マリンインフォマティクス基盤システム
「MI-SURUGA」

ジョブスケジューラと
アプリケーション利用の実際

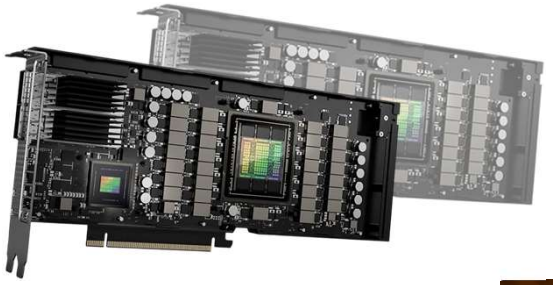
利用者説明会資料 クラスタ活用編

2025/6/2版

システム概要

GPUも備えたメニーコアHPCクラスタ：

◆計算ノード　CPU64コア、メモリ512GB、SSD 950GB
　汎用並列ノード　20台
　GPU搭載ノード　1台 @NVIDAI H100x2



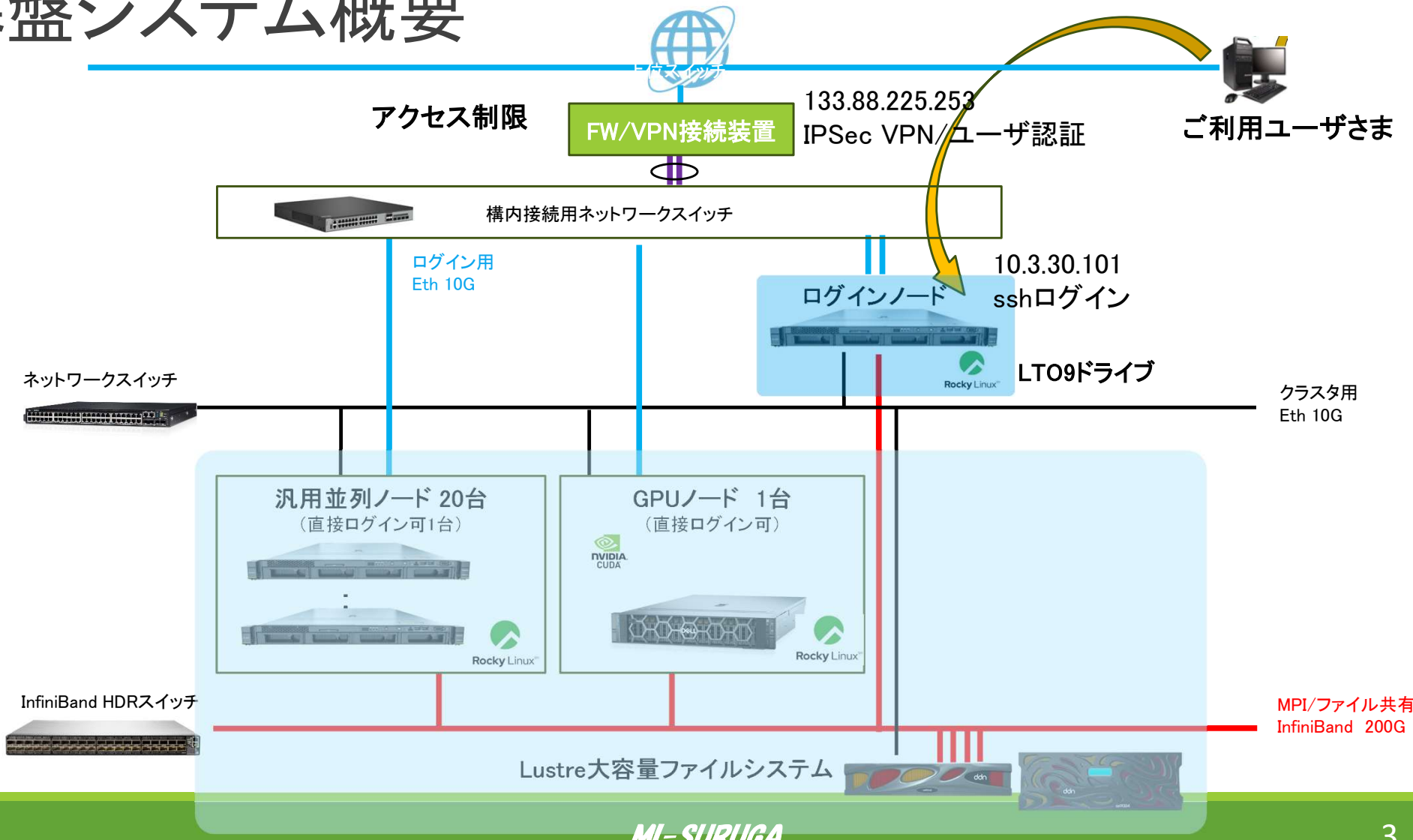
◆Lustre 高速ストーレジ(200Gpbs接続)　実効1PB

◆200Gpbs InfiniBandにてMPI計算が可能

◆基本環境
　OS：Rocky Linux 8.9
　JOBスケジューラ：OpenPBS
　（オプション コンテナ環境：Singularity）

◆利用方法
　IPsec VPN/ssh

基盤システム概要


インターネット
アクセス制限
FW/VPN接続装置
133.88.225.253
IPSec VPN/ユーザ認証
ご利用ユーザさま
構内接続用ネットワークスイッチ
ログイン用
Eth 10G
10.3.30.101
sshログイン
ログインノード
LTO9ドライブ
ネットワークスイッチ
クラスタ用
Eth 10G
汎用並列ノード 20台
（直接ログイン可1台）
GPUノード　1台
（直接ログイン可）
InfiniBand HDRスイッチ
MPI/ファイル共有
InfiniBand　200G
Lustre大容量ファイルシステム

目次

- OpenPBSとは
- ジョブ実行の流れ
- ジョブスクリプトの例：OpenMPI, IntelMPI
- GPUジョブ
- ジョブスクリプトによるダウンロードの例
- GUIアプリケーションとポート転送（JupyterLabの利用方法）
- GUIアプリケーションとX11転送（GrADSの利用方法）

OpenPBSとは：HPCのためのジョブスケジューラ

OpenPBSは、HPCクラスタにおけるバッチジョブの投入・管理・実行を効率化するジョブスケジューラです。各ジョブに対して必要な計算資源（CPUコア数、メモリ、実行時間など）を適切に割り当て、システム全体のリソースを最大限に活用します。

◆ **主な特徴**
- ジョブごとにリソース（CPU、メモリ、実行時間など）を指定可能
- 複数ユーザー間での公平な資源スケジューリング
- キューによるジョブの優先制御やグループ分け
- クラスタの状態を常時監視し、リソースが空き次第ジョブを順次実行

◆ **利用の流れ**
- ログインノード上でジョブスクリプト（シェルスクリプト）を作成
- “qsubコマンド”でジョブを投入
- OpenPBSが最適なタイミングとノードを自動的に選定して実行

※軽作業用のqloginは、Intrキューでリソースを確保します。
※バッチジョブの実行は、必要な計算資源を確保できるキューを指定してqsubコマンドで行ってください。

Job実行の制約

一人のユーザが大量のJobを投入して、システムを占有してしまうことがないよう、
以下の制約をかけています。

OpenPBS

◆**実行Job数の制限**

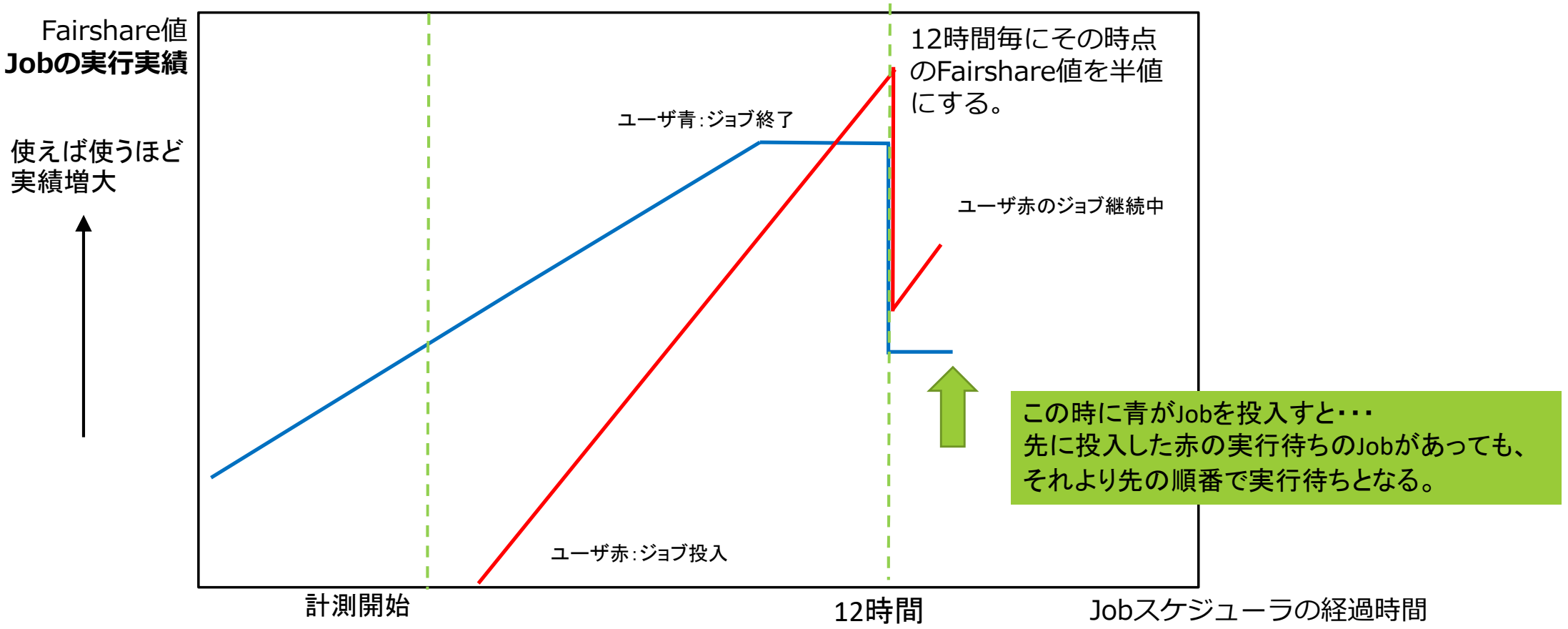
ユーザがあるキューで実行できるJob数の上限を設定しています。
その数以上のJobは投入できますが、実行待ちの状態になります。

◆**Fairshare機能設定**

ユーザのJob実行実績により、その時点で実績の少ないユーザのJobが優先して実行開始されます。

FairShare値：
Job実行で使用した（コア数 x 計算時間）の累積

Fairshareの動作イメージ



計算ノードとキュー構成

（キュー構成は運用状況により見直されます。）

計算ジョブの投入は、下記のキューを使って行います。
ログイン後qloginコマンドで作業環境に移動し、各種コマンドの実行やqsubコマンドでのジョブの投入を行います。

キュー	ノード種類	ノード	CPUコア数	メモリ量	使用可能 CPUコア数	使用可能メモリ量 /ジョブ	制限時間 /ジョブ	使用可能GPU数	実行Job数の制限	このキューでの実行例	利用目的
			デフォルト		指定すれば利用可能						
		設定Attribute	resources_default. ncpus	resources_default. mem	resources_max. ncpus	resources_max. mem	resources_max. walltime	resources_max. ngpus	max_run= [u:PBS_GENERIC=?]		
intr	計算ノード	cpunode01-03 gpunode	1 Core	4GB	2Core(medium) 4 Core(large)	32GB(medium) 64GB(large)	6 時間	0	4	（ログインノードのみ） qlogin 　または qlogin medium/large qlogin -l ncpus=3:mem=48gb	インターラクティブモードにより計算ノードにログインするためのキューです。 対話型Jobの実行、Jobスクリプトの作成やデバッグを行う想定。qlogin後 6 時間経つと自動ログアウトされます。 qlogin先でコマンドライン実行する場合は、リソースサイズを指定して、実行します。
mjobs (Defult キュー)	計算ノード	cpunode02-20	1 Core	4GB	4 Core	64GB	2日	0	64	qsub ＜Jobスクリプト＞ 　または qsub -q mjobs ＜Jobスクリプト＞	デフォルトでJobを実行するキューです。 指定することで最大4Core/64GBまで使用できます。 Jobスクリプトでの使用例 #PBS -l ncpus=4:mem=16gb
mpi	計算ノード	cpunode02-20	10Core	32GB	1024Core	無し	7日	0	10	qsub -q mpi ＜Jobスクリプト＞	MPI を使うJob実行用のためのキューです。
single	計算ノード	cpunode02-20	1Core	4GB	1Core	4GB	2日	0	64	qsub -q single ＜Jobスクリプト＞	最低限のリソースを設定したミニマムタスクキューです。
long	計算ノード	cpunode02-20	1Core	8GB	1Core	16GB	7日	0	32	qsub -q long ＜Jobスクリプト＞	比較的長い実行時間を想定した、最低限のリソースキューです。
gpu	GPUノード	gpunode	2Core	16GB	32Core	512GB	14日	2	10	qsub -q gpu ＜Jobスクリプト＞	GPUを必要とするJobの実行のためのキューです。 指定することで最大32Core/512GB/2GPUまで使用できます。 Jobスクリプトでの使用例 #PBS -l ncpus=8:mem=64gb:ngpus=2
copy	計算ノード	cpunode20	1Core	4GB	1Core	4GB	無し	0	2	qsub -q copy ＜Jobスクリプト＞	大容量のデータ転送を行うためのキューです. 汎用計算ノードの20号機で実行されます.

ジョブ実行の流れ

MI-SURUGAで処理を行うには、以下の2つの方法があります。

◆バッチジョブの投入（プログラム実行）の基本

最大6時間利用可能

① ログインノード または、ログインノードからqlogin コマンドで作業用ノード：cpunode01~03, gpunodeに移動
② ジョブスクリプトの作成
③ qsubコマンドでジョブを適当なキューに投入
④ qstatコマンドでジョブの状況を確認

qloginで、gpunodeでGPUを1基使用したい場合
例
$ qlogin -l host=gpunode:ngpus=1

◆qloginした上でのジョブの実行の仕方

ジョブの投入

qsub −q <キュー> −l <必要なリソース指定> ジョブスクリプト

※GPUを使用するジョブの場合
qsub −q gpu −l ngpus=1（あるいは2） ジョブスクリプト

GPUキューは最大14日利用可能

インターラクティブ（対話形式）での実行

qsub −I −q <キュー> −l <必要なリソース指定>

※GPUを使用するジョブの場合
qsub −I −q gpu −l ngpus=1（あるいは2）

GPUキューは最大14日利用可能ですが、
VPN接続は最大1週間に制限されています。
GPUは2基しかないので、いたずらに占有せず、
使用後は速やかにexit してください。

qloginコマンドの利用例

qloginコマンド実行で cpunode01 に入る
→ 直後にexit

```
[test@login ~]$ qlogin
qsub: waiting for job 1368.login to start
qsub: job 1368.login ready

[test@cpunode01 ~]$ exit
logout

qsub: job 1368.login completed
### Wed Apr  9 09:33:11 JST 2025 : qlogin exit ###
```

◆ qloginで明示的にgpunodeに入り、
→ module availコマンド実行

✓ デフォルトキューはmjobsです。
この状態で、キューを指定せずqsubを実行すると、
mjobsに投入されます。

```
[test@login ~]$ qlogin gpunode
qsub: waiting for job 1367.login to start
qsub: job 1367.login ready

[test@gpunode ~]$ module avail
--------- /usr/local/package/modulefiles/env --------
intelOneAPI/2025.0(default)
   :
      <以下略>
```

OpenMPI/gccを使ったJob実行例

① OpenMPI-gcc8.5.0/4.1.8の環境設定
$ module load OpenMPI-gcc8.5.0/4.1.8

② プログラムのコンパイル
例:姫野ベンチマーク
計算条件作成:L 128並列 (8x4x4=128)
$ ./paramset.sh L 8 4 4
コンパイル
$ **mpif77** himenoBMTxpr.f -O3 -o **128Lo4**

③ Jobスクリプトの作成
$ vi mpi-L.sh

④ Jobの実行
$ qsub mpi-L.sh

⑤ Jobの実行結果の確認
$ qstat （自分の実行しているJob一覧）

```
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------- -------- - -----
1427.login        STDIN            test2            00:00:00 R intr
1428.login        STDIN            washio           00:00:00 R gpu
1429.login        mpi-L.sh         test3                   0 R mpi
```

$ ls
mpi-L.sh.e1429　mpi-L.sh.o1429

mpi-L.sh
```
-----------------------
#!/bin/sh
#PBS -l select=2:ncpus=64:mpiprocs=64
#PBS -l place=scatter
#PBS -q mpi                                  <- mpiキュー指定
module load OpenMPI-gcc8.5.0/4.1.8
NCPU=`wc -l < $PBS_NODEFILE`
cd $PBS_O_WORKDIR
mpiexec -np $NCPU -machinefile $PBS_NODEFILE ./128Lo4
```

⑥ Jobの実行状況の確認

```
$ tracejob 1429

Job: 1429.login

04/22/2025 16:30:42  L    Considering job to run
04/22/2025 16:30:42  S    enqueuing into mpi, state Q hop 1
04/22/2025 16:30:42  S    Job Queued at request of test3@login, owner = test3@login, job name = mpi-L.sh, queue = mpi
04/22/2025 16:30:42  S    Job Run at request of Scheduler@login on exec_vnode (cpunode02:ncpus=64)+(cpunode03:ncpus=64)
04/22/2025 16:30:42  L    Job run
04/22/2025 16:30:56  S    Obit received momhop:1 serverhop:1 state:R substate:42
04/22/2025 16:30:58  S    Exit_status=0 resources_used.cpupercent=9081 resources_used.cput=00:22:14
                          resources_used.mem=7221040kb resources_used.ncpus=128 resources_used.vmem=509733552kb
                          resources_used.walltime=00:00:13
```

Intel MPIの利用

◆実行準備

$ module load intelOneAPI/2025.0

Loading intelOneAPI/2025.0
  Loading requirement: tbb/2022.0 advisor/2025.0 compiler-rt/latest umf/0.9.1 compiler/2025.0.4 compiler-rt/2025.0.4
    compiler-intel-llvm/2025.0.4 debugger/2025.0.0 dev-utilities/2025.0.0 dpct/2025.0.0 intel_ippcp_intel64/2025.0
    mkl/2025.0 vtune/2025.0 mpi/2021.14 ccl/2021.14.0

◆コンパイル

$ ./paramset.sh XL 8 4 4 ← パラメーターの設定
$ mpiifx -O3 himenoBMTxpr.f -o 128XLi ← コンパイル

◆ジョブスクリプトの作成

$ vi mpi-L.sh ← サンプルファイルを編集

mpi-L.sh

```
#!/bin/sh
#PBS -l select=2:ncpus=64:mpiprocs=64
#PBS -l place=scatter
#PBS -q mpi                          ← 使用するキューの指定
module load intelOneAPI/2025.0
NCPU=`wc -l < $PBS_NODEFILE`
cd $PBS_O_WORKDIR
mpiexec -np $NCPU -machinefile $PBS_NODEFILE ./128XLi
```

Intel MPIの利用

◆ **ジョブの投入と状況確認**

$ qsub mpi-L.sh　　<- ジョブの実行

$ qstat -n

```
login:
                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1442.login      washio   intr     STDIN      18743*   1   1    4gb 06:00 R 00:47
   cpunode01/0
1446.login      washio   intr     STDIN      18747*   1   1    4gb 06:00 R 00:40
   cpunode01/1
1452.login      test3    mpi      mpi-L.sh   18696*   2 128   32gb 168:0 R 00:00
   cpunode02/0*64+cpunode03/0*64
```

$ tracejob 1452

```
Job: 1452.login

04/22/2025 17:49:40  S    enqueuing into mpi, state Q hop 1
04/22/2025 17:49:41  L    Considering job to run
04/22/2025 17:49:41  S    Job Queued at request of test3@login, owner = test3@login, job name = mpi-L.sh, queue = mpi
04/22/2025 17:49:41  S    Job Run at request of Scheduler@login on exec_vnode (cpunode02:ncpus=64)+(cpunode03:ncpus=64)
04/22/2025 17:49:41  L    Job run
04/22/2025 17:50:42  S    Obit received momhop:1 serverhop:1 state:R substate:42
04/22/2025 17:50:44  S    Exit_status=0 resources_used.cpupercent=5577 resources_used.cput=01:02:08
                          resources_used.mem=18868292kb resources_used.ncpus=128 resources_used.vmem=1276340456kb
                          resources_used.walltime=00:01:00
```

GPUの利用：qloginコマンドの利用

- qloginコマンド実行で gpunode に入る
  → 直後にexit

```
[test3@login ~]$ qlogin -l host=gpunode
qsub: waiting for job 1441.login to start
qsub: job 1441.login ready

[test3@gpunode ~]$ exit
Logout
qsub: job 1441.login completed
### Tue Apr 22 17:01:18 JST 2025 : qlogin exit ###
```

- GPUを利用するには ngpusで使用GPU数を指定してください。
  ※インターラクティブ（対話型）モードの例

```
[test3@login ~]$ qlogin -l host=gpunode
qsub: waiting for job 1443.login to start
qsub: job 1443.login ready

[test3@ gpunode ~]$ qsub -I -q gpu -l ngpus=1

[test3@gpunode ~]$ nvidia-smi -L
GPU 0: NVIDIA H100 NVL (UUID: GPU-d6743c7e-6bcc-c53f-c61c-2cfdea86d9ee)
```

GPU利用の注意点

本システムではGPUはgpunodeに2枚搭載されています。
OpenPBSでGPUを利用するには　リソース行に
ngpus=1 あるいは ngpus=2　を明記する必要があります。

現在の仕様は排他的にGPUを利用できるメリットがありますが、ジョブ（インタラクティブジョブを含む）により2枚が利用宣言された場合、新規のジョブは実行待ちとなります。qloginでgpunodeにログインすることもできなくなります。

GPUの利用：ジョブスクリプトの例

ログインノードでgpunodeを使用するジョブの実行例

- GPUを使うにはキュー「gpu」を指定します。
- GPUを利用するには ngpusで使用数を指定します。
- ジョブスクリプトは以下の通り。

gpu_run.sh

```
-----------------------------------------------
#!/bin/bash
#
#PBS -l select=1:ncpus=16:ngpus=1
#PBS -q gpu            <- gpuキュー指定
#
export LANG=C
NCPU=`wc -l < $PBS_NODEFILE`
cd $PBS_O_WORKDIR
module load CUDA/12.5
nvidia-smi -L
./deviceQuery
```

```
[test3@login deviceQuery]$ qsub gpu_run.sh
1447.login
[test3@login deviceQuery]$ qstat
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------- -------- - -----
1442.login        STDIN            washio           00:00:02 R intr
1446.login        STDIN            washio           00:00:03 R intr
1447.login        gpu_run.sh       test3                   0 R gpu

$ cat gpu_run.sh.o1447
GPU 0: NVIDIA H100 NVL (UUID: GPU-d6743c7e-6bcc-c53f-c61c-2cfdea86d9ee)
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA H100 NVL"
  CUDA Driver Version / Runtime Version          12.8 / 12.5
  CUDA Capability Major/Minor version number:    9.0
  Total amount of global memory:                 95330 MBytes (99961274368
bytes)
  (132) Multiprocessors, (128) CUDA Cores/MP:    16896 CUDA Cores
  GPU Max Clock rate:                            1785 MHz (1.78 GHz)
  Memory Clock rate:                             2619 Mhz
  Memory Bus Width:                              6144-bit
  L2 Cache Size:                                 62914560 bytes
.....
```

ジョブスクリプトによるダウンロードの例

cpunode20は外部への接続が可能な設定です。
このノードを使ってデータやアプリケーションのダウンロードを行うためのキュー「copy」を使用します。

システムは、基本的にはインターネットへの接続を許可していません。
しかし、公共サイトやユーザ様がご用意したサイトに限り、直にデータファイルなどをダウンロードすることができます。

利用するサイトの事前利用申請をお願いします。

run.sh （Intel oneAPIのインストーラをダウンロードする例）

```
------------------------------------------------
#!/bin/bash
#PBS -q copy
#PBS -l select=1:ncpus=1
#
export LANG=C
#
wget https://registrationcenter-download.intel.com/akdlm/IRC_NAS/b7f71cf2-8157-4393-abae-8cea815509f7/intel-oneapi-hpc-toolkit-2025.0.1.47_offline.sh
```

参考　ジョブの実行確認（qstatのstatus表示）

状態	説明
B	Jobアレイのみに表記される状態です。（Jobアレイが起動）
E	Jobは実行済みで終了処理中です。
H	Jobは保留状態です。
Q	Jobはキュー待機状態です。
R	Jobは実行中です。
S	Jobはサーバによって中断中です。 他の優先度の高いJobで計算リソースが必要になると、Jobは中断状態になります。
T	Jobは移行中です。
U	Jobがビジー状態になったため中断中です。
W	Jobは要求された実行時間になるまで待機中であるか、 Jobは何らかの理由で失敗したステージイン要求を指定しています。
X	サブJobのみに表記される状態です。（時間切れにより、サブJobが終了）

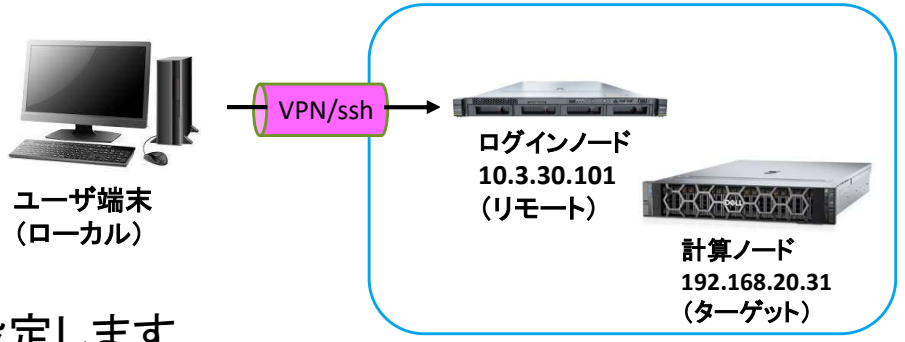
参考　ジョブの実行確認（qstatのオプション）

オプション	機能
-x	終了したJobの情報も表示します。
-n	**ジョブに割り当てられたホストリストも表示します。**
-a	ユーザのすべてのJobに対する情報を表示します。 要求されたメモリ量や要求された経過時間、Jobの状態の経過時間などが表示されます。
-r	-a オプションと同じフォーマットですが、実行中のJobのみを表示します。
-t	アレイJobのサブJobも表示します。
-f <jobID>	jobIDに対するもっとも詳細な情報を表示します。 Jobが既に終了している場合は、-x オプションを同時に指定して下さい。

GUIアプリケーションとポート転送 Jupyter Lab

GUIが必要なアプリケーションは、sshのポート転送によって、ご利用の端末に画面を表示可能です。
実際にアプリケーションを実行するのは何れかの計算ノードですので、ログインノードを経由して、
２重の転送が必要となります。

GPU搭載ノードでのJupyter Lab実行の例を示します。



■ユーザ様の使用端末側での設定

- Tera term/Rloginなどのsshターミナルソフトにポート転送を設定します。
  ログインノード（10.3.30.101）接続設定
  ポートフォワーディングを設定
  ローカル　localhost　8888ポート
  ターゲット　192.168.20.31　50001ポート

同じノードで複数のユーザがJupyterLabを使用する場合、
重複しないようUIDを利用したポート番号を必ず使用してください。

ポート番号のポリシー：UID＋40000
例　UID：10001 → 50001

- MacOSなどのターミナルでssh接続の場合は、ssh接続に際にポートフォワーディング設定を行います。

```
$ ssh  -L 8888:192.168.20.31:50001   test@10.3.30.101
```

GUIアプリケーションとポート転送 Jupyter Lab

■MS-SURUGA側での操作

① ログインノードへssh接続

② qlogin実行

③ GPU搭載ノードのリソース取得（gpunode:192.168.20.31）

```
$ qsub -I -q gpu (or qsub -I -q gpu ngpus=1)
```

④ Python環境のload（全Pythonバージョンに JupyterLabは導入されています）

```
$ module load  python/3.11.11 CUDA/
```

⑤ Jupyter Lab の起動　※1ユーザは、いずれかのノードで1起動のご利用に制限されます。

```
$ juyter-lab
```

⑥ ユーザ端末のWebブラウザでアクセス

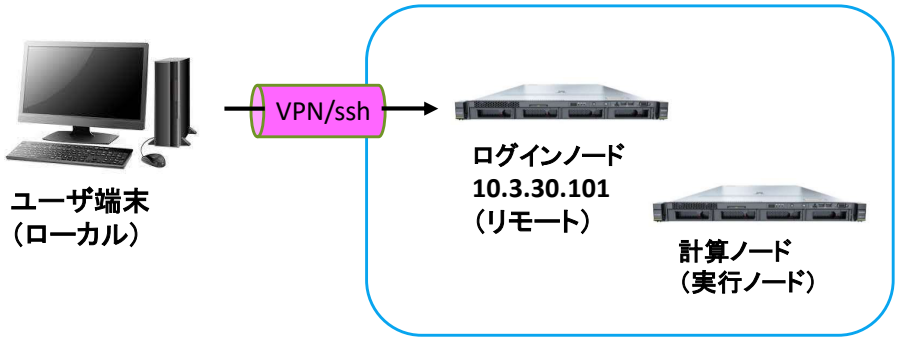
http://localhost:8888/

詳細は別紙ドキュメントを参照ください。

GUIアプリケーションとX11転送 GrADS

X11の描画が必要なGUIアプリケーションは、sshでX11転送を許可して接続してください。
qloginコマンドで実行されるOpenBPSのインタラクティブモードでは、X11のDISPLAY変数等の設定は自動で行われます。
実際にアプリケーションを実行するのは何れかの計算ノードですので、ログインノードを経由して、2重の転送が必要となります。

The Grid Analysis and Display System (GrADS)の実行例を以下に示します。



■ユーザ様の使用端末側での設定

① X Windowサーバ（エミュレーション）ソフトのインストール
Windows/Macのユーザ端末では、無償のXmigやXQuartz等や
有償のExceed、ASTEC-X、Reflection Desktop for Xなどの製品を
インストールしてください。

② Tera term/Rloginなどのsshターミナルソフトに X11転送を設定します。

GUIアプリケーションとX11転送 GrADS

■MS-SURUGA側での操作

① ログインノードへssh接続

② qlogin実行し、汎用計算ノード（又はGPUノード）に移動

④ GrADS環境のload

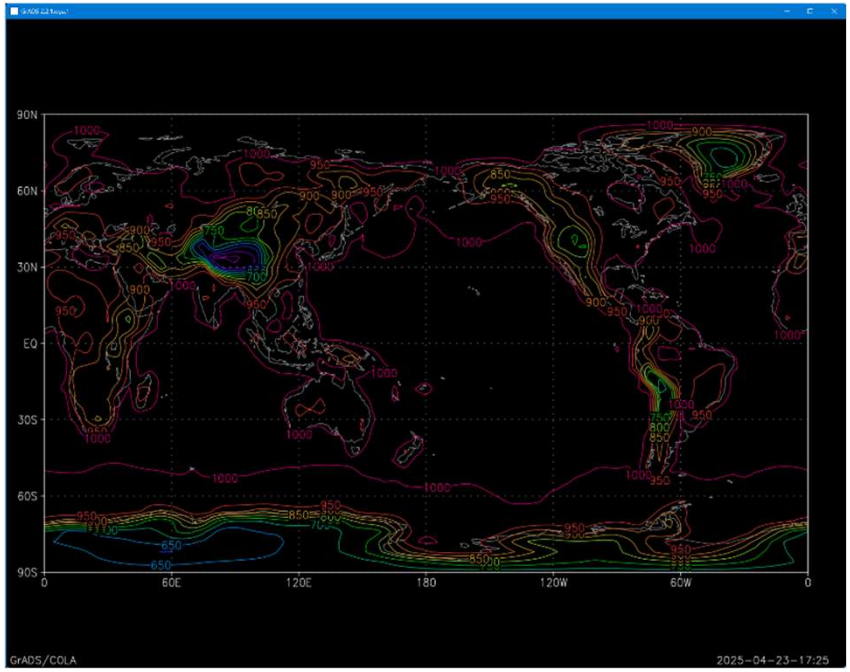
```
$ module load GrADS
```

⑤ GrADSの起動

```
$ grads
```

⑥ ユーザ端末の画面上にGrADSのウインドウが表示されます。

詳細は別紙ドキュメントを参照ください。



cora.gmu.edu のサンプル

参考ドキュメント

PBS Professional関連：

以下のURLで Product:PBS Proffesional, Version:2021.1.3として検索してください。

https://community.altair.com/community?id=altair_product_documentation

「**Altair PBS Professional 2021.1.2　User's Guide**」など

Inte oneAPI 関連

https://jp.xlsoft.com/documents/intel/oneapi/download/programming-guide.pdf

「インテル® oneAPI プログラミング・ガイド」