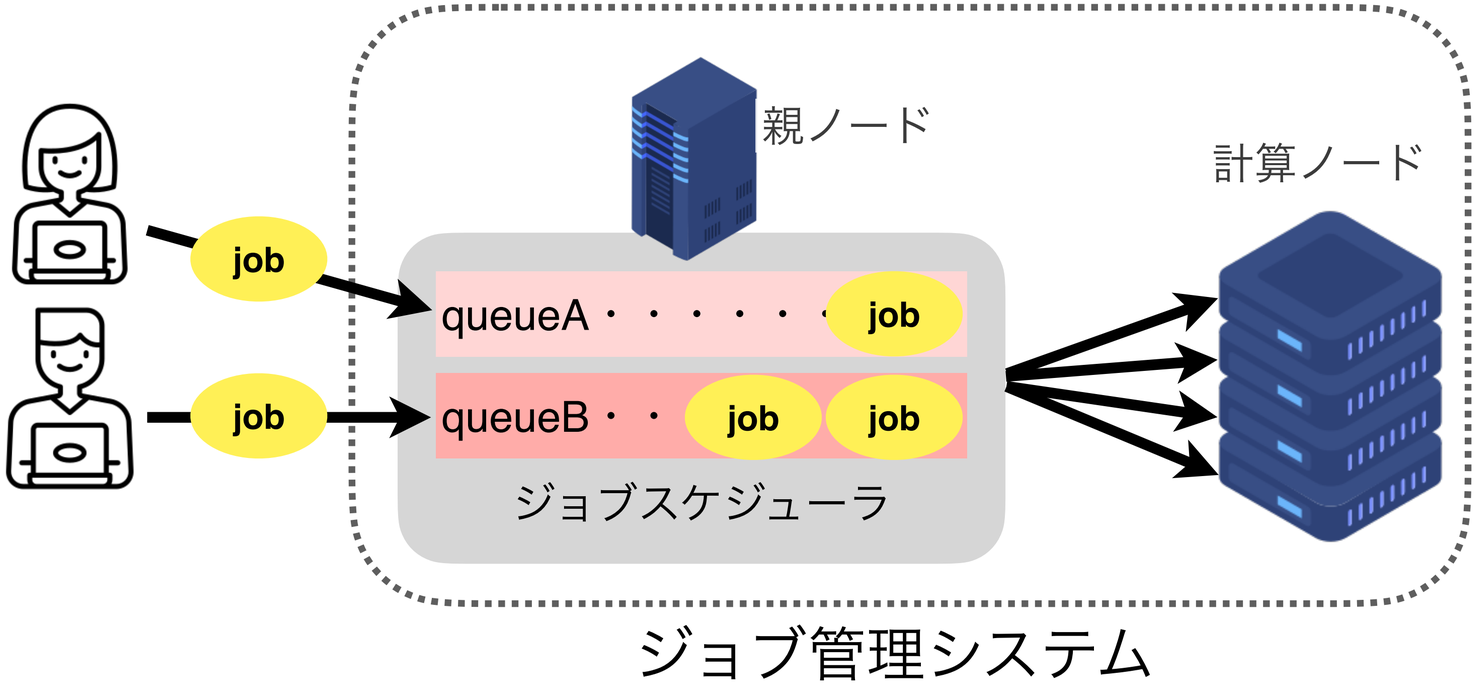
RCCS計算機利用方法

24/June/2024 Hiroyo NISHIDE

ジョブ管理システム：

- 複数の人間が同じ計算機群を使う
- どのマシン/CPUが空いてるか？ どの計算を優先させるべきか？

- 自動で計算機資源の割り当てて効率を上げる
- ユーザはコマンドで親ノードにジョブを投げるだけ
- ジョブは「キュー」と呼ばれるリストに入って実行を待つ



- ジョブ：実行するコマンドやアプリケーションを記述したシェルスクリプト形式のタスク

RCCSにおけるジョブ管理システム

全ての解析はジョブ管理システムを経由して実行すること

- 基本は**PBS**の仕様に従う
- RCCSではジョブ投入、ジョブ確認などに独自コマンドを採用している

コマンド対応表 PBS/RCCS ジョブ管理システム

PBSコマンド	RCCSコマンド	説明
qsub *file_name*	**jsub *file_name***	ジョブを投入する
qstat -u *my_account*	**jobinfo**	自分のジョブの状態を表示
qstat -Q	**jobinfo -s**	キュー/ジョブタイプの詳細を表示
qstat	**jobinfo -a**	全てのジョブを表示（-g で同グループのみを表示）
qdel *jobID*	**jdel *jobID***	指定したIDのジョブを削除
qhold *jobID*	**jhold *jobID***	指定したIDの待機中ジョブを実行されないようにホールド
qrls *jobID*	**jrls *jobID***	ホールド状態のジョブをリリース（実行待ち状態にする）
tracejob *jobID*	**joblog**	終了したジョブの履歴を表示（CPU点数も表示）

ジョブ実行の流れ

- 実行したいコマンドをシェルスクリプトに書く

testjob.sh

```
#!/bin/bash
#PBS -l select=1:ncpus=12:mpiprocs=1:ompthreads=12
#PBS -l walltime=4:00:00
source /apl/bio/etc/bio.sh
module load diamond
cd ${PBS_O_WORKDIR}
diamond blastp --db spo.dmnd --out sce.tab --outfmt 6 --query sce_prot.fasta
```

- **jsub** コマンドにシェルスクリプトのファイルを渡して実行

```
$ jsub testjob.sh
6439875.ccpbs1
```

- **jobinfo** コマンドでジョブの実行状況を確認

```
$ jobinfo
```

シェルスクリプトファイル (ジョブファイル)

```
#!/bin/bash
#PBS -l select=1:ncpus=12:mpiprocs=1:ompthreads=12
#PBS -l walltime=4:00:00
source /apl/bio/etc/bio.sh
module load diamond
cd ${PBS_O_WORKDIR}
diamond blastp --db spo.dmnd --out sce.tab --outfmt 6 --query sce_prot.fasta
```

←①シェルの指定

←② jsubオプション

←③module呼び出しとアプリケーションload

←④ジョブ投入ディレクトリへの移動

↑⑤実行するコマンド

① **シェルの指定：**ジョブスクリプトのシェルの種類を指定

② **jsubオプション：**行の先頭を #PBS で始めると、jsubコマンドのオプション指示として処理される

③ **module呼び出しとアプリケーションload：**前講義を参照のこと

④ **ジョブ投入ディレクトリへの移動**：jsubを実行したディレクトリに移動

（変数 ${PBS_O_WORKDIR} に jsub実行ディレクトリ名が自動で設定される）

これをしないとホームディレクトリにいる状態で続くコマンドを実行しようとする

⑤ **実行するコマンド ：** コマンド、パスの設定、変数のセット等スクリプト本体を記述

ジョブ実行：　jsub

- シェルスクリプトを作成

testjob.sh

```
#!/bin/bash
#PBS -l select=1:ncpus=12:mpiprocs=1:ompthreads=12
#PBS -l walltime=4:00:00

.....  (略)
```

- jsubコマンドにシェルスクリプトのファイルを渡して実行

```
$ jsub testjob.sh
6439875.ccpbs1
```

- jobinfo コマンドでジョブの実行状況を確認

```
$ jobinfo -c
--------------------------------------------------------------------------------
Queue   Job ID Name              Status CPUs User/Grp    Elaps    Node/(Reason)
--------------------------------------------------------------------------------
H(c)    6439875 testjob.sh        Run    12  ebv/zo0     0:00:20  ccc147
--------------------------------------------------------------------------------
```

ジョブの確認：jobinfo

- -c オプション：最新の状態を表示

```
$ jobinfo -c
-------------------------------------------------------------------------
Queue  Job ID   Name        Status  CPUs User/Grp   Elaps    Node/(Reason)
-------------------------------------------------------------------------
H      6439875  job1.sh       Run    12  ebv/---    0:00:20   ccc147
-------------------------------------------------------------------------
H      6439876  job2.sh       Run    12  ebv/---    0:00:20   ccc147
-------------------------------------------------------------------------
```

- オプションなし：数分古い情報だがjobtype, グループ名も表示

```
$ jobinfo
-------------------------------------------------------------------------
Queue  Job ID   Name        Status  CPUs User/Grp   Elaps    Node/(Reason)
-------------------------------------------------------------------------
H(c)   6439875  job1.sh       Run    12  ebv/zo0    0:00:20   ccc147
-------------------------------------------------------------------------
H(c)   6439876  job2.sh       Run    12  ebv/zo0    0:00:20   ccc147
-------------------------------------------------------------------------
```

自身とグループ使用状況の確認 jobinfo -s

```
$ jobinfo -s

User/Group Stat:
-------------------------------------------------------------------------
 queue: H                              | user(ebv)                | group(zo0)
-------------------------------------------------------------------------
   NJob (Run/Queue/Hold/RunLim)        |    1/    0/    0/-       |    1/    0/    0/5000
   CPUs (Run/Queue/Hold/RunLim)        |   12/    0/    0/-       |   12/    0/    0/4096
   GPUs (Run/Queue/Hold/RunLim)        |    0/    0/    0/-       |    0/    0/    0/20
   core (Run/Queue/Hold/RunLim)        |    0/    0/    0/1600    |    0/    0/    0/1600
   lmem (Run/Queue/Hold/RunLim)        |   12/    0/    0/-       |    0/    0/    0/896
 sncore (Run/Queue/Hold/RunLim)        |    0/    0/    0/-       |    0/    0/    0/4096
-------------------------------------------------------------------------
 queue: HB                             | user(ebv)                | group(zo0)
-------------------------------------------------------------------------
   NJob (Run/Queue/Hold/RunLim)        |    0/    0/    0/-       |    0/    0/    0/-
   CPUs (Run/Queue/Hold/RunLim)        |    0/    0/    0/-       |    0/    0/    0/-
   GPUs (Run/Queue/Hold/RunLim)        |    0/    0/    0/-       |    0/    0/    0/-
-------------------------------------------------------------------------
```

jobinfoオプション（抜粋）

オプション	説明
`--help`	ヘルプを表示
`-c`	最新の情報を表示。-m, -w, -s と同時指定不可
`-s`	キューの空き状況、自身の利用状況を表示。他のオプションと同時指定不可
`-w`	ジョブ投入時の作業ディレクトリを表示　-c と同時指定不可
`-m`	メモリ使用量の表示　-c と同時指定不可
`-L`	実行中ホストの全リストを表示
`-n`	各計算ノードの状況を表示
`-g`	所属グループのユーザーのジョブも表示
`-a`	全ユーザーの情報を表示、ユーザー名やジョブ名などは隠蔽される

ジョブを削除する：jdel

```
$ jdel jobID
```

- ジョブIDの番号を指定して削除（IDはjobifnoなどで確認）
- 自分がjsubしたジョブのみ削除可能
- CPU点数はそれまでかかった分消費される

```
$ jdel 6439875
```

ジョブ終了後：.e ファイルと .o ファイル

- ジョブ終了後に自動で作られるファイル

- 標準エラー出力の内容　***job_name.ejobID***
- 標準出力の内容　　***job_name.ojobID***

- ジョブが実行されたノード上で保持され、ジョブの終了後に*jsub*されたマシンに送られる
- エラーがあった際の詳細が書かれている

リソース消費量, 残量の確認　showlim/joblog

- これまでにグループで消費したCPU点数を表示

```
$ showlim -c
```

- これまでにグループで消費したCPU点数をメンバーごとに表示

```
$ showlim -c -m
```

- 過去のジョブが消費してきた点数を表示

```
$ joblog
```

- グループメンバーのストレージ使用量と残量を表示

```
$ showlim -d -m
```

カウントは年度変わり目でリセットされる

ジョブスクリプトにおけるリソース指定の書き方

- RCCSではキュー名の指定はなく、指定されたリソース量によって投入先ノードが変わる
- **#PBS -l** に続けて「：」で区切りながらリソースを指定する
- mem= で総メモリ量の指定はできない

```
#!/bin/bash
#PBS -l select=1:ncpus=12:mpiprocs=1:ompthreads=12
#PBS -l walltime=4:00:00
source /apl/bio/etc/bio.sh
module load diamond
cd ${PBS_O_WORKDIR}
diamond blastp --db spo.dmnd --out sce.tab --outfmt 6 --query sce_prot.fasta
```

RCCS用スクリプト例

ジョブスクリプトにおけるリソース指定の書き方

```
#!/bin/bash
#PBS -l select=1:ncpus=12:mpiprocs=1:ompthreads=12
#PBS -l walltime=4:00:00
```

- 必ず指定するリソース
  - ノード数：**select=**
  - ノードあたりのコア数：**ncpus=**
  - ノードあたりのプロセス数：**mpiprocs=**
  - プロセスあたりのスレッド数：**ompthreads=**
  - 計算時間（制限）　（時間）：（分）：（秒）：**walltime=**

ompthreads = ncpus x mpiprocs であること

1ノードで実行する

12コアを使う

プロセスは1つ

12スレッドで実行する

最大計算時間は4時間

ジョブスクリプトにおけるリソース指定の書き方

- 必要に応じて指定するリソース

  - ジョブタイプ ： **jobtype=**

```
#!/bin/bash
#PBS -l select=1:ncpus=64:mpiprocs=1:ompthreads=64:jobtype=largemem
```

7.785Gbx64=504Gbメモリ

  - 大容量メモリ（256Gb以上）を使いたい場合

  - ノードあたりのGPU数：**ngpus=**

```
#!/bin/bash
#PBS -l select=1:ncpus=128:mpiprocs=1:ompthreads=128:ngpus=8
```

8 GPU, 128コア

  - GPUを使いたい場合

ジョブ実行：　jsub

- シェルスクリプトを作成

```
#!/bin/sh
#PBS -l select=1:ncpus=12:mpiprocs=1:ompthreads=12
#PBS -l walltime=4:00:00
diamond blastp --db spo.dmnd --out sce.tab --outfmt 6 --query sce_prot.fasta
```

my_script.sh

- jubコマンドにシェルスクリプトのファイルを渡して実行

```
$ jsub my_script.sh
6439875.ccpbs1
```

キューは今のところ一種類

「H」

-

```
$ jobinfo
--------------------------------------------------------------------------
Queue   Job ID Name             Status CPUs User/Grp       Elaps Node/(Reason)
--------------------------------------------------------------------------
H(c)    6439875 test.sh          Run       4  ebv/zo0     0:00:20  ccc147
--------------------------------------------------------------------------
```

jobtype=　ジョブタイプについて

jobtype	計算ノード	メモリ/コア	利用単位	1ジョブあたりの制限	特徴
largemem	TypeF	7.875 Gb	vnode / ノード	1～14 vnode(s) （64～896コア） （ncpus=64 or 128）	大容量メモリ 1/2ノード(1vnode) 以上占有
vnode	TypeC	1.875 Gb	vnode / ノード	1～50 vnode(s) （64～3,200コア） （ncpus=64 or 128）	1/2ノード(1vnode) 以上占有
core	TypeC	1.875 Gb	コア	1～63 コア （ncpus<64）	64コア以下 総メモリ118Gb以下
gpu	TypeG	1.875 Gb	コア	1～48 GPU, 1～16 コア/GPU (ngpus>0)	GPU利用

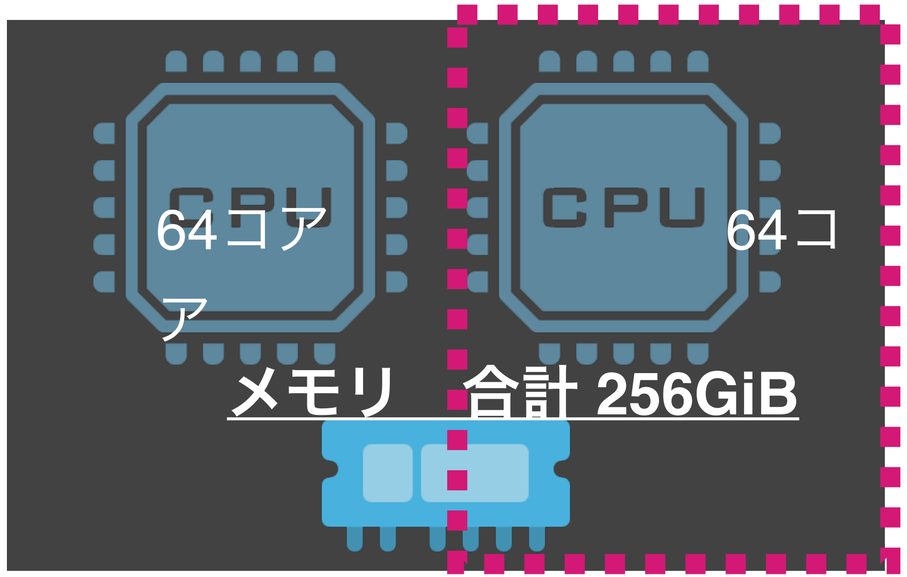
- largemem 以外の jobtype は指定されたリソースより判定可能なため省略可
- 使えるメモリ量は、指定コア数 x 計算ノードタイプのコア当たりのメモリ量

出典：https://ccportal.ims.ac.jp/manual/queues

ノードタイプとジョブタイプ

TypeC 806台

jobtype : core / vnode

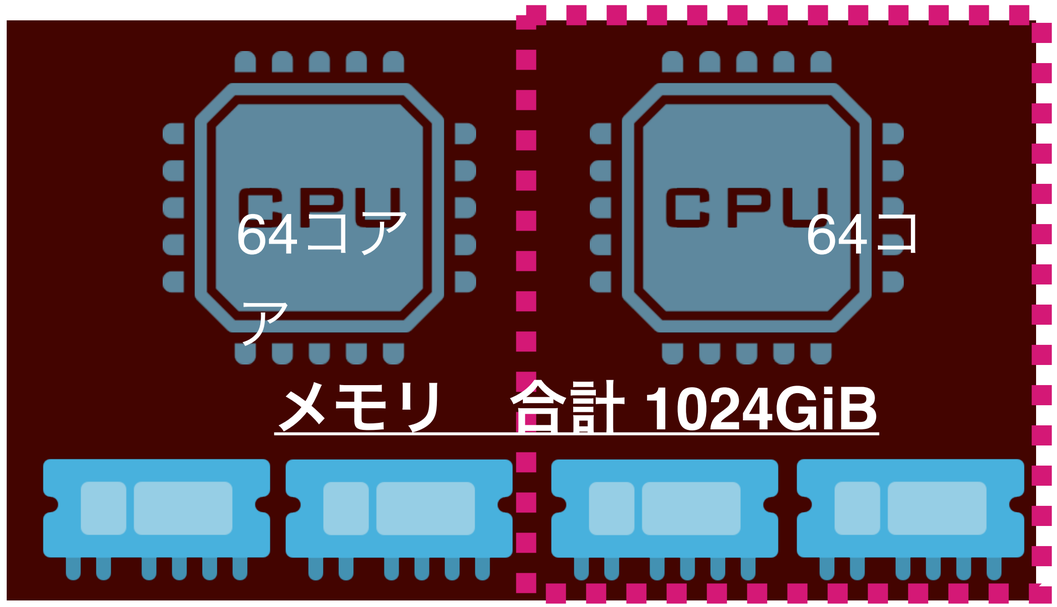


Vノード （1/2ノード）

jobtype=vnode で使われる単位

TypeF 14台

jobtype : largemem

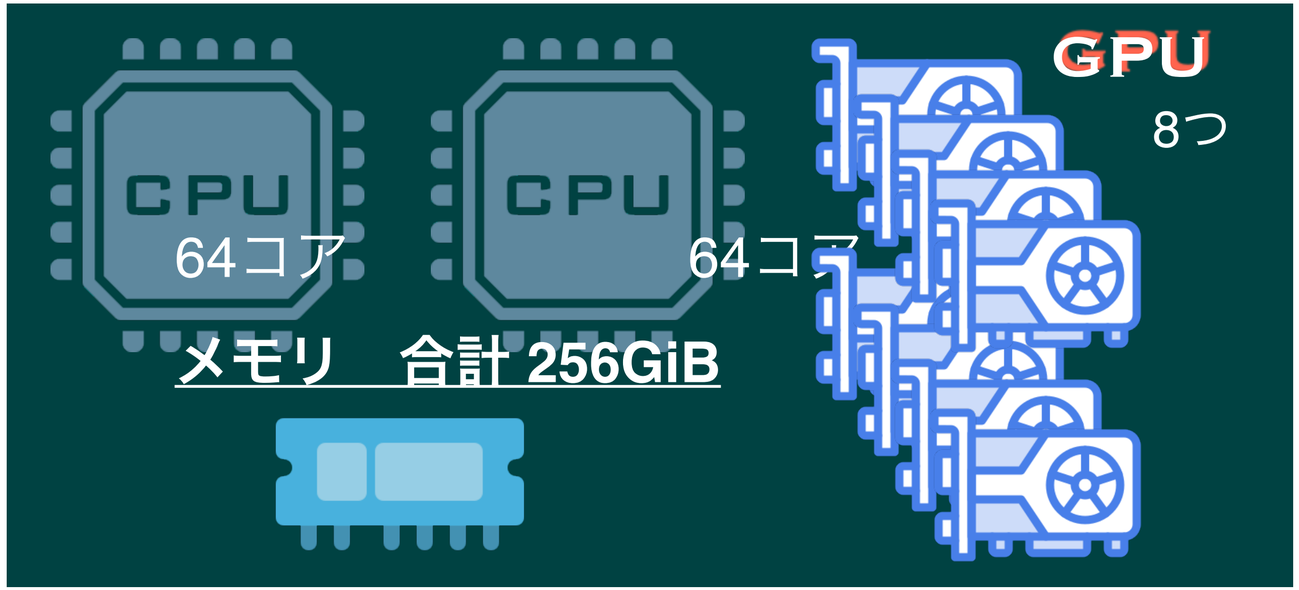


Vノード （1/2ノード）

ノードタイプとジョブタイプ

TypeG　　16台

jobtype : gpu



1 ノードに

8GPU＋128CPUコア

1GPUあたり16コア

参考：

- 分子生物学アプリケーションにおいて、複数プロセス立ち上げや、複数マシンにまたがった計算を行えるものは非常に少ない
  - select>1 の指定が有効なアプリケーションはほとんどない
- 複数ノード指定が有効なのは、物理マシンをまたがって並列計算できるアプリケーション (MPI等を使う）のみ

- PBSによって制限されたメモリを超えて使おうとして、PBSによりジョブが強制終了されることはない（biasとの相違点）
  - アプリケーション上で足りなくなって落ちることはある
- 指定したwalltime以内に終わらないとPBSによって強制終了

CPU点数

jobtype	CPU キュー係数	GPU キュー係数
largemem	60 点 / (1vnode * 1 時間)	-
vnode	45 点 / (1vnode * 1 時間)	-
core	1 点 / (1コア * 1 時間)	-
gpu	1 点 / (1コア * 1 時間)	60 点 / (1GPU * 1 時間)

- 消費点数は実際のジョブ実行時間で計算される。　walltimeではない

出典：https://ccportal.ims.ac.jp/manual/queues

CPU点数計算例

`select=1:ncpus=16:mpiprocs=1:ompthreads=16`

- diamond blastx：nr検索、 クエリDNA配列：96,706件、クエリファイルサイズ183MB
- 1ノード、16コア指定、jobtype: **`core`**
- 所要時間: 73:54:05、typeCノードで実行された
- 消費点数： 1点 x 16コア x 73.9時間　＝ 1,182点

`select=1:ncpus=64:mpiprocs=1:ompthreads=64:jobtype=largemem`

- InterProScan クエリDNA配列：115,079件、クエリファイルサイズ227MB
- 1ノード、64コア指定（1 vnode)、jobtype: **`largemem`**
- 所要時間： 16:30:22、typeFノードで実行された
- 消費点数： 60点 x 1vnode x 16.5時間　= 990点

アレイジョブ

- オプション　**#PBS -J** 開始番号-終了番号
- ジョブファイル中で変数 **${PBS_ARRAY_INDEX}** に番号がセットされ、インクリメントしながら回数分実行される

```
#!/bin/sh
#PBS -J 1-5
#PBS -l select=1:ncpus=4:mpiprocs=1:ompthreads=4
#PBS -l walltime=4:00:00

diamond blastx --threads ${NCPUS} --db nr --outfmt 6 \
 --query ./my.${PBS_ARRAY_INDEX}.fa --out ./out.${PBS_ARRAY_INDEX}.tab
```

- 上記は4コアジョブが5回サブミットされたのと同じ
- 消費点数は5つのジョブそれぞれがかかった時間に依存する
- あまり細かく分割しないこと （大量サブミットしたユーザのジョブは実行が遅れる）

アレイジョブ

```
$ jobinfo
-------------------------------------------------------------------------
Queue   Job ID Name              Status CPUs User/Grp        Elaps Node/(Reason)
-------------------------------------------------------------------------
H(c)   6441620[] test_array.sh  Array      4  ebv/zo0      0:11:39
H(c)   6441620[1] test_array.sh  Run        4  ebv/zo0       0:10:17 ccc035
H(c)   6441620[2] test_array.sh  Run        4  ebv/zo0       0:10:16 ccc035
H(c)   6441620[3] test_array.sh  Run        4  ebv/zo0       0:10:17 ccc035
H(c)   6441620[4] test_array.sh  X          4  ebv/zo0       0:00:04 ccc039
H(c)   6441620[5] test_array.sh  X          4  ebv/zo0       0:00:04 ccc039
-------------------------------------------------------------------------
```

一時ファイルのためのディレクトリ

- 原則として `/tmp, /var/tmp, /dev/shm` の一時ディレクトリの使用は禁止

- jsubジョブ実行では、環境変数 ${TMPDIR} が 自動で上記以外に設定されている
- 書き込み可能容量　：　11.9Gb/コア
  - 不足する場合は、コア数を増やす

▸定期メンテナンス

▸報告書

- メンテナンス等の最新情報は **[RCCS] メールニュース** を参照のこと
- メンテナンス日は（9:00〜17:00）ログインできない
  - 現在は偶数月の１日
  - 不定期に変更あり
- 年度の変わり目ごとに報告書を提出すること
- 2024年6月1日締切

情報の在処など

- 計算科学研究センター(RCCS) Webサイト

  https://ccportal.ims.ac.jp/

- 基生研が編集している移行情報wiki

  https://biaswiki.nibb.ac.jp/menu/index.php/RCCS

- お問い合わせ先

  support@nibb.ac.jp

ストレージについて

- １メンバーあたり500Gb
- それ以上が必要な場合は申請時にその理由を記載
- CPU点数、ディスク容量の追加申請も可能

- グループメンバーのストレージ使用量と残量を表示

```
$ showlim -d -m
```