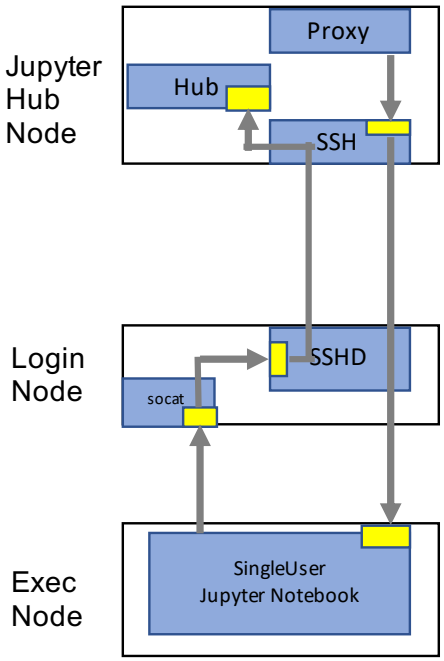

図 5　JupyterHub を用いた手法の詳細

Notebook のポートフォワーディングを設定し、そのローカルポートを指定してブラウザ上にページを開く。

ここで 2.4 で述べた ControlMaster を使わないのは Windows での動作を実現するためである。ControlMaster は Unix ドメインソケットに依存するため、Windows では利用できない。

3.3　JupyterHub を利用する方法

前節で述べた方法は、簡便ではあるがユーザがローカル環境とサーバ側にそれぞれプログラムをインストールし、更に起動時にはシェルスクリプトを起動する必要がある。これは従来の大規模計算機ユーザにとっては当たり前のことだが、新規ユーザを取り込むためには障壁となる。特に機械学習や人工知能の研究者には、GUI 環境以外での作業を嫌うものも多く、可能な限り参入障壁を下げる必要がある。

これを実現するために、われわれは Jupyter Hub を用いる機構を提案する。ターゲットとなる大規模計算機外部に Jupyter Hub を設置し (AWS などのパブリッククラウドを想定)、この Jupyter Hub を複数のエンドユーザで共有する。

エンドユーザは OAuth を用いて認証を行い、Jupyter Hub にログインする。Jupyter Hub は大規模計算機上に Jupyter Notebook を起動し、トンネリング設定することで Client からの接続をリダイレクトする。この様子を図 4 に示す。

自動ポートフォワード法との相違点は、実行ノード上で起動するのが、Jupyter Notebook そのものではなく、Jupyter Notebook をラップした Single user notebook server であることだ。JupyterHub は Single user notebook server と通信して管理を行う。この管理のコネクションは Single user notebook server の側から JupyterHub へ接続される。つまり、JupyterHub と実行ノードの間に、双方向のポートフォワードが必要になる。

順方向のポートフォワードは前節で述べた自動ポートフォワードと同様の方法で実現できる。逆方向のポートフォワードはログインノードでポートをオープンし、これをハブノードに対してフォワードしてやれば良い。

このようなコールバック接続を実現するには、SSHD がリモートフォワーディングのポートを任意のアドレスに対してバインドすることを許すように設定されていなければならない。これは SSHD config の GatewayPorts で設定されるが、デフォルトでは許可されていないことが多い。これは潜在的なセキュリティリスクとなる可能性があるからだ。

このような場合には socat[7] を用いる。socat は SOcket CAT を意味し、汎用のソケット間フォワード機構として機能する。

図 5 に提案手法のコネクションの図を示す。図中の黄色い四角は受信ポートを表す。ノードの境界に置かれている受信ポートは外部からのコネクションを受け付けるもの、境界に接していない受信ポートは内部からの接続しか受け付けないものである。複数のプロセスを経由して接続がフォワードされている事がわかる。

3.4 実装

われわれは提案システムを産総研の保有する大規模計算機である ABCI をターゲットとして実装した。自動ポートフォワード手法に関しては Windows と mac OS X と Linux で動作を確認した。Windows 版は PowerShell で、それ以外は bash で実装した。

JupyterHub を設置する手法では OpenSSH の ControlMaster を用いて動的にポートフォワードの設定を行った。JupyterHub を設置するパブリッククラウドとしては AWS を用いた。OAuth プロバイダとしては GitHub を用いた。これは JupyterHub がデフォルトでサポートしていたからである。

サーバ側の Jupyter Notebook 環境としては Singularity コンテナ [8] を利用した。Singularity は Docker と互換性を持つコンテナ環境で、コンテナ内の root アクセスを許さないなど、大規模計算機環境に適した特徴を持つ。コンテナを用いたのは標準的なライブラリを実装した整合した環境の提供を容易に実現するためである。

ABCI ではセキュリティ上の観点から GatewayPorts を許可していない。このため前節でのべたように socat を用いた実装をおこなった。

4. 議論

4.1 秘密鍵の管理

JupyterHub を用いる手法では、JupyterHub のノードから大規模計算機にパスワードなしで ssh できるように設定する必要がある。つまりパスフレーズを設定していない秘密鍵を JupyterHub のノード上に配備しておく必要がある。これは一般にセキュリティリスクとなりうる。

一般には、SSH でクライアント鍵ごとに実行できるコマンドを限定することができ、これによってある程度セキュリティリスクを低減できる。しかし ABCI では独自の SSH 鍵管理機構を利用しているため、この機構を利用することができない。

4.2 ユーザの管理

JupyterHub を用いる手法は、大規模計算機上のひとつのユーザアカウントを、複数のエンドユーザが共有することを前提としている。これが許されるかどうかは大規模計算機の運用ポリシに依存する。

ABCI ではエンドユーザの管理を厳密に行うことを前提にこのような運用を許容している。例えば SONY の運用する Neural Network Console ではバックエンドとして ABCI を利用することができる [9] が、この場合も ABCI 上のユーザアカウントは単一となっている。

4.3 接続切れの検出

接続が切れた場合には自動的に Notebook ジョブをシャットダウンする必要がある。SSHD config に ClientAlive を設定すると、クライアントとの接続を監視し、接続が切れると SSH で起動されたプロセスに SIGHUP がシグナルされる。

ABCI で利用しているバッチキューイングシステムの GridEngine の qrsh は SIGHUP を無視するので、これだけでは Notebook job を停止することができない。プロセスを簡単なラッパでくるみ、SIGHUP を SIGKILL に変換すれば、接続断時に自動的に Notebook ジョブを停止することができる。

4.4 Google Colab

Google Colab[10] は Google 社が提供する機械学習を指向した Jupyter Notebook 環境である。NVIDIA 社の GPU が提供されるだけでなく、TPU を選択する事もできる。また、Github などからコードやノートブックをクローンすることも容易にできる。さらに単一のノートブックを複数のユーザで共有し、同時に作業することも可能となっている。

GPU を持つランタイムへの切り替えはメニューから容易に行うことができるが、切り替え時にはノートブックの状態は失われる。このため、たとえば比較的重い前処理を GPU を持たないランタイムで行っておき、その状態をメモリ上に維持したまま GPU を持つランタイムに切り替えて学習を行う、というような運用はできない。

一方提案システムでは、GPU を持つランタイムと持たないランタイムを切り替えるには、ジョブの再投入とノートブックの再起動が必要になる。ノートブックそのものは同じものが立ち上がるが、もちろんノートブックの状態は維持されない。Google Colab と比較すると、実際に起こることは大差ないのだが、ユーザ経験としては大きく劣ると言える。

5. おわりに

本稿では大規模なクラスタ型計算機のユーザ利便性を向上するための 2 つの手法を提案し、試作を行った。1 つ目はローカル環境にスクリプトをインストールすることのできるユーザを対象とした、SSH のポートフォワードを自動化する手法である。もう一つは、他者がセットアップした環境を利用するもので、別のクラウド上に構築された JupyterHub を用いる手法である。この手法ではユーザの手元には Web ブラウザのみがあれば良い。これらの手法をもちいることでユーザの利便性は大きく向上させる事ができると考える。

今後の課題としては、提案したしたシステムを ABCI 上で実サービスとして展開することが挙げられる。また、Google Colab と同様に、ノートをクローズすることなくランタイムを切り替える機構を組み込むことも今後の課題である。

謝辞 実装をお手伝いいただいた西岡真吾様に感謝いたします。

この成果の一部は、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）の委託業務の結果得られたものです。

参考文献

[1] : ABCI AI Bridge Infrastructure: https://abci.ai/. Accessed: 2019-02-01.
[2] : Jupyter Project, https://jupyter.org/. Accessed: 2020-03-17.
[3] Randles, B. M., Pasquetto, I. V., Golshan, M. S. and Borgman, C. L.: Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study, *Proceedings of the 17th ACM/IEEE Joint Conference on Digital Libraries*, IEEE Press, p. 338–339 (2017).
[4] : Jupyter kernels, https://github.com/jupyter/jupyter/wiki/Jupyter-kernels. Accessed: 2020-03-17.
[5] : JupyterHub, https://jupyter.org/hub. Accessed: 2020-03-17.
[6] : OpenSSH/Cookbook/Multiplexing, https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Multiplexing. Accessed: 2020-03-17.
[7] : socat - Multipurpose relay (SOcket CAT), https://linux.die.net/man/1/socat. Accessed: 2020-03-17.
[8] : Singularity, https://sylabs.io/singularity/. Accessed: 2020-03-17.
[9] : ABCI x Neural Network Console, https://dl.sony.com/ja/cloud/abci/. Accessed: 2020-03-17.
[10] : Google Colaboratory, https://colab.research.google.com/. Accessed: 2020-03-17.