AOBA-S

Frovedis 利用手順書

日本電気株式会社

2024 年 5 月 16 日

第 3 版

＜＜　改版履歴　＞＞

版数	改版日	改版理由	改版者
初版	2022.9.20	AOBA-A 初版として作成	NEC
第 2 版	2024.4.8	AOBA-S 向けに改版	NEC
第 3 版	2024.5.16	一部で文言について修正および ログインシェルによるエラー時の対応の追記	NEC

目次

はじめに

本書は、東北大学サイバーサイエンスセンター AOBA-S(以下 AOBA-S)において、データ分析向けのフレームワークである Frovedis を利用するための環境設定、およびチュートリアル(サンプル実行)の手順を記したものです。

1 Frovedis

1.1 Frovideis 概要

Frovedis はデータ分析向けのフレームワークです。SX-Aurora TSUBASA の VE を用い、MPI を用いた並列実行を行うことで、高速なデータ分析を可能にします（x86 での実行も可能です）。
Python から利用するインタフェースと、C++から利用するインタフェースがあります。Python からは、機械学習とデータフレーム処理について、scikit-learn および Pandas と互換性のあるインタフェースで利用可能です。C++からは機械学習、データフレーム処理を実現する API および、Apache Spark に類似した低レベルの API を提供しています。
Frovedis はオープンソースソフトウェアであり、https://github.com/frovedis/frovedis で公開されています。

参考資料：https://jpn.nec.com/hpc/sxauroratsubasa/column/009.html

1.2 AOBA-S での Frovedis 環境

Frovedis はオープンソフトウェアであり、コード・マニュアル等が公開されていますが、AOBA-S では、利用者の個人環境で容易に利用することができるように、Singularity のコンテナイメージとして提供されています。
そのため、公開されている環境設定および操作方法とは違いがありますので、本書の手順を参照して利用してください。
また、以下の点に留意して利用してください。

① python インタフェースを利用する場合には、1 ノード(1VH＋8VE, 最大 128 並列)までの利用が可能です。
② C++API を利用する場合には、複数のノードを利用した並列処理を行うことが可能です。

2　Frovedis コンテナの利用方法

2.1　AOBA-S Singularity 利用環境

Singularity は、AOBA-S の以下のディレクトリにインストールされています。
/mnt/lustre/ap/singularity/3.11

ログイン時の環境変数 PATH には含まれていないため、「module」コマンドを使用して環境変数 PATH への追加を行ってください。

```
$ module load Singularity/3.11
```

2.2　Singularity の使い方

Singularity(コマンド) の利用法方法の詳細については、別途東北大学サイバーサイエンスセンターが提供しているマニュアルを参照してください。

以下に、本書で使用するコマンドの概要を記載します。

$ singularity [global option] **<command>** [option] ...

<Command>	**説明**
exec	コンテナ内で指定コマンドを実行します
shell	コンテナ内でシェルを起動します

option	**説明**
-e (--cleanenv)	コンテナ内でコマンドまたはシェルを実行する際に、ホスト側で設定された環境変数をクリアします

2.3　Frovedis コンテナイメージの準備

Frovedis がインストールされたコンテナイメージは、以下に準備されています。
/mnt/lustre/ap/Frovedis/frovedis.sif

AOBA-S フロントエンドサーバ上で、ホームディレクトリ配下の任意のディレクトリに、コンテナイメージをコピーしてください。

2.4　環境設定

(1) ディレクトリのバインド指定

Singularity では、ホスト側のディレクトリをコンテナにバインドすることで、ホスト側のディレクトリをコンテナ内から参照することが可能となります。
デフォルトで以下のディレクトリがバインドされ、コンテナ内で参照が可能となります。

- $HOME
- $PWD
- /tmp
- /etc
- /proc
- /sys
- /dev

Frovedis コンテナを利用する場合には、必要となるホスト側のディレクトリのバインドを追加する必要があります。以下の方法でディレクトリをバインドします。

```
$ export SINGULARITY_BIND=/mnt/lustre,/var/opt/nec,/opt/nec/aur_license
```

3　Python インタフェースの利用方法

AOBA-S で、Frovedis python インタフェースを利用する手順、およびチュートリアルの実行手順は、以下のとおりです。
尚、Frovedisコンテナで pythonインタフェースを利用する場合には、1ノード(1VH＋8VE，最大128並列)までの利用が可能です。

3.1　依存する python パッケージの準備

Frovedis の Python インタフェースは、複数の既存の Python パッケージに依存しています。そのため、Python の仮想環境を作成し、必要なパッケージ群をインストールします。

Python 仮想環境は、ホームディレクトリの任意のディレクトリに作成します。
以降の手順の説明では、~/venv/frovedis に仮想環境を作成した場合として記載します。
操作は、全て AOBA-S フロントエンドサーバ上で行います。

尚、本手順は、bash 環境で実施してください。

```
$ mkdir ~/venv
$ python3 -mvenv ~/venv/frovedis
$ source ~/venv/frovedis/bin/activate
(frovedis) $ pip install scikit-learn pandas "gensim==4.0.0" networkx
```

※AOBA-S で Frovedis を使用する際に必要な python パッケージ：
　scikit-learn, pandas, genism(4.0.0), networkx

3.2　Frovedis コンテナの実行確認

Frovedis コンテナが正常に動作するか、以下の手順で確認します。
以降の手順の説明では、ホームディレクトリ配下の ~/singularity/sif/frovedis.sif にコンテナイメージをコピーした場合として記載します。

```
$ cp /mnt/lustre/ap/Frovedis/frovedis.sif ~/singularity/sif/frovedis.sif
$ export IMAGE=~/singularity/sif/frovedis.sif   ※$IMAGE でコンテナを指定できる
                                                 ように環境変数を設定しています。
$ singularity shell ${IMAGE}
Singularity> ls /opt/nec/frovedis
data  getting_started.md  ve  x86
Singularity> exit
```

コンテナを立上げて、コンテナ内のシェルを起動し、コンテナ内にインストールされている frovedis ディレクトリ(/opt/nec/frovedis)を ls コマンドで参照しています。
上記のようにディレクトリが参照することができれば、コンテナの動作は正常です。

3.3　チュートリアルの実行

Frovedis コンテナには、python インタフェースのチュートリアルが格納されています。
チュートリアルを使用して、python インタフェースの AOBA-S での実行方法を記載します。

(1) チュートリアルのコピー

Frovedis コンテナから、ホームディレクトリ配下の任意のディレクトリにチュートリアルをコピーします。

```
$ singularity exec ${IMAGE} cp -r /opt/nec/frovedis/ve/doc/tutorial_python .
$ ls tutorial_python
src  tutorial_python.md  tutorial_python.pdf
$ cd tutorial_python/src
$ ls
tut3  tut5-1  tut5-2  tut5-3  tut5-4  tut5-5  tut6-1  tut6-2
$ cd tut3
$ ls
tut.py
$
```

tutorial_python.[md, pdf]: チュートリアル本文
src: チュートリアルプログラムディレクトリ

以降では、src/tut3 に格納されたサンプルプログラム(tut.py)を実行します。

(2) チュートリアルプログラム実行スクリプトの作成

Frovedis コンテナ内でチュートリアルプログラムを実行するためのスクリプトを作成します。
(チュートリアルの src/tut3 配下に作成。)

チュートリアル(tut3) プログラム実行スクリプト例: tut_cont.sh

```
#!/bin/sh
source ~/venv/frovedis/bin/activate
python tut.py
```

※ Frovedis の公開ドキュメントでは、実行に際し、
"source /opt/nec/frovedis/ve/bin/veenv.sh"
を実行し、Frovedis 用の環境設定を行う必要があるのですが、AOBA-S のコンテナ環境では自動的にこれを行うようになっているため、不要です。

作成したプログラム実行スクリプトに実行権を設定します。

```
$ ls
tut.py   tut.sh   tut_cont.sh
$ chmod +x tut_cont.sh
$
```

(3) ジョブ実行スクリプトの作成

qsub コマンドで Frovedis を実行するジョブを投入するジョブ実行スクリプトを作成します。
(チュートリアルの src/tut3 配下に作成。)

チュートリアル(tut3) ジョブ実行スクリプト例: tut.sh

```
#!/bin/sh
#PBS -q sxs
#PBS --venode 1
#PBS -l elapstim_req=0:10:00

module load Singularity/3.11
export SINGULARITY_BIND=/mnt/lustre,/var/opt/nec,/opt/nec/aur_license
IMAGE=~/singularity/sif/frovedis.sif
cd ${PBS_O_WORKDIR}
singularity exec ${IMAGE} ./tut_cont.sh
```

※ログイン環境によっては module の load が失敗する場合があります。投入用スクリプト内で「module load ～」行の前に、以下を記載することで回避できます。

source /etc/profile.d/modules.sh（投入用スクリプトが bash の場合）
source /etc/profile.d/modules.csh（投入用スクリプトが csh の場合）

(4) ジョブ実行スクリプトの投入

作成したジョブ実行スクリプトを AOBA に投入します。

```
$ qsub tut.sh
```

(5) 実行結果の確認

実行が終了すると、実行ジョブの標準出力(.o ファイル)に以下の出力がされていれば、プログラムは正常に実行されています。

```
score: 0.9560632688927944
```

4　C++API の利用方法

C++ API を使うことで、機械学習やデータフレーム機能以外にも、MPI を隠蔽して容易に分散処理が記述できる機能など、Frovedis が提供するより低レベルの機能を利用することができます。

AOBA-S で、Frovedis C++API を利用する手順、およびチュートリアルの実行手順は、以下のとおりです。
尚、C++API では、複数のノードを利用した並列処理を行うことが可能です。

4.1　チュートリアルの実行

(1) チュートリアルのコピー

Frovedis コンテナから、ホームディレクトリ配下の任意のディレクトリにチュートリアルをコピーします。

```
$ singularity exec ${IMAGE} cp -r /opt/nec/frovedis/ve/doc/tutorial .
$ ls tutorial
src   tutorial.md   tutorial.pdf
$ cd tutorial/src
$ ls
Makefile.each        tut2.4.4-1   tut2.5.5     tut3.1.3-2   tut4.5.1-2   tut5.3
Makefile.each.omp    tut2.4.4-2   tut2.6.1     tut3.2.1-1   tut4.5.2-1   tut5.4
Makefile.in.ve       tut2.4.5     tut2.6.2     tut3.2.1-2   tut4.5.2-2   tut5.5
exec_ve.sh           tut2.4.6     tut2.7-1     tut3.2.2     tut4.5.3-1   tut5.6
make.sh              tut2.4.7-1   tut2.7-2     tut3.2.3     tut4.5.3-2   tut5.7
make_clean.sh        tut2.4.7-2   tut2.7-3     tut4.1-1     tut4.6       tut5.8
tut2.1               tut2.4.7-3   tut2.7-4     tut4.1-2     tut4.7-1     tut5.9-1
  :
$ cd tut2.1
$ ls
Makefile   tut.cc
$
```

tutorial_python.[md, pdf]: チュートリアル本文
src: チュートリアルプログラムディレクトリ

以降では、src/tut2.1 に格納されたサンプルプログラム(tut.cc)を実行します。

(2) C++プログラムの make

Frovedis を使用する C++ プログラムを、Frovedis コンテナ内で make します。
(チュートリアルの src/tut2.1 配下で実行。フロントエンドサーバにインストールされている MPI と環境変数の競合が発生する可能性があるため、環境変数を引き継がないオプション(-e)をつけています。)

```
$ singularity exec -e ${IMAGE} make
  :
mpinc++ -c -O4 -fno-defer-inline-template-instantiation -finline-functions -finline-max-depth=10 -finline-max-function-size=200 -fno-cse-after-vectorization -Wno-unknown-pragma -fdiag-vector=2 -fdiag-inline=0 -msched-block -D_MPIPP_INCLUDE -I/opt/nec/frovedis/ve/opt/boost/include -I/opt/nec/ve3/nlc/3.1.0/include -I/opt/nec/frovedis/ve/include tut.cc -o tut.o
  :
$ ls
Makefile   tut.cc   tut   tut.o
```

コンテナ内でコンパイラが起動し、tut.cc がコンパイルされ、tut というバイナリが作成されます。
このバイナリはコンテナ内のファイルに依存しない MPI の実行ファイルになっています。そのため、通常の MPI プログラムの実行と同様のスクリプトで実行することができます。

(3) ジョブ実行スクリプトの作成

qsub コマンドで Frovedis を実行するジョブを投入するジョブ実行スクリプトを作成します。
(チュートリアルの src/tut2.1 配下に作成。)
make したバイナリは、通常の MPI プログラムですので、mpirun で実行します。

チュートリアル(tut2.1) ジョブ実行スクリプト例: tut.sh

```
#!/bin/sh
#PBS -q sxs
#PBS --venode 1
#PBS -l elapstim_req=0:05:00
cd $PBS_O_WORKDIR
mpirun -np 4 ./tut
```

(4) ジョブ実行スクリプトの投入

作成したジョブ実行スクリプトを AOBA に投入します。

```
$ qsub tut.sh
```

(5) 実行結果の確認

実行が終了すると、実行ジョブの標準出力(.o ファイル)に以下の出力がされていれば、プログラムは正常に実行されています。

```
2
4
6
8
10
12
14
16
```

参考 1：python インタフェースチュートリアルプログラム tut.py

```
import os
from sklearn.datasets import load_breast_cancer

from frovedis.exrpc.server import FrovedisServer
from frovedis.mllib.linear_model import LogisticRegression
#from sklearn.linear_model import LogisticRegression # sklearn

X, y = load_breast_cancer(return_X_y=True)

FrovedisServer.initialize("mpirun -np 4 {}".format(os.environ['FROVEDIS_SERVER']))
clf = LogisticRegression(random_state=0).fit(X, y)
score = clf.score(X, y)
FrovedisServer.shut_down()

print("score: {}".format(score))
```

from frovedis.exrpc.server import FrovedisServer によって、後で利用する FrovedisServer を import しています。
次に、from sklearn.linear_model import LogisticRegression の代わりに from frovedis.mllib.linear_model import LogisticRegression を import しています。このように、scikit-learn では sklearn.*の部分を frovedis.mllib.*に変更することで、多くのアルゴリズムが Frovedis で実行できるようになります。

FrovedisServer.initialize("mpirun -np 4 {}".format(os.environ['FROVEDIS_SERVER']))では、VE 上で動作するサーバプログラムを起動します。引数が起動のためのコマンドで、ここでは”mpirun -np 4”のように 4 コアでサーバを起動しています。サーバプログラムのバイナリのパスは FROVEDIS_SERVER という環境変数に設定されているので、これを取得して引数にしています。
このコマンドを書き換えることで、利用コア数を変えたり、複数の VE で実行したりすることができます 。

利用が終了したら、” FrovedisServer.shut_down()”を呼び出すことで、サーバプロセスを停止します。
機械学習の内容は、load_breast_cancer でテスト用のデータをロードし、LogisticRegression で学習、その結果の精度についてスコアを表示する、というもので、この部分については scikit-learn と違い

はありません。
このように、サーバの停止、起動と import の変更を除けば、scikit-learn と同じインタフェースで機械学習を行うことができます。Frovedis が提供する機械学習アルゴリズムのリストは https://github.com/frovedis/frovedis/blob/master/doc/misc/ml_algorithms.md にあります。

チュートリアルを参照して頂くことで、データフレームなど他の Frovedis の Python インタフェースの機能も理解して頂けると思います。
さらに詳しくは、コンテナ内の /opt/nec/frovedis/ve/doc/manual/manual_python.pdf や GitHub にあるマニュアル
（https://github.com/frovedis/frovedis/tree/master/doc/manual）
を参照してください。
また、コンテナ内の/opt/nec/frovedis/x86/foreign_if_demo/python/に Python インタフェースを使ったプログラム例がありますので、こちらも参考にしてください。

参考 2：C++API チュートリアルプログラム tut.cc

```
#include <frovedis.hpp>

int two_times(int i) {return i*2;}

int main(int argc, char* argv[]){
  frovedis::use_frovedis use(argc, argv);

  std::vector<int> v = {1,2,3,4,5,6,7,8};
  auto d1 = frovedis::make_dvector_scatter(v);
  auto d2 = d1.map(two_times);
  auto r = d2.gather();
  for(auto i: r) std::cout << i << std::endl;
}
```

“frovedis::use_frovedis use(argc, argv);”は初期化です。
”auto d1 = frovedis::make_dvector_scatter(v);”では、vector である v を MPI の各 rank に scatter し、分散された vector である d1 を作成します。
“auto d2 = d1.map(two_times);”では、分散された vector である d1 の各要素に対し、並列に関数を適用する機能である”map”を呼び出しています。ここでは、上で定義している two_times という関数を並列に適用しています。この場合 two_times は引数を 2 倍する関数なので、配列の各要素が 2 倍されることになります。これによって、分散された vector である d2 を新たに作成します。
“auto r = d2.gather();”で、分散された vector を gather し、std::vector である r を作成します。
次の行でこの結果を標準出力に出力しています。
そのため、確かに 1,2,3…という配列が 2 倍された結果である 2,4,6…という結果が出力されます。

このように、Frovedis の C++ API を使うと、MPI を直接利用するのに比べ、分散並列処理を簡潔に記述することができます。もちろん、機械学習やデータフレームあるいはその部品である行列演算といった機能も提供しています。

C++ API について詳しくは、tutorial.[md,pdf]およびそこから参照されている src/以下のプログラムを参考にしてください。