ジョブの投入方法(jsub)

(最終更新日: 2025/7/11)

ジョブの投入は jsub コマンドを使って行います。必要なインプットファイルは scp, sftp を用いて ccfep 上に配置するなどしてください。(scp, sftp に関してはクイックスタートガイドの情報をご確認ください。)

なお、Gaussian ジョブについては g16sub/g09sub という専用コマンドを用意しております。Gaussian を使う場合は、まず g16sub/g09sub の方をご検討ください。(一応 jsub でも投入は可能ですが面倒です)

ORCA についても osub という専用コマンドを用意しました。こちらもご利用ください。(ORCA を利用するには別途事前登録が必要です。)

基本的な実行方法

後述するジョブスクリプト(job.sh)を用意した上で ccfep にて以下のように実行します。(実際に入力するのは $ より後の jsub job.sh の部分だけです。)正常に実行できれば、ジョブの ID とサーバ名が表示されます。

```
$ jsub job.sh
6169323.ccpbs1
$
```

jsub
コマンド自体の実行はすぐに終了しますが、投入されたジョブがすぐに実行されるとは限りません。混雑時などにはしばらく ccfep からログアウトした後も残ります。

表示された数字のジョブのID(上記例では 6169323)はジョブを削除したりする場合に必要になります。ただし、jsub の実行時に表示されたものを常に覚えておく必要まではありません。通常は投入後に jobinfo
コマンドで表示されるものを確認する程度で十分です。

キューの指定は不要ですが、jsub -q H job.sh のように H キューを指定しても問題ありません。

jsub コマンドのオプションについては jsub --help を実行することで確認できます。投入された
ジョブの状況については jobinfo コマンドで確認することができます。投入済の
ジョブを取り消したい、強制的に終了させたい場合は jdel コマンドを使います。

ジョブスクリプトの作成

RCCS
で導入したアプリケーションで、ジョブとして実行することを想定されるものについてはジョブスクリプトのサンプルを用意
サンプルジョブの実行方法のページを参照ください
。独自のアプリを実行する場合にこれらサンプルをジョブスクリプトのテンプレートとして利用することもできるかもしれま

以下に一例として 32 MPI 並列、個々のプロセスでは OpenMP で 2
並列を行うジョブのジョブスクリプトを示します。ジョブは最大で 24
時間実行することにします。実行するプログラムは自身でビルドした my-program で、RCCS の用意した Open MPI
4.1.6 を使います。(以下のサンプルで ☒
で示した注釈部分を実際のファイルに書くとエラーになりますので、書かないようにお願いします。)

```
#!/bin/sh   ? 1 ????????(shebang)???
#PBS -l select=1:ncpus=64:mpiprocs=32:ompthreads=2  ? 2 ???????
#PBS -l walltime=24:00:00  ? 3 ??????
                    ? 4 ??????????????

# ?????????????????????(??; ???????????)
# ??????????????????????????????????????????
# ?????????????????????????????????
cd ${PBS_O_WORKDIR}

# ?????? module ???????????
module -s purge
module -s load openmpi/4.1.6

# ??????????????????????????
export PATH="/home/users/${USER}/bin:$PATH"

INPUT=myinp.inp
OUTPUT=myout.out

# ???????????????
mpirun -np 32 my-program -i ${INPUT} -o ${OUTPUT}
```

1 - インタープリター(shebang)の指定(必須)

スクリプトのインタープリター(shebang)を指定します。通常は以下のいずれかを指定することになるかと思います(sh/bash, csh/tcsh, zsh にそれぞれ対応します)。

- #!/bin/sh
- #!/bin/csh -f
- #!/bin/zsh

この1行目のヘッダーの指定は必須です。

2 - リソースの定義(select; 必須)

キューの構成や CPU 点数の係数についてはこちらのページを確認ください。

この部分のサンプルについては、こちらのページにもいくつか例を掲載しています。

select 文では CPU コア数、MPI 並列数、OpenMP 並列数、GPU 数を指定します。メモリの大きなノードを使う場合もここで指定が必要です。(行頭の#PBS -l も必須です。)基本的な書式は以下のようになります。

```
#PBS -l select=(????):ncpus=(CPU???):mpiprocs=(MPI???):ompthreads=(OpenMP???
)[:ngpus=(GPU?)][:jobtype=largemem]
```

ノード数について

- ncpus=1-63 で GPU を使わない場合(jobtype=core 相当)はノード数は常に 1 です。
- ncpus=64 もしくは 128 の時(jobtype=vnode 相当)、ノード数は 1 以上の数字となります。利用するCPUコア数は (ノード数)*(CPUコア数) となります。
- ngpus が 1 以上の場合(jobtype=gpu 相当)、ノード数は 1 以上の数字となります。利用するCPUコア数は (ノード数)*(CPUコア数) で、利用する GPU数は (ノード数)*(GPU数) です。

select= の直後に入力するノード数以外の情報についてはノードあたりの数字になります。mpiprocs の値は MPI の machine file の指定と関係します。ompthreads の値は環境変数 OMP_NUM_THREADS に相当する意味となります。

- ncpus= では利用するノードあたりの CPU コア数を指定します。指定できる数字は 1-64 と 128 です。
- mpiprocs= ではノードあたりの MPI 並列数を指定します。通常は mpiprocs * ompthreads の値が ncpus と一致するようにします。
- ompthreads= では各プロセスの OpenMP 並列数を指定します。通常は mpiprocs * ompthreads の値が ncpus と一致するようにします。OpenMP 以外のスレッド並列の場合はこの値は無視されます。
- ジョブで使用できるメモリ量は ncpus= で指定した CPU コア数に比例します。jobtype=largemem 指定時は 7.875 GB/コア で、それ以外の場合は 1.875 GB/コア です。
  キュー情報のページも参照してください。

MPI ライブラリのビルド方法によってはキューイングシステムから提供される machine file を正しく考慮できない可能があります。その場合は環境変数 PBS_NODEFILE で指定されているファイルを machine file (host file)として使う必要があります。RCCS で用意している MPI ライブラリは暗黙的にキューイングシステムが提供する machine file を考慮できるようになっています。

- GPU を使う場合は ngpus= で利用する GPU 数を指定することができます。なお、GPU あたりの CPU コア数(ncpus/ngpus)は 16 以下である必要があります。また、 1 ノードあたりの GPU 数が 8

であるため、ngpus の値は 1-8 の値となります。

```
# 32 CPU ????2 GPU ????
#PBS -l select=1:ncpus=32:mpiprocs=2:ompthreads=16:ngpus=2
```

- メモリが約 1 TB ある Type F ノードを使いたい場合は jobtype=largemem を指定します。Type F は全部で 14 ノードしかないため非常に混雑します。できるだけ通常の Type C ノードを使うようにしてください。なお、Type F ノードは GPU を搭載していないため、jobtype=largemem 指定時は ngpus を指定できず、ncpus の値も 64 か 128 となります。

```
# jobtype=largemem ????(2 ?????? 256 ??)
#PBS -l select=2:ncpus=128:mpiprocs=64:ompthreads=2:jobtype=largemem
```

3 - 計算時間の指定(walltime; 必須)

```
#PBS -l walltime=(??):(??):(??)
```

の書式で計算時間を指定します。walltime=30:00 のように書いた場合は 30 時間ではなく、30 分で指定されてしまいます。ご注意ください。

ジョブで消費される CPU 点数はここで指定した時間(walltime)ではなく、実際に実行した時間(elapsed time)から計算されます。そのため、ある程度は余裕を持って時間を指定していただいて大丈夫です。ただし、長い時間を指

次回のメンテナンス開始までに終了しないジョブについては次回メンテナンス終了後まで実行されません(この場合 jsub は warning を出力します。)しかし、順番待ちが長い場合はメンテナンスの前にあらかじめジョブを投入しておけば、その

4 - オプションパラメータの指定

以下のパラメータを使うと便利な場合があるかもしれません。

```
#PBS -m abe
```

```
#PBS -m n
```

メール送信機能: ジョブの開始時、終了時、abort 時にメールを送信する(-m abe)、あるいはメール送信を完全にやめる(-m n)ことができます。abe の a は abort 時、b は開始(begin)時、e は終了(end) 時にメールを送信することを示します。デフォルトは abort 時のみ送信します。-m abe のかわりに -m ae とすればアボート時と終了時だけメールが送信するようにもできます。-m n の場合はメールを送信しません。注意: 大量のジョブを実行する場合には b や e は指定しないことを推奨します。

```
#PBS -r n
```

ジョブの再実行抑制: ノードダウン等でジョブが abort した場合にはシステム側で強制的にジョブを別のノードで再実行します。一時ファイル等の影響で再実行されると困る、正

したジョブについては CPU 点数は消費されません。再実行された場合は再実行の時間分だけ CPU 点数が消費されます。)

```
#PBS -j oe
```

出力の merge: 通常は標準出力(stdout)と標準エラー出力(stderr)の内容は別々のファイルに出力されます。このオプションを加えると出力

j eo にすると標準エラー出力にまとめて出力できます。)

```
#PBS -N (???????)
```

ジョブの名前: この行を加えることでジョブに名前をつけることができます。例えば -N myjob1 とすれば myjob1 という名前になります。jsub 実行時に -N オプションを与えることでも同様に名前をつけられます(例: jsub -N myjob1 job.sh)。何も指定しない場合はジョブスクリプトのファイル名がジョブの名前になります。

ジョブの依存関係(-W depend=afterok, -W depend=afterany)

一例として、以下のような指定が可能です。ジョブの開始及び終了時にメール通知を行い、ノードエラー時の再実行は不可
merge して出力、ジョブの名前は myjob1 となります。

後述のステップジョブについてもご確認ください。そちらの方が使いやすい場合が多いと思われます。

一連のジョブ実行(ステップジョブ; --step もしくは --stepany)

```
#!/bin/sh
#PBS -l select=1:ncpus=64:mpiprocs=32:ompthreads=2
#PBS -l walltime=24:00:00
#PBS -m abe
#PBS -N myjob1
```

--step もしくは --stepany をつけることで比較的簡単にジョブを順番に実行することができます。

jsub 実行時に … 投入するジョブに依存関係を定義することができます。以下のように実行すれば、ジョブ 6169323 が正常に終了した場合に job2.sh を実行することができます。(ジョブ 6169323 が異常終了した場合は job2.sh が実行されることはありません。)

ジョブスクリプト中の変数を投入時に指定する(-v 変数名=値)

以下の例では job1.sh, job2.sh, job3.sh, job4.sh を順番に実行することができます。前のジョブがエラー終了した場合、後続のジョブは実行されません。上述の afterok の依存関係を指定した時と同じ挙動になっています。

-v オプションを利用して変数を定義することができます。前述のサンプルから INPUT=myinp.inp と OUTPUT=myjob.out の行を削除したもの(job.sh)を例にして動作を説明します。

jsub は通常ジョブを投入するだけですぐに終了しますが、-W block=true をつけるとジョブが終了するまで jsub …

特殊な利用方法について

```
#PBS -l walltime=24:00:00
$ jsub -W block=true job.sh
cd ${PBS_O_WORKDIR}
module -s purge
module -s load openmpi/4.1.6
export PATH=/home/users/${USER}/bin
mpirun -np 32 my_program < ${INPUT} > ${OUTPUT}
```

CPU コアはそれほど必要無いもののメモリが多く必要な場合

RCCS でメモリを多く使う場合には CPU コア数を増やす必要があります。例えば CPU は 1 コアしか使わないがメモリを 60 GB 使うのであれば、以下のように 32 コアを確保(メモリ約 60 GB)した上で 1 コアだけを使って計算するようにしてください。(CPU 点数は 32 コアを利用しているとして計算されます。)

1 つのジョブで複数の計算を実行したい場合

1 コアジョブを投入することもできますが、あまりにそのようなジョブが多いとキューイングシステムへの負荷が高くなるため、1 つのジョブにまとめることが必要かもしれません。例えば、32 個の計算を 1 つのジョブにまとめる場合の例です。

#PBS -l select=1:ncpus=32:mpiprocs=1:ompthreads=1

… で回避が可能です。長く連なったステップジョブを実行する場合は -m n オプションを追加することを推奨しています。(jsub --step -m n (ジョブ1) (ジョブ2) ...)

このインプットでは INPUT と OUTPUT というシェル変数が必要ですが、それを jsub …

それでも足りない場合は jobtype=largemem のノードを利用を検討ください。(ncpus=64 (メモリ約 500 GB/ノード)もしくは 128 (メモリ約 1 TB/ノード)の指定となります。)ただし、jobtype=largemem は …

この時、ジョブスクリプトの書き方が問題になりますが、以下のように書けば無駄が少なく、そして取りこぼしが起きるこ…

```
#!/bin/sh
#PBS -l select=1:ncpus=32:mpiprocs=1:ompthreads=1
#PBS -l walltime=24:00:00

cd ${PBS_O_WORKDIR}

sh job1.sh &
sh job2.sh &
sh job3.sh &
... (??)
sh job31.sh &
sh job32.sh &

wait # ????????????????????
```

最後の wait がポイントで、これがあることで先に実行した 32 個のプロセスの終了を待つことができます。(wait が無い場合はジョブスクリプトが終了してしまい、ジョブも終了したとみなされてしまいます。)この wait は bash の組み込み関数ですが、csh/tcsh でも同じような wait コマンドがありますので、上記の方法は csh/tcsh でも利用可能です。

上記では ncpus=32 (jobtype=core
相当)で実行していますが、細かいジョブの数が数千、数万となるようであれば ncpus=64 もしくは 128 (jobtype=vnode 相当)での利用を推奨します。