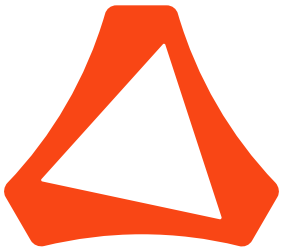

# Altair PBS Professional 2022.1

## User's Guide

altair.com

Altair PBS Professional 2022.1

# User's Guide（UG）

１０/１１/２２更新



# 連絡先

最新情報を入手するには、PBS Works の Web サイト www.pbsworks.com で “My PBS” を選択し、お客様のサイト ID とパスワードを入力してログインしてください。

## Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

## 営業

pbssales@altair.com 248.614.2400

ご質問と改善に関するご提案は agu@altair.com までお送りください。

# テクニカルサポート

テクニカルサポートをご希望の場合は、現地時間の午前 8 時から午後 5 時の時間帯にお問い合わせください。

| 場所 | 電話 | 電子メール |
|---|---|---|
| オーストラリア | +1 800 174 396 | anz-pbssupport@india.altair.com |
| 中国 | +86 (0)21 6117 1666 | pbs@altair.com.cn |
| フランス | +33 (0)1 4133 0992 | pbssupport@europe.altair.com |
| ドイツ | +49 (0)7031 6208 22 | pbssupport@europe.altair.com |
| インド | +91 80 66 29 4500<br>+1 800 208 9234（フリーコール） | pbs-support@india.altair.com |
| イタリア | +39 800 905595 | pbssupport@europe.altair.com |
| 日本 | 03 6225 5821 | pbs@altairjp.co.jp |
| 韓国 | +82 70 4050 9200 | support@altair.co.kr |
| マレーシア | +91 80 66 29 4500<br>+1 800 425 0234（フリーコール） | pbs-support@india.altair.com |
| 北米 | +1 248 614 2425 | pbssupport@altair.com |
| ロシア | +49 7031 6208 22 | pbssupport@europe.altair.com |
| 北欧 | +46 (0)46 460 2828 | pbssupport@europe.altair.com |
| シンガポール | +91 80 66 29 4500<br>+1 800 425 0234（フリーコール） | pbs-support@india.altair.com |
| 南アフリカ | +27 21 831 1500 | pbssupport@europe.altair.com |
| 南米 | +55 11 3884 0414 | br_support@altair.com |
| 英国 | +44 (0)1926 468 600 | pbssupport@europe.altair.com |

# 目次

# 目次

# PBSのドキュメントについて

PBS Professionalの各種ガイドとリリースノートは、PBS Professionalの*商用*リリースを対象としています。

## ドキュメント内の表記規則

Abbreviation

コマンドまたはサブコマンドの使用可能な最短の短縮名には下線が引かれています。

Attribute

属性、パラメータ、オブジェクト、変数名、リソース、型

`Command`

`qmgr`や`scp`などのコマンド

### Definition

定義される用語

`File name`

ファイル名およびパス名

`Input`

コマンドラインの指示

***Method***

メソッドまたはクラスのメンバー

`Output`

出力、サンプルコード、ファイルの内容

*Syntax*

構文、テンプレート、書式

***`Utility`***

プログラムなどのユーティリティの名前

*Value*

キーワード、インスタンス、状態、値、ラベル

## 表記

***オプションの引数***

オプションの引数は大括弧で囲んで表記されています。たとえば、`qstat`のマニュアルページでは`-E`オプションが次のように記述されています。

*qstat [-E]*

このオプションを使用するには、次のように指定します。

```
qstat -E
```

***変数の引数***

ジョブIDやvnode名などの変数の引数は山括弧で囲んで表記されています。この変数には実際の値を指定します。たとえば、pbsnodesのマニュアルページには次のように記述されています。

*pbsnodes -v <vnode>*

名前が“my_vnode”のvnodeに対してこのコマンドを使用するには次のように指定します。

```
pbsnodes -v my_vnode
```

***オプションの変数***

オプションの変数は、山括弧で囲み、さらにそれを大括弧で囲んで表記されています。たとえば、qstatのマニュアルページではジョブIDはオプションとなります。

*qstat [<job ID>]*

名前が“1234@my_server”のジョブを問い合わせるには次のように指定します。

```
qstat 1234@my_server
```

***リテラルターム***

リテラルタームは、使用されるとおりに表記されます。たとえば、コマンドのバージョンを取得するには、そのコマンドの後に“--version”を記述します。構文は次のとおりです。

*qstat --version*

使用方法は次のとおりです。

```
qstat --version
```

***複数の代替選択肢***

複数のオプションがあり、そこから1つを選択する場合は、そのオプションが中括弧で囲んで表記されています。たとえば、“-n”と“--name”のどちらかを使用できる場合は、以下のように記述されています。

```
{-n | --name}
```

# PBS Professionalのドキュメントリスト

PBS Professionalの各種ガイドとリリースノートは、PBS Professionalの*商用*リリースを対象としています。

*PBS Professional リリースノート*

サポートされるプラットフォーム、このリリースの新機能と想定されていない機能、非推奨事項とインターフェースの変更、未解決のバグと解決済みのバグ、最新情報などを取り上げています。管理者およびジョブ投入者を対象としています。

*PBS Professional Big Book*

すべてのお気に入りのPBSガイドが一か所に：*Installation & Upgrade Guide*、*Administrator's Guide*、*Hooks Guide*、*Reference Guide*、*User's Guide*、*Programmer's Guide*、*Cloud Guide*、*Budget Guide*および*Simulate Guide*が1冊にまとめられています。

*PBS Professional Installation & Upgrade Guide*

PBS Professionalをインストールおよびアップグレードする方法を説明しています。管理者を対象としています。

*PBS Professional Administrator's Guide*

PBS Professionalを構成および管理する方法を説明しています。PBS管理者を対象としています。

*PBS Professional Hooks Guide*

PBS Professionalでhookを記述および使用する方法を説明しています。PBS管理者を対象としています。

*PBS Professional Reference Guide*

PBSのコマンド、リソース、属性、設定ファイルなどに関するPBSのリファレンス情報を扱っています。

*PBS Professional User's Guide*

ジョブを投入、モニタリング、追跡、削除および操作する方法を説明しています。ジョブ投入者を対象にしています。

*PBS Professional Programmer's Guide*

PBSのアプリケーションプログラミングインターフェース（API）について説明しています。インテグレータを対象にしています。

*PBS Professional のマニュアルページ*

PBSのコマンド、リソース、属性、APIについて説明しています。

*PBS Professional Licensing Guide*

PBS Professionalのライセンスを設定する方法について説明しています。PBS管理者を対象としています。

*PBS Professional Cloud Guide*

クラウド上へジョブをバーストするためのPBS Professional Cloud機能を設定および使用する方法について説明しています。

*PBS Professional Budget Guide*

予算を設定してその動向を追跡し、PBSジョブによるリソースの使用量を管理します。Budgetの設定方法およびそれを使用してPBSジョブによるリソースの使用量を追跡、管理する方法について説明しています。

*PBS Professional Simulate Guide*

PBS Professional Simulate機能を設定および使用する方法について説明しています。

# ドキュメントの保管場所

*Big Book*を使用していない場合は、相互参照が機能するように、すべてのPBS Guideを同じディレクトリに置きます。

# ソフトウェアおよびライセンスの注文

ソフトウェアパッケージまたは追加ソフトウェアライセンスの購入を希望される場合は、Altairの販売担当者（pbssales@altair.com）にご連絡ください。

# PBS のドキュメントについて

# 1 PBSについて

## 1.1 PBSを使用する理由

PBSを使用すると、ユーザーは作業を完了するメカニズムを気にする必要がなくなります。1つ1つのジョブを適切なマシンに割り当てたり、入出力をあちこちにコピーしたり、特定のマシンが使用可能になるまで待機したりする必要がありません。必要なのは、実行するタスクの要件を指定して、タスクをPBSに渡すことのみです。PBSは、スロットが開放されるまで各タスクを保留し、入力ファイルの実行ディレクトリへのコピー、タスクの実行、および結果の出力を処理します。

PBSは、使用可能なハードウェア、および待機中と実行中のタスクを追跡し続けます。各タスクの要件に合わせて適切なハードウェアおよび時間スロットを割り当てて、サイトのポリシーに従ってタスクが動作することを保証します。また、使用率とスループットを最大化します。

## 1.2 PBSのタスクとコンポーネント

### 1.2.1 PBSのタスク

PBSは、分散型ワークロード管理システムです。PBSは、1つまたは複数のコンピュータ上の計算処理ワークロードを管理およびモニタリングします。PBSは、以下の処理を実行します。

**ジョブのキューイング**

PBSは、1つまたは複数のコンピュータ上で実行されるジョブ（処理作業単位またはタスク）を収集します。ユーザーはPBSにジョブを投入します。PBSでこれらを実行する準備ができるまで、投入されたジョブはキューイングされます。

**ジョブのスケジューリング**

PBSは、どのジョブをいつどこで実行するかを、ジョブで要求されたリソースとサイト管理者が指定しているポリシーに従って選択します。PBSでは、管理者がさまざまな方法でジョブの優先順位を付けて、リソースを割り当てることができ、それによって効率やスループットを最大化します。

**ジョブのモニタリング**

PBSは、システムリソースを追跡し、使用ポリシーを施行し、使用状況をレポートします。ジョブの完了を追跡して、システム障害が発生しても、ジョブが動作することを保証します。

**出力を返す**

PBSでは、ユーザーが指定した場所にジョブの出力が返されます。35ページの第3章「ジョブの入力ファイルと出力ファイル」をご参照ください。

## 1.2.2 PBSのコンポーネントとプロセス

PBSは、以下の図のように、ジョブを実行するための一連のコマンドおよびシステムデーモンとシステムサービスで構成されています。

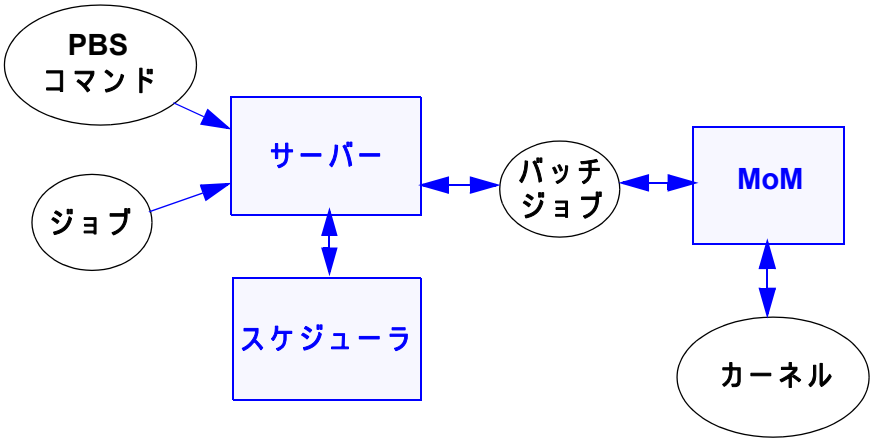


図1-1：ジョブがPBSサーバーに投入されます。スケジューラは、ジョブをいつどこで実行するかを選択し、ジョブをMoMに送信します。PBSコマンドは、サーバーと通信します。

サーバーデーモンとスケジューラデーモンは、サーバーホスト上で動作します。ジョブを実行するマシンは実行ホストと呼ばれます。各実行ホストがMoMデーモンを実行します。サーバーホストは、MoMデーモンを実行できます。１つのサーバーが、任意の数のMoMデーモンを管理します。デーモン間の通信は通信デーモンによって処理されます。コマンドは、サーバーホスト、実行ホスト、およびコマンド専用クライアントホストから実行できます。サーバー / スケジューラ / 通信ホスト、実行ホスト、およびクライアントホストは、*PBSコンプレックス*と呼ばれます。

### コマンド

PBSには、ジョブを投入、監視、変更、および削除する一連のコマンドが用意されています。PBSコマンドは、サポートされている任意のプラットフォームにインストールできます。他のPBSコンポーネントの有無は関係ありません。

PBSのコマンドやコマンドオプションには、あらゆるPBSユーザーが実行できるものと、実行するために昇格した権限を必要とするものがあります。

### ジョブ

PBSジョブは、実行するコマンドやアプリケーションを記述しているシェルスクリプト、cmdバッチファイル、Pythonスクリプトなどの形式のタスクです。タスクをPBSに渡すと、PBSジョブになります。

### サーバー

PBSサーバーは、PBSコンプレックスのジョブを管理します。PBSコマンドがPBSサーバーと通信し、ジョブがサーバーに投入され、サーバーがジョブをキューイングして実行ホストに送信します。

### スケジューラ

スケジューラは、サイト管理者が指定しているポリシーに従って、ジョブを実行します。スケジューラは、各ジョブの要件と使用可能なリソースを照合し、ジョブに優先順位を付け、ポリシーに従ってリソースを割り当てます。

### MoM

ジョブの実行ホストに送信されたジョブは、MoMによって管理されます。各実行ホストでは、1つのMoMでジョブを管理します。MoMは、ファイルをステージインしてprologueがあればそれを実行します。各ジョブを起動して監視し、ファイルをステージアウトして、ジョブ投入者に出力を返します。epilogueがあればそれを実行して、ジョブ終了後にクリーンアップします。また、任意の実行ホストhookを実行できます。

MoMは、ユーザーのログインセッションと可能な限り同一の新規セッションを作成します。たとえば、Linuxでは、ジョブの投入者のログインシェルがcshである場合、MoMは.loginに加えて.cshrcも実行されるセッションを作成します。

MoMは、Machine-oriented Mini-serverの略語です。

### 通信デーモン

通信デーモン（pbs_comm）は、他のPBSデーモン間の通信を処理します。

# 1.3 PBSに対するインターフェース

PBSはコマンドラインインターフェースを提供し、AltairはAccessと呼ばれるPBSのWebベースのフロントエンドを提供します。Accessは別製品です。本書では、PBSのコマンドラインインターフェースについて説明します。Accessの詳細についてはwww.altair.comをご参照ください。

## 1.3.1 PBSコマンド

PBSには、ジョブを投入、監視、および管理する一連のコマンドが用意されています。PBSコマンドには、任意のPBSユーザーが使用できるコマンドや、管理者のみが使用できるコマンドがあります。また、コマンドを呼び出すユーザーのロールに応じて動作が異なるコマンドもあります。本書では、任意のPBSユーザーが使用できるコマンドについて説明します。すべてのコマンドの詳細な説明とその要件については、『PBS Professional Reference Guide』の「List of Commands」（22ページ）をご参照ください。

**表1-1：PBSコマンド**

| コマンド | アクション |
|---|---|
| mpiexec | Linux上のPBSでMPIプログラムを実行します。 |
| pbsdsh | PBSでvnodeにタスクを配布します。 |
| pbsnodes | PBSホストまたはvnodeのステータス問い合わせ、ホストに対するfreeまたはofflineのマーク指定、ホストのコメント変更、またはvnode情報の出力を実行します。 |
| pbs_attach | PBSジョブにセッションIDを付与します。 |
| pbs_hostn | ホスト名とネットワークアドレスを報告します。 |
| pbs_login | 暗号化したユーザーパスワードを認証用としてキャッシュに保存します。 |
| pbs_mpihp | HP MPIを使用してPBSジョブの中でMPIアプリケーションを実行します。 |
| pbs_mpirun | MPICHを使用してPBSでMPIプログラムを実行します。 |
| pbs_python | コマンドラインでhookスクリプトをデバッグするためのPythonインタープリタ |
| pbs_ralter | 既存の先行予約、継続予約、またはジョブ固有予約を変更します。 |
| pbs_rdel | PBSの先行予約、継続予約、またはジョブ固有予約を削除します。 |

**表1-1：PBSコマンド**

| コマンド | アクション |
|---|---|
| pbs_release_nodes | PBSジョブに割り当てられているシスターホストまたはvnodeを解放します。 |
| pbs_rstat | PBSの先行予約、継続予約、またはジョブ固有予約のステータスを表示します。 |
| pbs_rsub | PBSの先行予約、継続予約、またはジョブ固有予約を作成します。 |
| pbs_tclsh | **非推奨**。TCLでラップしたPBS APIを使用したTCLシェル |
| pbs_tmrsh | MPI実装で使用するためにTM対応として`rsh/ssh`を置き換えたコマンド |
| pbs_wish | **非推奨**。TCLでラップしたPBS APIを使用したTKウィンドウシェル |
| qalter | PBSジョブを変更します。 |
| qdel | PBSジョブを削除します。 |
| qhold | PBSバッチジョブを保留状態にします。 |
| qmgr | PBSを管理する管理者用コマンドインターフェース |
| qmove | キューから別のキューにPBSジョブを移動します。 |
| qmsg | 1つまたは複数のジョブ出力ファイルにメッセージ文字列を書き込みます。 |
| qorder | キューで2つのPBSジョブが占める位置を互いに入れ替えます。 |
| qrls | 保留状態のPBSジョブを解放します。 |
| qsig | 指定されたPBSジョブを選択します。 |
| qsig | PBSジョブにシグナルを送信します。 |
| qstat | PBSジョブ、キュー、またはサーバーのステータスを表示します。 |
| qsub | PBSにジョブを投入します。 |

# 1.4 環境の設定

## 1.4.1 アカウントの前提条件

PBSが正常に動作するために、アカウントに以下の特性が必要です。

- アカウントがすべてのPBSホストにアクセスできること。
- アカウントの有効なユーザー名およびグループがすべての実行ホストとサーバーに設定されていること。
- アカウントが、管理者が選択したファイル選択メカニズムを使用して、ホスト間でファイルを転送できること。このことは、『PBS Professional Administrator's Guide』の第9.7節「Setting File Transfer Mechanism」（441ページ）の中で説明されています。
- 先行予約と継続予約を使用するため、タイムゾーン環境変数が正しく設定されていること。10ページの第1.4.4節「投入ホストのタイムゾーンの設定」をご参照ください。
- ユーザー名が256文字以内の長さであること。
- 環境が正しく構成されていること。
  - Linuxの場合は5ページの第1.4.2節「Linux環境の設定」をご参照ください。
  - Windowsの場合は7ページの第1.4.3節「Windows環境の設定」をご参照ください。
- アカウントにジョブを実行する適切なユーザー権限があること。
  - Linuxの場合は7ページの第1.4.2.7節「Linuxでのユーザー権限」をご参照ください。
  - Windowsの場合は8ページの第1.4.3.4節「Windowsでのユーザー権限」をご参照ください。

## 1.4.2 Linux環境の設定

### 1.4.2.1 PBSコマンドへのパス設定

PBSコマンドは、`$PBS_EXEC/bin`で参照されるディレクトリに格納されています。このパスはPBSのインストールごとに異なる可能性があるので、絶対パスではなく、変数を使用します。`$PBS_EXEC`の場所は、`/etc/pbs.conf`で指定されています。以下の手順を実行すると、簡単にPBSコマンドを使用できるようになります。

1. `.login`ファイルで`/etc/pbs.conf`を読み込みます。

   `bash`または`sh`を使用している場合は、以下を実行します。

   ```
   % ./etc/pbs.conf
   ```

   `csh`を使用している場合は、以下を実行します。

   ```
   %source /etc/pbs.conf
   ```

2. PBS コマンドへのパスを PATH 環境変数に追加します。絶対パスではなく、`$PBS_EXEC` を使用します。たとえば、`MY_PATH`に既存の一連のパスが設定されている場合、以下のように記述します。

   ```
   setenv PATH ${MY_PATH}:$PBS_EXEC/bin/
   ```

### 1.4.2.2 PBSマニュアルページへのパス設定

PBSマニュアルページへのパスをMANPATH環境変数に追加します。

```
setenv MANPATH /usr/man:/usr/local/man:$PBS_EXEC/share/man/
```

### 1.4.2.3 ジョブに読み込まれたログインファイルとログアウトファイルの正しい動作の保証

デフォルトでは、PBSはユーザーのジョブをそのユーザーのログインの下で実行します。つまり、ジョブを実行するたびに、そのユーザーのログインファイルとログアウトファイルが読み込まれます。.cshrc、.login、.profile、または.logoutに端末制御情報を設定するコマンドや、stdoutへの書き込みなどの出力を行うコマンドが含まれている場合、ジョブが動作しない可能性があります。これらのファイルがPBSジョブ内で実行される場合は、それらのコマンドがすべてスキップされるようにしてください。PBS は、ジョブ内で PBS_ENVIRONMENT 環境変数を設定します。PBS_ENVIRONMENT環境変数をテストして、設定されていない場合のみコマンドを実行するようにします。以下に、.loginファイルの場合の例を示します。

```
if ( ! $?PBS_ENVIRONMENT ) then
    ここで端末の設定を行います
    ここで出力を伴うコマンドを実行します
endif
```

### 1.4.2.4 正しいジョブ終了ステータスの取得

PBSジョブを実行した場合、最後に実行されたコマンドの終了ステータスがそのジョブの終了ステータスとして、ジョブのシェルによってPBSに報告されます。ジョブの終了ステータスは、ジョブの依存関係とジョブ連鎖にとって重要です。Linuxでは、実行ホストに.logoutが存在する場合、最後に実行されるコマンドがそのジョブの最後のコマンドであるとは限りません。この場合、最後に実行されるのは、ユーザージョブのコマンドではなく、.logoutのコマンドです。これを回避するには、以下に示すように、.logoutファイルの冒頭でジョブの終了ステータスを保存し、最後に明示的にexitを実行する必要があります。

```
set EXITVAL = $status
ここに.logoutファイルのこれまでの内容を記載
exit $EXITVAL
```

Windowsでは、ジョブの終了ステータスを保存する特別な手順を実行する必要はありません。

### 1.4.2.5 ジョブ内のバックグラウンドプロセスの回避

ログインファイルがPBSジョブから呼び出されるときに、そのログインファイルによってバックグラウンドでプロセスが実行されていないようにしてください。ログインファイルにPBSジョブ内でバックグラウンドで実行されるコマンドが含まれている場合、プロセスが永続化してトラブルを引き起こす可能性があります。

### 1.4.2.6 ジョブへのbash関数の提供

bash関数をエクスポートしてジョブから使用できるようにする必要がある場合、それらの関数を実行ホストの.profileまたは.loginに設定できます。qsub -Vまたはqsub -v <関数名>を使用して、ジョブ投入時に関数を転送することもできます。-vまたは-Vを使用する場合、環境変数と同じ名前の関数が存在しないことを確認してください。135ページの第6.12.4節「エクスポートされたシェル関数の転送」をご参照ください。

### 1.4.2.7 Linuxでのユーザー権限

サーバーのflatuid属性は、同じユーザー名が同じユーザーを意味するとみなすかどうかを判断します。*True*である場合、User1が投入ホストとサーバーホストの両方に存在すると、User1はそのサーバー上でジョブを実行できると想定します。*True*ではない場合、サーバーでは`ruserok()`を呼び出します。この`ruserok()`では、`/etc/hosts.equiv`または`.rhosts`を使用して、User1にUser1として実行する権限を与えます。この場合、-uオプションを使用してユーザー名を指定するには、サーバーホスト上の`.rhosts`ファイルにジョブオーナーがリストされている必要があります。つまり、サーバー上のUser1には、User1がリストされている`.rhosts`ファイルが必要です。

**表1-2：Linuxユーザー IDとflatuid**

| flatuidの値 | 投入ホストユーザー名とサーバーホストユーザー名 | |
|---|---|---|
| | User1とUser1で同じ | User1とUserAで異なる |
| *True* | サーバーは、ユーザーがジョブを実行する許可を得ていると想定します。 | サーバーは、User1がUserAとしてジョブを実行できるかどうかをチェックします。 |
| *False/未設定* | サーバーは、User1がUser1としてジョブを実行できるかどうかをチェックします。 | サーバーは、User1がUserAとしてジョブを実行できるかどうかをチェックします。 |

例1-1： ユーザーが、投入ホスト上ではUserA、サーバー上ではUserBであるとします。UserAとしてジョブを投入し、UserBとしてジョブを実行する場合、UserBには、UserAがリストされている`.rhosts`ファイルがサーバーホスト上に存在することが必要です。

-uオプションを使用して異なるユーザー名を指定した場合、flatuidの値に関係なく、チェックが行われます。

`hosts.equiv`の使用は推奨されません。

### 1.4.2.8 LinuxクライアントからのLinuxジョブの投入

Linuxクライアントホストでの認証方法がpwdに設定されている場合、Linuxジョブを投入する前に、これをmungeに設定し直します。以下に例を示します。

```
export PBS_AUTH_METHOD=munge; qsub -lselect=1:arch=linux -- sleep 100
```

## 1.4.3 Windows環境の設定

### 1.4.3.1 WindowsユーザーのHOMEDIR

PBSは、ジョブオーナーのホームディレクトリでジョブを起動します。このディレクトリは、`HOMEDIR`で参照されています。

ユーザーがホームディレクトリを明示的に割り当てていない場合、PBSは、Windowsが割り当てたデフォルトを、ユーザーのデフォルトホームディレクトリの基点として使用し、そこでジョブを起動します。Windowsは、以下のデフォルトホームパスを割り当てます。

```
[PROFILE_PATH]¥My Documents¥PBS Pro
```

たとえば、userAにホームディレクトリが割り当てられていない場合、デフォルトホームディレクトリは以下のようになります。

```
¥Documents and Settings¥userA¥My Documents¥PBS Pro
```

Windowsでは、PROFILE_PATHは次の書式のいずれかで返されます。

```
¥Documents and Settings¥username
¥Documents and Settings¥username.local-hostname
¥Documents and Settings¥username.local-hostname.00N
```

*N*は数字を表します

```
¥Documents and Settings¥username.domain-name
```

### 1.4.3.2 Windowsユーザー名の要件

- ユーザー名には英数文字、ドット（.）、アンダースコア（_）、ハイフン"-"のみを使用できます。
- ハイフンは、ユーザー名の先頭文字としては使用できません。
- ユーザー名内に"@"がある場合、Windowsドメインアカウントusername@domainnameのコンテキストとみなされます。
- スペース文字は許可されます。スペース文字がユーザー名内に現れる場合、文字列は引用符で囲まれて表示され、引用符で囲んで指定されなくてはなりません。

### 1.4.3.3 Windowsユーザーアカウントの要件

Windowsユーザーアカウントは、通常のユーザーアカウントである必要があります。SYSTEMアカウントからジョブを投入することはできません。

### 1.4.3.4 Windowsでのユーザー権限

PBSは、ユーザーのジョブをそのアカウントで実行します。ジョブがリモート実行ホストで動作する場合、そのアカウントを使用してログインし、ファイルを転送できる必要があります。システム管理者がhosts.equivでアクセスを設定していない場合、.rhostsファイルでアクセスを設定できます。サーバーの.rhostsファイルを使用することで、リモートマシンからサーバーにジョブを投入できます。

.rhostsファイルは、ユーザーのPROFILE_PATH、ホームディレクトリ、PBSサーバーホスト、および各実行ホストで設定します。以下に例を示します。

```
¥Documents and Settings¥username¥.rhosts
```

.rhostsファイルのフォーマットは以下のとおりです。

*hostname username*

.rhostsファイルはユーザーまたは管理者タイプのグループによって所有され、書き込み権限がユーザーまたは管理者、あるいは管理者グループのみに確実に与えられるようにしてください。

.rhostsファイルには、以下のように、すべてのPBSホストを追加します。

```
Host1 user1
Host2 user1
Host3 user1
```

ホストを認識できる名前をすべてリストしてください。たとえば、Host4が"Host4"、"Host4.<subdomain>"、または"Host4.<subdomain>.<domain>"として認識される場合は、３つすべてを.rhostsファイルにリストする必要があります。

```
Host4 user1
Host4.subdomain user1
Host4.subdomain.domain user1
```

ユーザー名に空白が含まれる場合、.rhosts ファイルでは引用符で囲みます。

```
Host4.subdomain.domain "Bob Jones"
```

例 1-2： サーバー上のユーザー user1 の .rhosts ファイルに以下のエントリが記述されている場合、ユーザー user1 はワークステーション wks031 から投入されたジョブを実行できます。

```
wks031 user1
```

実行ホスト Host1 で動作するジョブからの user1 の出力ファイルを PBS が自動的に user1 に返すことができるようにするには、user1 が実行ホスト Host1 を指定するエントリをワークステーション上の .rhosts ファイルに追加します。

```
Host1 user1
```

### 1.4.3.5 パスの設定

ジョブの投入またはファイルのステージインとステージアウトで、あるいは出力ファイルとエラーファイルの格納先として、マップされたドライブを使用する場合、ローカルシステムアカウントを使用してそのドライブをマップする必要があります。この場合、UNC パスを使用することをお勧めします。ローカルシステムアカウントを使用しない場合、ファイル転送動作が定義されません。ローカルシステムアカウントを使用して、グローバルにアクセスできるドライブをマップするには、SysInternals から psExec ユーティリティを使用します。

*<psexec バイナリへのパス> -s net use < マップされるドライブ文字>: < マップする UNC パス>*

以下に例を示します。

```
psexec -s net use Z: ¥¥examplehost¥mapping_directory¥mydirectory
```

マップされたドライブをマップ解除するには、以下を設定します。

*<psexec バイナリへのパス> -s net use /delete < マップされているドライブ文字>*

以下に例を示します。

```
psexec -s net use /delete Z:
```

PBS では、ユーザー名がサーバーとその実行ホストで一致している必要がありますが、投入ホストとサーバーとの間で一致している必要はありません。1 人のユーザーが 1 つまたは複数のサーバーにアクセスする場合や、各サーバーで異なるユーザー名を使用する場合もあることでしょう。ジョブのユーザー ID は変更できます。30 ページの第 2.5.4 節「ジョブのユーザー名の指定」をご参照ください。

### 1.4.3.6 ジョブ投入認証のパスワード

パスワードを変更した場合は必ず pbs_login コマンドを実行します。まだ実行されていないジョブには、新しいパスワードを使用します。

#### 1.4.3.6.i Windows クライアントでのパスワードの設定

ジョブを投入し、PBS クライアントコマンドを実行できるようにするには、Windows の各投入ホストで 1 回ずつ pbs_login コマンドを実行します。

```
echo <パスワード>| pbs_login -p
```

クライアントコマンドを実行できるかどうかテストします。

```
qstat -Bf
```

まだ実行されていないジョブには、新しいパスワードを使用します。

#### 1.4.3.6.ii Linuxクライアントでのパスワードの設定

Windowsジョブを投入するLinuxクライアントホストでpbs_loginコマンドを実行します。PBS_AUTH_METHODを*pwd*に設定します。

```
export PBS_AUTH_METHOD=pwd; pbs_login
```

### 1.4.3.7 クライアントコマンドの認証

認証方法としてpwdまたはmungeを使用して、qsubを除くすべてのクライアントコマンドを実行できます。このため、qstatなどのコマンドで変更を行う必要はありません。

## 1.4.4 投入ホストのタイムゾーンの設定

投入ホストでは、環境変数PBS_TZIDが設定されていることを確認します。この環境変数をPBS Professionalで認識されているタイムゾーンの場所に設定します。適切なゾーンの場所は、PBSサーバーホストから取得できます。

Linuxでは、tzselectコマンドを使用するか（使用可能な場合）、または/usr/share/zoneinfo/zone.tabからゾーンの場所を取得します。

その他のすべてのプラットフォームでは、$PBS_EXEC/lib/ical/zoneinfo/zones.tabの下で使用できるlibicalサポートのゾーン情報ロケーションのリストを使用します。

PBS_TZIDの形式はタイムゾーンの場所になります。タイムゾーンのPOSIX省略形ではありません。PBS_TZIDの値の例を以下に示します。

```
America/Los_Angeles
America/Detroit
Europe/Berlin
Asia/Calcutta
```

# 2
# PBS ジョブの投入

## 2.1 PBS ジョブの概要

PBS を使用するには、*バッチジョブ*（通常は単に *ジョブ* と呼びます）を作成し、これを PBS に渡します（または *投入* します）。バッチジョブは、1 つまたは複数の実行マシン上で実行する一連のコマンドやアプリケーションであり、ファイルに記述するか、またはコマンドラインで入力します。特性を指定する命令を記述できます。このような特性として、ジョブで必要とするリソース要件（メモリ、CPU 時間など）やジョブ名などがあります。ジョブファイルは、Linux のシェルスクリプト、Windows の `cmd` バッチファイル、Python スクリプト、Perl スクリプトなどで作成できます。

時間を 1 時間、メモリを 400MB、CPU を 4 つ要求し、`my_application` を実行する、単純な PBS バッチジョブファイルの例を以下に示します。

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
./my_application
```

このジョブを PBS に投入するには、`qsub` コマンドを使用します。ジョブスクリプトは、`qsub` への引数として指定します。“`my_script`”という名前のスクリプトを投入する例を以下に示します。

```
qsub my_script
```

ジョブスクリプトの作成の詳細については14 ページの第 2.2 節「PBS ジョブスクリプト」を、ジョブ投入の詳細については20 ページの第 2.3 節「PBS ジョブの投入」を、それぞれご参照ください。

## 2.1.1 PBSジョブのライフサイクルの概要

PBSジョブのライフサイクルを以下に示します。

1. ジョブスクリプトを作成します。
2. ジョブをPBSに投入します。
3. PBSはジョブを受け取り、ジョブIDを投入者に返します。
4. PBSスケジューラは、ジョブを実行する適切な場所と時間を見つけて、選択した実行ホスト（複数可）にジョブを送信します。
5. アプリケーションライセンスがチェックアウトされます。
6. PBSにより、各実行ホスト上（指定されている場合）で、ジョブ固有のステージングと実行のディレクトリが作成されます。
7. PBS_JOBDIRとジョブのjobdir属性がジョブのステージングと実行のディレクトリのパスに設定されます。
8. ジョブに割り当てられている各実行ホスト上で、一時スクラッチディレクトリが作成されます。
9. TMPDIR環境変数は一時ディレクトリのパス名に設定されます。
10. ディレクトリの作成時または変数の設定時に何らかのエラーが発生すると、ジョブは再キューイングされます。
11. 入力ファイルまたは入力ディレクトリがプライマリ実行ホストにコピーされます。

- prologueが存在する場合、プライマリ実行ホスト上で実行されます。その際、`PBS_HOME/mom_priv`をカレントディレクトリとし、PBS_JOBDIRおよびTMPDIRは環境変数として設定されます。

12. ジョブがプライマリ実行ホスト上でユーザーとして実行されます。
13. ジョブの関連するタスクが実行ホスト上でユーザーとして実行されます。
14. epilogueが存在する場合、プライマリ実行ホスト上で実行されます。その際、ジョブのステージングと実行のディレクトリをカレントディレクトリとし、PBS_JOBDIRおよびTMPDIRは環境変数として設定されます。
15. 出力ファイルまたは出力ディレクトリが指定された場所にコピーされます。
16. 一時ファイルおよび一時ディレクトリがクリーンアップされます。
17. アプリケーションライセンスがプールに戻されます。

ジョブのライフサイクルの詳細については、42ページの第3.2.8節「ジョブのライフサイクルの詳細」をご参照ください。

## 2.1.2 PBSジョブが動作する場所としくみ

PBSジョブは、管理者がPBSに対して実行ホストとして指定したホスト上で動作します。PBSスケジューラは、ジョブに必要なリソースを持つ１つまたは複数の実行ホストを選択します。

PBSは、ユーザーのジョブをそのユーザーアカウントで実行します。これは、各ジョブでユーザーのログインファイルとログアウトファイルが実行され、ユーザーの環境の一部がジョブに渡されることを意味します。ログインファイルとログアウトファイルがジョブの実行の妨げにならないようにすることが重要です。5ページの第1.4.2節「Linux環境の設定」をご参照ください。

## 2.1.3 ジョブ識別子

ジョブを投入すると、PBSは*ジョブ識別子*を返します。ジョブの場合は以下の形式になります。

*<シーケンス番号>.<サーバー名>*

ジョブアレイの場合は以下の形式になります。

*<シーケンス番号>[].<サーバー名>.<ドメイン>*

ジョブステータスの確認、ジョブの修正、ジョブの追跡、ジョブの削除など、ジョブに対して何らかの処理を行う場合は、このジョブ識別子を指定する必要があります。

最大となるジョブIDの上限はデフォルトで7桁の9,999,999ですが、これより大きい値を管理者が設定できます。最大のジョブIDが割り当てられると、以降のジョブには再びゼロから始まるIDが割り当てられます。

## 2.1.4 ジョブのシェルスクリプト

PBSは、ユーザーのジョブを実行する場合、ユーザーがジョブに指定したトップシェルを起動します。トップシェルはデフォルトでは実行ホスト上のユーザーのログインシェルですが、ジョブのShell_Path_List属性を使用して別のシェルに設定できます。21ページの第2.3.3.1節「ジョブのトップシェルの指定」をご参照ください。

Linuxでは、ユーザーがジョブスクリプトでシェルを指定しない場合、デフォルトで`/bin/sh`が使用されます。ユーザーがジョブスクリプト内で別のシェルを指定している場合、トップシェルがそのシェルを生成してスクリプトを実行します。21ページの第2.3.3.2節「ジョブスクリプトのシェルまたはインタープリタの指定」をご参照ください。

Windowsでは、ジョブシェルはトップシェルと同じです。

## 2.1.5 ジョブのスクラッチスペース

PBSは、ユーザーのジョブを実行する場合、各実行ホスト上にジョブの一時スクラッチディレクトリを作成します。管理者は、$tmpdir MoMパラメータを使用して、各実行ホスト上の一時ディレクトリのルートを指定できます。

このディレクトリは、ジョブの完了時に削除されます。一時ディレクトリの場所はPBSによって設定されるので、TMPDIRを指定しないでください。

ジョブスクリプトは、スクラッチスペースにアクセスできます。以下に例を示します。

Linux：

```
cd $TMPDIR
```

Windows：

```
cd %TMPDIR%
```

MPIジョブのスクラッチスペースについては、88ページの第5.2.3節「MPI使用の注意事項」をご参照ください。

### 2.1.5.1 Linuxにおける一時スクラッチスペースの場所

管理者が一時ディレクトリを指定していない場合、`/var/tmp`が一時ディレクトリのルートになります。TMPDIR環境変数は一時スクラッチディレクトリのフルパスに設定されます。

### 2.1.5.2 Windowsにおける一時スクラッチスペースの場所

Windowsでは、PBSは、一時ディレクトリを作成して、TMPをWindowsの%TMPDIR%環境変数の値に設定します。管理者が一時ディレクトリを指定していない場合、PBSは、`¥winnt¥temp`または`¥windows¥temp`のどちらかの下に一時ディレクトリを作成します。

### 2.1.6 ジョブのタイプ

PBSでは、標準バッチジョブまたは*インタラクティブ*ジョブを投入できます。異なるのは、インタラクティブジョブが動作している間はインタラクティブセッションが動作して、ユーザーがジョブプロセスにインタラクティブにアクセスできます。標準バッチジョブにはインタラクティブにアクセスできません。インタラクティブジョブについては、129ページの第6.11節「ジョブのインタラクティブな実行」をご参照ください。

### 2.1.7 ジョブの入力ファイルと出力ファイル

アクセス可能な任意の場所にあるファイルまたはディレクトリを実行ホストにコピーすること、および実行ホストから出力ファイルおよび出力ディレクトリを指定した場所にコピーすることをPBSに指示できます。この方法については、35ページの第3章「ジョブの入力ファイルと出力ファイル」をご参照ください。

## 2.2 PBSジョブスクリプト

### 2.2.1 ジョブスクリプトの概要

PBSジョブスクリプトは、下記のもので構成されています。

- オプションのシェル指定
- PBSの指示文
- ジョブタスク（プログラムまたはコマンド）

### 2.2.2 ジョブスクリプトの種類

PBSでは、ジョブスクリプトとして以下を使用できます。

- WindowsまたはLinuxで実行可能なPython、Perlなどのスクリプト
- Linuxで実行するシェルスクリプト
- Windows環境のコマンドまたはPowerShellバッチスクリプト

#### 2.2.2.1 Linuxシェルスクリプト

ジョブファイルはシェルスクリプト形式にできるため、シェルスクリプト形式のジョブファイルの先頭行で、スクリプトの実行に使用するシェルを指定します。ユーザーのログインシェルがデフォルトですが、変更することが可能です。ログインシェルを使用してジョブファイルを解釈できる場合は、この先頭行は省略可能です。常にシェルを指定することをお勧めします。

#### 2.2.2.2 Pythonジョブスクリプト

PBSでは、WindowsまたはLinuxでPythonスクリプトを使用してジョブを投入できます。PBSにはPythonパッケージが含まれているため、Pythonジョブスクリプトを実行できます。Pythonをインストールする必要はありません。Pythonジョブスクリプトの実行方法を以下に示します。

Linux：

```
qsub <スクリプト名>
```

Windows：

**qsub -S %PBS_EXEC%¥bin¥pbs_python.exe <スクリプト名>**

パスに空白が含まれている場合は、次のように引用符で囲む必要があります。

**qsub -S "%PBS_EXEC%¥bin¥pbs_python.exe" <python ジョブスクリプト>**

Linux シェルスクリプトと同様に、Python ジョブスクリプトには PBS の指示文を含めることができます。以下に例を示します。

```
% cat myjob.py
#!/usr/bin/python
#PBS -l select=1:ncpus=3:mem=1gb
#PBS -N HelloJob
print "Hello"
```

Python ジョブスクリプトでは Win32 API にアクセスできます。この API には以下のモジュールが含まれます。

- Win32api
- Win32con
- Pywintypes

### 2.2.2.2.i Python ジョブスクリプトのデバッグ

PBS の外部で Python をインタラクティブに実行し、Python ジョブスクリプトをデバッグできます。スクリプトの一部をテストするには、Python インタープリタを使用します。

Linux では、pbs_python コマンドで-i オプションを使用します。以下に例を示します。

**/opt/pbs/bin/pbs_python -i <return>**

Windows では -i オプションは必要ありませんが、使用することは可能です。たとえば、次のいずれでも機能します。

**C:¥Program Files¥PBS¥exec¥bin¥pbs_python.exe <return>**

**C:¥Program Files¥PBS¥exec¥bin¥pbs_python.exe -i <return>**

Python インタープリタが実行されている場合は、その独自のプロンプトが表示されます。以下に例を示します。

```
% /opt/pbs/bin/pbs_python -i <return>
>> print "hello"
hello
```

### 2.2.2.2.ii Python Windows の注意事項

- Python がネイティブにインストールされていて、win32api を使用する必要がある場合は、必ず win32api の前に pywintypes をインポートしてください。これを行わないと、エラーが発生します。以下の内容を実行します。

  ```
  cmd> pbs_python
  >> import pywintypes
  >> import win32api
  ```

- Windows のジョブを投入するときは、必ずアーキテクチャとして Windows を指定します。ジョブで必要なリソースを記述する -l select の作成では、チャンクごとにアーキテクチャを指定します。-l select については 53 ページの第 4 章「ジョブへのリソースの 割り当てとジョブの配置」をご参照ください。以下に例を示します。

  ```
  #PBS -l select=ncpus=2:mem=1gb:arch=windows
  ```

  “windows”では大文字と小文字が区別されます。

## 2.2.2.3 Windowsジョブスクリプト

Windowsスクリプトとして、.exeファイルまたは.batファイル、またはPythonスクリプトまたはPerlスクリプトを使用できます。

### 2.2.2.3.i Windowsコマンドスクリプトの要件

- Windowsのジョブを投入するときは、必ずアーキテクチャとしてWindowsを指定します。ジョブで必要なリソースを記述する-l selectの作成では、チャンクごとにアーキテクチャを指定します。-l selectについては53ページの第4章「ジョブへのリソースの 割り当てとジョブの配置」をご参照ください。以下に例を示します。

  ```
  #PBS -l select=ncpus=2:mem=1gb:arch=windows
  ```

  “windows”では大文字と小文字が区別されます。
- Windowsでは、ジョブスクリプトのコメントはASCII文字にする必要があります。
- PBSのジョブスクリプト内で実行される.batファイルは､次のように"call"という接頭辞を付ける必要があります。

  ```
  @echo off
  call E:¥step1.bat
  call E:¥step2.bat
  ```

  “call”がない場合、最初の.batファイルのみが実行され、制御が呼び出し元のインタープリタに戻されません。

  たとえば、次のような内容のジョブスクリプトは、

  ```
  @echo off
  E:¥step1.bat
  E:¥step2.bat
  ```

  以下の内容にする必要があります。

  ```
  @echo off
  call E:¥step1.bat
  call E:¥step2.bat
  ```

### 2.2.2.3.ii Windowsのアドバイスと注意事項

- Windowsでは、ジョブスクリプトの作成にnotepadを使用している場合、最後の行は自動的には改行されません。明示的に改行を追加しないと、PBSジョブは以下のエラーメッセージ、

  ```
  More?
  ```

  が、Windowsコマンドインタープリタがその最終行を実行しようとしたときに表示されます。
- ドライブマッピングコマンドは、通常はジョブスクリプトで記述します。
- ジョブスクリプト内でxcopyは使用しないでください。代わりに、copy、robocopy、またはpbs_rcpを使用してください。xcopyコマンドは、ユーザーに入力を要求する場合があります。そのため、入力ハンドルを割り当てる必要があります。PBSは、入力ハンドルが割り当てられたジョブプロセスを作成しないので、xcopyをPBSジョブスクリプト内で使用すると、失敗したり、動作が異常になる可能性があります。
- cygwinから投入されたPBSジョブは、ネイティブのcmd環境で実行され、cygwin環境では実行されません。

## 2.2.3 ジョブ特性の設定

### 2.2.3.1 ジョブの属性

PBS では、ジョブの特性が*属性*として表されます。たとえば、ジョブの名前はそのジョブの属性であり、ジョブの Job_Name 属性の値として格納されます。ジョブの属性には、ユーザーが設定できる属性や、管理者のみが設定できる属性があります。また、PBS のみが設定する属性もあります。PBS のジョブの属性の完全なリストについては、『PBS Professional Reference Guide』の「Job Attributes」（327 ページ）をご参照ください。ジョブ属性は大文字と小文字が区別されません。

### 2.2.3.2 ジョブのリソース

PBS では、ジョブが使用する可能性があるものが*リソース*として表されます。たとえば、実行ホスト上の CPU の数とメモリの量は、リソースです。PBS では複数の組み込みリソースが提供されており、PBS 管理者がリソースを定義することもできます。すべての組み込み PBS リソースのリストについては、259 ページの第5章「List of Built-in Resources」をご参照ください。リソースは大文字と小文字が区別されません。

### 2.2.3.3 ジョブの属性の設定

次の同等な方法を使用して、ジョブの属性を設定したり、リソースを要求したりできます。

- コマンドラインで qsub コマンドに個々のオプションを指定します。たとえば、エラーパスを設定するには、-e <パス> と指定します。
- ジョブスクリプトで PBS の指示文を使用します。たとえば、エラーパスを設定するには、#PBS -WError_Path=<パス> と指定します。

これらの方法は、同じ機能を有しています。qsub に競合するオプションを指定した場合は、最後に指定したオプションが他のオプションより優先されます。qsub コマンドへのオプションは、PBS の指示文よりも優先され、またデフォルトの設定よりも優先されます。ジョブの属性やリソースには、デフォルト値を持つものがあります。管理者は、一部の属性やリソースのデフォルト値を設定できます。

ジョブの投入後、qalter コマンドを使用してジョブの特性を変更できます。

### 2.2.3.4 PBSの指示文の使用

PBSの指示文を使用してジョブの各種属性値を設定できます。指示文には、最初の空白以外の文字として指示文プレフィックスを指定します。このプレフィックスのデフォルトは*#PBS*です。

PBS指示文はすべて、スクリプトファイルの一番上に、どのコマンドよりも先に記述します。スクリプトの実行可能行より後に記述されている指示文は無視されます。たとえば、スクリプトで"@echo"と記述する場合、この行はすべてのPBS指示文より後に記述します。

#### 2.2.3.4.i 指示文プレフィックスの変更

デフォルトでは、文字列"*#PBS*"は、ジョブファイル内のどの行がPBS指示文であるかを判断するために、PBSによって使用されます。先頭の記号"#"は、Linuxシステム上で共通して使用されるすべてのシェルスクリプト記述言語へのコメント区切り記号であるため、選択されています。指示文はコメントと解釈され、スクリプト言語では無視されます。

ただし、Windowsでは、コマンドインタープリタが'#'記号をコメントとして認識せず、"*#PBS*"文字列に遭遇するたびに、無害で重要ではない警告が生成されます。これはバッチジョブについては問題を生じないものの、ユーザーにとっては煩わしいものです。Windowsを使用する場合、PBS_DPREFIX環境変数、またはqsubの"-C"オプションのいずれかを使用して、異なるPBS指示文を指定することをお勧めします。qsubオプションは、環境変数より優先されます。たとえば、"*#PBS*"に代わって文字列"*REM PBS*"を使用し、下記のジョブスクリプト内にこの指示文を使用することが可能です。

```
REM PBS -l walltime=1:00:00
REM PBS -l select=mem=400mb:arch=windows
REM PBS -j oe
date /t
.¥my_application
date /t
```

上記のジョブスクリプトの場合、以下の２つの方法のいずれかでPBSに投入できます。

```
set PBS_DPREFIX=REM PBS
qsub my_job_script
```

または

```
qsub -C "REM PBS" my_job_script
```

#### 2.2.3.4.ii PBSの指示文の注意事項と制限事項

- *PBS_DPREFIX*を指示文プレフィックスとして使用することはできません。
- 指示文文字列の長さ制限は4,096文字です。

## 2.2.4 ジョブタスク

ユーザータスクは、プログラムまたはコマンドです。これは、実行すべきアプリケーションをユーザーが指定するところです。

## 2.2.5 ジョブスクリプト名

ジョブスクリプト名には特殊文字を使用しないことをお勧めします。使用する必要がある場合は、Linuxではバックスラッシュ（"¥"）文字を使用して特殊文字をエスケープする必要があります。

### 2.2.5.1 PBSによるジョブスクリプトの解析

PBSは、ジョブスクリプトを前半と後半に分けて解析します。最初に、qsubコマンドがスクリプトをスキャンしながら指示文を探して、最初に見つかった実行可能行の位置でスキャンを終了します。このことは、qsubに指示文を処理させる場合は、指示文をどの実行可能行より先に記述する必要があることを意味します。最初の実行可能行より後に記述されている指示文は無視されます。最初の実行可能行は指示文ではない最初の行であり、その最初の空白以外の文字は"#"でも空白文字でもありません。指示文については、18ページの第2.2.3.4節「PBSの指示文の使用」をご参照ください。

次に、ジョブシェルがスクリプトの行を処理します。PBSがジョブスクリプトファイルの名前をパイプでトップシェルの入力として渡すと、トップシェルはジョブシェルを実行してスクリプトを実行します。トップシェルとして使用するシェルを指定できます（21ページの第2.3.3.1節「ジョブのトップシェルの指定」をご参照ください）。また、Linuxでは、スクリプトの最初の実行可能行で、スクリプトを実行させるシェルを指定できます（21ページの第2.3.3.2節「ジョブスクリプトのシェルまたはインタープリタの指定」をご参照ください）。

#### 2.2.5.1.i 同等のLinuxジョブスクリプトとWindowsジョブスクリプトの比較

以下に示すLinuxジョブスクリプトとWindowsジョブスクリプトは同じ結果をもたらします。

Linux：

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l select=mem=400mb
#PBS -j oe

date
./my_application
date
```

Windows：

```
REM PBS -l walltime=1:00:00
REM PBS -l select=mem=400mb:arch=windows
REM PBS -j oe

date /t
my_application
date /t
```

Windowsスクリプトの先頭行では、シェルへのパスが記述されていません。これは、Windowsジョブスクリプト内では、シェルまたはインタープリタへのパスを指定できないためです。21ページの第2.3.3.2節「ジョブスクリプトのシェルまたはインタープリタの指定」をご参照ください。

両方のファイルの残りの行はほとんど同じです。主な相違点は、ファイル内でディレクトリパスの指定（ドライブ文字、パスの区切り文字としてスラッシュ、バックスラッシュの使用など）です。

"#PBS"および"REM PBS"で始まる行は、PBS指示文です。PBSはジョブスクリプトを上から順に読み取り、有効なPBS指示でない最初の行を検出した時点で読み取りを終了します。次に、ジョブシェルまたはインタープリタがスクリプトの行を読み取ります。上記の例では、６行目から８行目までがジョブシェルで実行するコマンドです。

上記の例では、"-l <リソース>=<値>"の行で特定のリソースを要求しています。ここでは、ジョブワイドの要求として１時間の経過時間を要求し、400MBのメモリをチャンク内で要求しています。これがWindowsのジョブである場合は、チャンクの記述に":arch=windows"を追記します。リソースの要求については、53ページの第4章「ジョブへのリソースの 割り当てとジョブの配置」をご参照ください。

"-j oe"の行では、ジョブのstdoutとstderrの出力ストリームを１つのストリームに*結合*するよう要求しています。出力の結合については、48ページの「出力ファイルとエラーファイルの結合」をご参照ください。

最後の３行は、アプリケーションを実行するためのコマンドラインです。必要に応じて、希望する数のアプリケーション、タスク、またはジョブステップを指定することができます。

# 2.3 PBSジョブの投入

## 2.3.1 ジョブ投入の前提条件

ジョブを投入する前に、環境を適切に設定します。5ページの第1.4節「環境の設定」の手順に従ってください。

## 2.3.2 PBSジョブを投入する方法

以下の方法でqsubコマンドを使用して、通常のジョブまたはインタラクティブジョブをPBSに投入できます。

- ジョブスクリプトでqsubを呼び出すことができます。20ページの第2.3.3節「スクリプトによるジョブの投入」をご参照ください。
- 実行可能ファイルとその引数を使用してqsubを呼び出すことができます。23ページの第2.3.4節「コマンドラインで実行プログラムを指定してジョブを投入する方法」をご参照ください。
- qsubを呼び出して、キーボードからの入力を渡すことができます。24ページの第2.3.5節「キーボード入力を使用してジョブを投入する方法」をご参照ください。

Altairフロントエンド製品を使用して、ジョブを投入し、監視できます。www.pbsworks.comをご参照ください。

## 2.3.3 スクリプトによるジョブの投入

PBSにジョブを投入するには、qsubコマンドを使用します。qsubの詳細については、『PBS Professional Reference Guide』の「qsub」（216ページ）をご参照ください。PBSジョブを投入するには、次のように入力します。

- Linuxシェルスクリプト：

  **qsub <シェルスクリプト名>**
- LinuxのPythonまたはPerlのスクリプト：

  **qsub <PythonまたはPerlのジョブスクリプト名>**
- Windowsコマンドスクリプト：

  **qsub <ジョブスクリプト名>**
- WindowsでのPythonスクリプト：

  **qsub -S %PBS_EXEC%¥bin¥pbs_python.exe <Pythonジョブスクリプト名>**

  パスに空白が含まれている場合は、次のように引用符で囲む必要があります。

  **qsub -S "%PBS_EXEC%¥bin¥pbs_python.exe" <Pythonジョブスクリプト名>**

### 2.3.3.1 ジョブのトップシェルの指定

ジョブのトップシェルとして使用するシェルのパスと名前を指定できます。トップシェルを指定する場合のルールは、LinuxとWindowsでは異なります。この後に続く2.3.3.1.i節と2.3.3.1.ii節はスキップしないでください。

Shell_Path_Listジョブ属性は、トップシェルを指定します。デフォルトは、実行ホスト上のユーザーのログインシェルです。この属性は、以下のように指定できます。

- qsubの"-S <パスリスト>"オプションを指定する
- PBS指示文#PBS -WShell_Path_List=<パスリスト>を指定する

オプションの引数の*パスリスト*は、以下の形式で指定します。

*<パス>[@<ホスト名>][,<パス>[@<ホスト名>],...]*

Shell_Path_Listを設定する場合は、*パスリスト*を指定する必要があり、指定しない場合はエラーが返されます。指定するホスト1つにつきパスを1つのみ指定できます。対応するホストがないパスを1つのみ指定できます。

PBSは、実行ホストの名前と一致するホスト名に対応するパスを選択します。一致するホストが見つからない場合は、ホストなしで指定されているパスがあれば、それを選択します。

#### 2.3.3.1.i Linuxでのジョブのトップシェルの指定

Linuxでは、トップシェルはMoMがジョブを起動するときに起動するシェルであり、ジョブシェルはジョブスクリプトのコマンドを実行するシェルまたはインタープリタです。

Linuxでは、qsub -S <パス>と指定することによって、cshやshなどの任意のシェルを使用できます。トップシェルとしてPerlまたはPythonを使用することはできません。

例2-1： bashを使用する場合：

```
qsub -S /bin/bash <スクリプト名>
```

#### 2.3.3.1.ii Windowsでのジョブのトップシェルの指定

Windowsでは、ジョブシェルはトップシェルと同じです。

Windowsでは、PerlやPythonなどのシェルまたはインタープリタを指定でき、ジョブスクリプトがPerlまたはPythonの場合はqsubのオプションを使用して言語を指定する必要があります。ジョブスクリプト内では指定できません。

例2-2： WindowsでPythonスクリプトを実行する場合：

```
qsub -S "C:¥Program Files¥PBS¥exec¥bin¥pbs_python.exe" <スクリプト名>
```

#### 2.3.3.1.iii ジョブのトップシェルを指定する場合の注意事項

トップシェルの相対パスを指定する場合、そのフルパスが実行ホストのPATH環境変数に存在する必要があります。フルパスを指定することをお勧めします。

### 2.3.3.2 ジョブスクリプトのシェルまたはインタープリタの指定

#### 2.3.3.2.i Linuxでのジョブスクリプトのシェルまたはインタープリタの指定

ジョブスクリプトのシェルを指定していない場合、デフォルトで/bin/shが使用されます。任意のシェルを使用できるほか、PerlやPythonなどのインタープリタも使用できます。

シェルまたはインタープリタは、ジョブスクリプトの最初の行で指定します。トップシェルが指定されたプロセスを生成し、このプロセスがジョブスクリプトを実行します。たとえば、/bin/shを使用してスクリプトを実行するには、ジョブスクリプトの最初の行で以下のように指定します。

```
#!/bin/sh
```

PerlまたはPythonを使用してスクリプトを実行するには、スクリプトの最初の行で以下のようにPerlまたはPythonへのパスを指定します。

```
#!/usr/bin/perl
```

または

```
#!/usr/bin/python
```

#### 2.3.3.2.ii Windowsでのジョブスクリプトのシェルまたはインタープリタの指定

Windowsでは、ジョブシェルまたはジョブインタープリタは、トップシェルまたはトップインタープリタと同じです。トップ/ジョブシェルまたはトップ/ジョブインタープリタを指定できますが、ジョブシェルまたはジョブインタープリタを別途指定することはできません。デフォルト以外のシェルまたはインタープリタを使用するには、qsubのオプションを使用して指定する必要があります。

*qsub -S <シェルまたはインタープリタへのパス> <スクリプト名>*

### 2.3.3.3 スクリプトによるジョブ投入の例

例2-3： ジョブスクリプトの名前は“myjob”です。以下のように入力してこれを投入できます。

**qsub myjob**

PBSから以下のようなジョブIDが返されます。

```
16387.exampleserver.exampledomain
```

例2-4： 以下に“myjob”という名前のスクリプトの内容を示します。ここでは、ジョブ“testjob”を指定し、“myprogram”という名前のプログラムを実行します。

```
#!/bin/sh
#PBS -N testjob
./myprogram
```

例2-5： ジョブを投入する最も簡単な方法は、qsubの引数としてスクリプト名を指定してReturnキーを押すことです。

**qsub <ジョブスクリプト> <return>**

スクリプトに以下のように記述する場合

```
#!/bin/sh
./myapplication
```

単にmyapplicationを実行するようPBSに指示しています。

### 2.3.3.4 ジョブへの引数の引き渡し

ジョブスクリプトに引数を渡す必要がある場合、以下の方法で実行できます。

- スクリプトで環境変数を使用して、この環境変数に-vまたは-Vを使用して値を渡します。

  たとえば、a.outへの入力としてmyinfileを使用するには、ジョブスクリプトで以下のように記述します。

  ```
  #PBS -N myjobname
  a.out < $INFILE
  ```

  この場合、以下のように、-Vオプションを使用できます。

  **qsub -v INFILE=/tmp/myinfile <ジョブスクリプト>**

たとえば、a.out への入力として myinfile と mydata を使用するには、ジョブスクリプトで以下のように記述します。

```
#PBS -N myjobname
cat $INFILE $INDATA | a.out
```

この場合、以下のように、-v オプションを使用できます。

```
qsub -v INFILE=/tmp/myinfile, INDATA=/tmp/mydata <ジョブスクリプト>
```

先に環境変数をエクスポートすることもできます。

```
export INFILE=/tmp/myinfile
qsub -V <ジョブスクリプト>
```

- ヒアドキュメントを使用します。以下に例を示します。

```
qsub [オプション] [オプション] ...<return>
#PBS <指示文>
./jobscript.sh arg1      <^d>
152.examplehost
```

ジョブに引数を渡す必要がある場合、以下の方法で実行できます。

- シェルコマンドをパイプで qsub に渡します。

  たとえば、a.out への入力として myinfile と mydata を直接渡すには、以下のように入力するか、シェルスクリプトに記述します。

  ```
  echo "a.out myinfile mydata" | qsub -l select=...
  ```

  以下に例を示します。

  ```
  echo "jobscript.sh -a arg1 -b arg2" | qsub -l select=...
  ```

  たとえば、a.out への入力として myinfile を環境変数を使用して渡すには、以下のように入力するか、シェルスクリプトに記述します。

  ```
  export INFILE=/tmp/myinfile
  export INDATA=/tmp/mydata
  echo "a.out $INFILE $INDATA" | qsub
  ```

- qsub --<実行プログラム> <実行プログラムへの引数> を使用します。23 ページの第2.3.4 節「コマンドラインで実行プログラムを指定してジョブを投入する方法」をご参照ください。

## 2.3.4 コマンドラインで実行プログラムを指定してジョブを投入する方法

ジョブスクリプトの代わりに実行プログラムと引数を指定することにより、PBS ジョブを実行できます。このようにして qsub を実行すると、これは直接*実行プログラム*を実行します。これはシェルを起動せず、シェルなしの初期化スクリプトが実行され、実行パスとその他の環境変数は設定されません。異なるディレクトリでコマンドを実行する簡単な方法はありません。環境変数が正しく設定されていることを確認し、通常はコマンドへのフルパスを指定する必要があります。

直接ジョブを投入するには、コマンドラインで実行プログラムを指定します。

*qsub [<オプション>] -- <実行プログラム> [<実行プログラムへの引数>] <return>*

たとえば、myprog に引数 a および b を渡して実行するには、次のように入力します。

```
qsub -- myprog a b <return>
```

たとえば、myprog に引数 a および b を渡して実行し、ジョブを *JobA* と命名するには、次のように入力します。

```
qsub -N JobA -- myprog a b <return>
```

すでに定義されている環境変数を使用するには、次のように入力します。

```
export INFILE=/tmp/myinfile
export INDATA=/tmp/mydata
qsub -- a.out $INFILE $INDATA
```

## 2.3.5 キーボード入力を使用してジョブを投入する方法

qsubがキーボードから入力を読み出すように指定することも可能です。コマンドラインでリソースを要求してqsubコマンドを実行し、ジョブファイルを命名せずにEnterを押すと、PBSはキーボードから入力を読み出します（これは通常、“ヒアドキュメント”と呼ばれています）。ユーザーは、行自体にcontrol-dと入力するか（Linuxの場合）、control-zに続いてenterと入力する（Windowsの場合）ことによって、入力の読み出しを停止してジョブを投入するようqsubに指示できます。ダッシュのオペランドはあってもなくても同じ動作を得られます。

Linuxでは、qsubが入力を読み出し中にユーザーがcontrol-cを入力すると、qsubはそのプロセスを終了し、ジョブは投入されません。Windowsでは、使用されるコマンドプロンプトによっては、control-cシーケンスがqsubにジョブをPBSに投入させることがしばしばあります。そのような場合、control-breakシーケンスが通常qsubコマンドを終了します。

```
qsub [<オプション>] [-] <return>
    [<指示文>]
    [<タスク>]
    ctrl-D
```

## 2.3.6 Windowsジョブの投入

PBSコンプレックスは、実行およびクライアント（投入）ホストがすべてWindowsの場合もあれば、LinuxとWindowsが混在している場合もあります。コンプレックスに各実行ホストの一部がある場合、Linuxクライアントから投入するのかWindowsクライアントから投入するのかに関係なく、Windowsジョブが Windows実行ホスト上に配置されることを確認します。

### 2.3.6.1 WindowsクライアントからのWindowsジョブの投入

- 最初とパスワードが変更されるたびに、各投入ホストでpbs_loginコマンドを実行する必要があります。

  ```
  echo <●●●●●>| pbs_login -p
  ```

- Windowsクライアントから Windowsジョブを投入する場合は、必ずWindows実行ホストを要求します。archリソースを“windows”に設定するよう要求します。

  ```
  qsub -lselect=1:arch=windows
  ```

  “windows”では大文字と小文字が区別されます。

### 2.3.6.2 LinuxクライアントからのWindowsジョブの投入

- Windowsジョブを投入するLinuxクライアントホストでpbs_loginコマンドを実行します（まだ実行していない場合）。PBS_AUTH_METHODを*pwd*に設定します。

  ```
  export PBS_AUTH_METHOD=pwd; pbs_login
  ```

- Linux クライアントから Windows ジョブを投入するには、アーキテクチャが Windows であることを指定します。“arch=windows”では大文字と小文字が区別されます。以下に例を示します。

  ```
  export PBS_AUTH_METHOD=pwd; qsub -lselect=1:arch=windows -- pbs-sleep 100
  ```

### 2.3.6.3 Linux クライアントからの Windows ジョブと Linux ジョブの投入

Linux クライアントから Windows ジョブと Linux ジョブの両方を投入できますが、ジョブの種類ごとに認証方法を正しく設定する必要があります。たとえば、以下に示すように、MUNGE 認証を使用して Linux ジョブを投入した後、認証方法を *pwd* に設定して Windows ジョブを投入できます。

```
export PBS_AUTH_METHOD=munge; qsub -lselect=1:arch=linux -- pbs-sleep 100
export PBS_AUTH_METHOD=pwd; pbs_login
qsub -lselect=1:arch=windows -- pbs-sleep 100
```

PBS_AUTH_METHOD 設定パラメータの値を変更するには、PBS_AUTH_METHOD 環境変数で認証方法を設定します。これはプロファイルで設定できます。

# 2.4 ジョブ投入の推奨事項とアドバイス

## 2.4.1 スクリプトでのシグナルのトラップ

ジョブスクリプトでシグナルをトラップできます。たとえば、優先実行シグナルと中断シグナルをトラップできます。

ジョブスクリプトでシグナルをトラップする場合、シグナルによっては、ジョブのすべてのシェルでトラップする必要がある場合があります。

TERM は便利なシグナルです。このシグナルは、シェルには無視されますが、トラップしてステータスの書き出しなどの便利な処理を実行できます。

例2-6： リストされているシグナルを無視します。

```
trap "" 1 2 3 15
```

例2-7： リストされているシグナルを検出したら関数“goodbye”を呼び出します。

```
trap goodbye 1 2 3 15
```

# 2.5 ジョブ投入オプション

以下の表に、qsub コマンドのオプションとその説明の参照先をリストします。

**表2-1：qsub コマンドのオプション**

| オプション | 機能および参照項目 |
|---|---|
| -A <アカウント文字列> | 31ページの「アカウンティング文字列の指定」 |
| -a <日時> | 127ページの「実行の延期」 |
| -C "<指示文接頭辞>" | 18ページの「指示文プレフィックスの変更」 |
| -c <間隔> | 120ページの「チェックポイント処理の使用」 |
| -e <パス> | 46ページの「出力ファイルとエラーファイルのパス」 |
| -f | 32ページの「フォアグラウンドでの qsub の実行」 |
| -G | 133ページの「Windows でのインタラクティブ GUI ジョブの投入」 |
| -h | 122ページの「ジョブの保留/保留解除」 |

## 表2-1：qsubコマンドのオプション

| オプション | 機能および参照項目 |
|---|---|
| -I | 129ページの「ジョブのインタラクティブな実行」 |
| -J X-Y[:Z] | 162ページの「ジョブアレイの投入」 |
| -j <結合> | 48ページの「出力ファイルとエラーファイルの結合」 |
| -k <保持> | 48ページの「実行ホストでの出力ファイルとエラーファイルの保存」 |
| -l <リソースリスト> | 56ページの「リソースの要求」 |
| -M <ユーザーリスト> | 28ページの「電子メール受信者リストの設定」 |
| -m <メールオプション> | 27ページの「電子メール通知の指定」 |
| -N <名前> | 29ページの「ジョブ名の指定」 |
| -o <パス> | 46ページの「出力ファイルとエラーファイルのパス」 |
| -p <優先度> | 128ページの「ジョブの優先度の設定」 |
| -P <プロジェクト> | 29ページの「ジョブのプロジェクトの指定」 |
| -q <宛先> | 31ページの「サーバーまたはキュー、あるいはその両方の指定」 |
| -r <値> | 126ページの「ジョブの再実行の許可」 |
| -R <削除オプション> | 47ページの「stdoutとstderrの作成の回避」 |
| -S <パスリスト> | 21ページの「ジョブのトップシェルの指定」 |
| -u <ユーザーリスト> | 30ページの「ジョブのユーザー名の指定」 |
| -V | 134ページの「すべての環境変数のエクスポート」 |
| -v <変数リスト> | 134ページの「特定の環境変数のエクスポート」 |
| -W <属性>=<値> | 17ページの「ジョブの属性の設定」 |
| -W block=true | 128ページの「ジョブの終了までqsubを待機させる」 |
| -W create_resv_from_job=<値> | 148ページの「ジョブ固有開始予約」 |
| -W depend=<リスト> | 113ページの「ジョブの依存関係の使用」 |
| -W group_list=<リスト> | 30ページの「ジョブグループIDの指定」 |
| -W pwd | パスワードの入力を要求します |
| -W release_nodes_on_stageout=<値> | 136ページの「ジョブからの不要なvnodeの解放」 |
| -W run_count=<値> | 127ページの「ジョブの再実行回数の制御」 |
| -W sandbox=<値> | 35ページの「ステージングおよび実行ディレクトリ：ユーザーのホームディレクトリとジョブ固有ディレクトリ」 |
| -W stagein=<リスト> | 35ページの「入力/出力ファイルのステージング」 |
| -W stageout=<リスト> | 35ページの「入力/出力ファイルのステージング」 |
| -W umask=<値> | 49ページの「Linuxジョブのumaskの変更」 |

表2-1：qsubコマンドのオプション

| オプション | 機能および参照項目 |
|---|---|
| -X | 131ページの「インタラクティブLinuxジョブからのX出力の受け取り」 |
| -z | 32ページの「ジョブ識別子のstdoutへの出力の抑制」 |
| --version | PBSのバージョン情報を表示します。 |

## 2.5.1 電子メール通知の指定

PBSは、それぞれのジョブについて、そのジョブまたはサブジョブがそのライフサイクルの特定のポイントに達したときに、指定されている受信者にメールを送信できます。ジョブのライフサイクルには、PBSが常に電子メールを送信するポイントや、ユーザーが電子メールを受信することを選択できるポイントがあります。リストについては、以下の表をご参照ください。

表2-2：ジョブ/予約のライフサイクルのPBSのメール送信ポイント

| ライフサイクルのポイント | 常に送信かオプションか |
|---|---|
| ルーティングホップが多すぎるか、すべての宛先によって拒否されるため、ジョブをルーティングできないとき | オプション。メールは、-m aが指定された場合に送信されます。<br>サブジョブでは、メールは、-m ajが指定された場合に送信されます。 |
| ジョブがジョブオーナーによって削除されたとき | オプション（qdel -Wsuppress_emailの設定による） |
| ジョブがジョブオーナー以外のユーザーによって削除されたとき | 常に送信 |
| ジョブまたはサブジョブがPBSによって中止されたとき<br>不正なユーザー / グループアカウント、不正なチェックポイント / リスタートファイル、システムエラー、不正なリソース要求、または不正な依存関係により、ジョブまたはサブジョブを実行できません。 | オプション。メールは、-m aが指定された場合に送信されます。<br>サブジョブでは、メールは、-m ajが指定された場合に送信されます。 |
| 不正なパスワードによる保留でジョブがPBSによって保留されているとき | 常に送信 |
| ジョブが実行を開始したとき | オプション |
| ジョブが実行を終了したとき | オプション |
| ステージインが失敗したとき | 常に送信 |
| ファイルのステージアウトの試みがすべて失敗したとき | 常に送信 |
| 予約が確認または拒否されたとき | 常に送信 |

ジョブまたはサブジョブが削除されたとき、PBSは常にユーザーにメールを送信します。ジョブアレイの場合、PBSは１つのサブジョブにつき１つの電子メールを送信します。

ユーザーがジョブまたはサブジョブを削除したときにPBSが送信するジョブ関連の電子メールの数を制限できます。29ページの第2.5.1.3節「ジョブ削除時の電子メールの数の制限」をご参照ください。

### 2.5.1.1 ジョブのライフサイクルの電子メール送信ポイントの指定

PBSによるメール送信ポイントは、Mail_Pointsジョブ属性で指定します。-jサブオプションを1つまたは複数の他のサブオプションと一緒に使用すると、PBSはそれぞれのサブジョブに対してメールを送信します。このサブオプションがない場合は、ジョブと親アレイジョブに対してのみメールを送信します。Mail_Points属性は、以下の方法で設定できます。

- qsubの-m <メール送信ポイント>オプション
- qalterの-m <メール送信ポイント>オプション
- PBS指示文で#PBS -WMail_Points=<メール送信ポイント>と指定

*メール送信ポイント*引数は、次のいずれかを含む文字列です。

- "*n*"の1文字
- "*a*"、"*b*"、および"*e*"のうちの1つまたは複数の文字とオプションの"*j*"

-mオプションのサブオプションの一覧を次の表に示します。

**表2-3：mオプションのサブオプション**

| サブオプション | 意味 |
|---|---|
| *n* | メールを送信しません（not）。 |
| *a* | バッチシステムによってジョブまたはサブジョブが中止（aborted）された場合にメールを送信します。これがデフォルトです。 |
| *b* | ジョブまたはサブジョブの実行が開始（begin）されたときにメールを送信します。<br>例：<br>`Begun execution` |
| *e* | ジョブまたはサブジョブの実行が終了（end）したときにメールを送信します。 |
| *j* | サブジョブのメールを送信します。a、b、またはeのサブオプションのうち1つまたは複数と組み合わせる必要があります。 |

例2-8： ジョブが中止されたとき、または終了したとき、PBSはメールを送信します。

```
qsub -m ae my_job
#PBS -m ae
```

### 2.5.1.2 電子メール受信者リストの設定

PBSがメールを送信する受信者のリストは、Mail_Usersジョブ属性で指定します。Mail_Users属性は、以下の方法で設定できます。

- qsubの-M <メール受信者>オプション
- PBS指示文で#PBS -WMail_Users=<メール受信者>と指定

メール受信者引数は、次のように必要に応じてホスト名を付記したユーザー名のリストです。

*<ユーザー名>[@<ホスト名>][,<ユーザー名>[@<ホスト名>],...]*

以下に例を示します。

```
qsub -M user1@mydomain.com my_job
```

ジョブアレイに対してこのオプションを設定した場合、PBSはサブジョブごとにこのオプションを設定し、サブジョブごとにメールを送信します。

#### 2.5.1.3 ジョブ削除時の電子メールの数の制限

デフォルトでは、ユーザーがジョブまたはサブジョブを削除すると、PBS がユーザーに電子メールを送信します。`qdel -Wsuppress_email=<制限数>` を使用して、ユーザーが `qdel` を使用するたびにユーザーに送信される電子メールの数を制限できます。このオプションは、以下のように動作します。

制限数が *1* 以上
: 最大でも *制限数* の電子メールを受信します。

制限数が *0*
: PBS はこのオプションを無視します。

制限数が *-1*
: 電子メールを受信しません。

### 2.5.2 ジョブ名の指定

スクリプトを使用して、ジョブの名前を指定しないで、ジョブを投入する場合、ジョブの名前はデフォルトでスクリプトの名前になります。スクリプトを使用しないで、ジョブの名前を指定しないで、ジョブを投入する場合、ジョブの名前は STDIN になります。

ジョブの名前は、以下の方法で指定できます。

- `qsub -N <ジョブ名>` を使用
- `#PBS -N <ジョブ名>` を使用
- `#PBS -WJob_Name=<ジョブ名>` を使用

以下に例を示します。

```
qsub -N myName my_job
#PBS -N myName
#PBS -WJob_Name=my_job
```

ジョブ名は最大236文字で指定でき、印刷可能な空白以外の文字を使用する必要があります。先頭は、英文字、数字、ハイフン、アンダースコア、または正記号にする必要があります。

### 2.5.3 ジョブのプロジェクトの指定

PBS では、プロジェクトは、ユーザーやグループに関係なくジョブを整理する方法です。プロジェクトをタグとして使用して、一連のジョブをグループ化できます。各ジョブがメンバーになることができるのは 1 つのプロジェクトだけです。

プロジェクトがユーザーやグループに結び付けられることはありません。1 人のユーザーまたは 1 つのグループが、1 つ以上のプロジェクトのジョブを実行する場合もあります。たとえば、ユーザー Bob は ProjectA の JobA と ProjectB の JobB を実行します。ユーザー Bill は ProjectA の JobC を実行します。ユーザー Tom は ProjectB の JobD を実行します。Bob と Tom は Group1、Bill は Group2 に属しています。

ジョブの project 属性で、ジョブのプロジェクトを指定します。『PBS Professional Reference Guide』の「project」（341 ページ）をご参照ください。以下の方法で、ジョブの project 属性を設定できます。

- 投入時に `qsub -P <プロジェクト名>` を使用
- 投入後に、`qalter -P <プロジェクト名>` を使用。『PBS Professional Reference Guide』の「qalter」（130 ページ）をご参照ください。

## 2.5.4 ジョブのユーザー名の指定

PBSは、デフォルトでは、ユーザーのジョブを、そのユーザーがログインに使用するユーザー名の下で実行します。ジョブを実行するPBSサーバーに応じて、別のユーザー名の下でジョブを実行する必要がある場合があります。ジョブを実行できるユーザー名のリストを指定できます。リストのエントリのうち１つを除くすべてのエントリには、PBSサーバーのホスト名も指定する必要があります。これにより、PBSは、ホスト名を検索して、使用するユーザー名を選択できます。リストには、ホスト名を指定しないエントリを１つ指定できます。PBSは、ジョブが送信されたサーバーがリストに存在しない場合に、このエントリを使用します。

このユーザー名リストは、User_Listジョブ属性に格納されます。この属性の値は、デフォルトでは、ユーザーがログインで使用したユーザー名になります。この属性には、長さの制限はありません。

リストのエントリは、以下の形式で指定します。

*<ユーザー名>@<ホスト名>[,<ユーザー名>@<ホスト名> ...][,<ユーザー名>]*

User_Listの値は、ジョブを投入するときに`qsub -u <ユーザー名>`を使用して設定できるほか、ジョブを投入した後で`qalter -u <ユーザー名>`を使用して設定することもできます。

例2-9：ユーザーが投入ホストHostS上ではUserS、サーバーServerA上ではUserA、サーバーServerB上ではUserB、その他のサーバーではUserCであるとします。このユーザーは、すべてのExecutionA上ではUserA、すべてのExecutionBマシン上ではUserBでなくてはならないことにご留意ください。この場合、ユーザーはそのジョブに対して“`qsub -u UserA@ServerA,UserB@ServerB,UserC`”を使用できます。ジョブオーナーは常にUserSです。Linuxでは、UserA、UserB、UserCはそれぞれ、UserSをリスト表示する`.rhosts`ファイルをサーバーに保持する必要があります。

### 2.5.4.1 ジョブのユーザー名の変更に関する注意事項

- ユーザーは、どこでジョブを実行する場合も、指定したユーザー名の下でそのジョブを実行する許可が必要です。
  - Linuxの場合は7ページの第1.4.2.7節「Linuxでのユーザー権限」をご参照ください。
  - Windowsの場合は8ページの第1.4.3.4節「Windowsでのユーザー権限」をご参照ください。
- ユーザー名は最大で256文字です。

## 2.5.5 ジョブグループIDの指定

ユーザー名は複数のグループに属することができますが、各PBSジョブに関連付けられるのはそれらのグループのうち１つのみです。デフォルトでは、ジョブはプライマリグループの下で動作します。ジョブのグループは、group_listジョブ属性で指定します。ユーザーは、実行ホストでジョブを実行するグループを、コマンドラインで、またはPBS指示文を使用して、変更できます。

```
qsub -W group_list=<グループリスト>
#PBS group_list=<グループリスト>
```

以下に例を示します。

```
qsub -W group_list=grpA,grpB@jupiter my_job
```

*<グループリスト>*引数の形式は、以下のとおりです。

*<グループ>[@<ホスト名>][,<グループ>[@<ホスト名>],...]*

指定するホスト１つにつきグループを１つのみ指定できます。

対応するホストがないグループを１つのみ指定できます。引数リストで指定されていないホスト上で実行するときは、このグループ名が使用されます。

*group_list*は、デフォルトとして、そのジョブが動作するユーザー名のプライマリグループが適用されます。

#### 2.5.5.1 Windowsでのグループ名

Windowsでは、アカウントデータベースに照会した際にPBSが最初に検出したユーザー名のグループがプライマリグループとなります。

Windowsでは、割り当てられるデフォルトグループは、Windows APIのNetUserGetLocalGroup()とNetUserGetGroup()が最初のエントリとして何を返すかによって決まります。PBSは前者の出力（ローカルグループ）をチェックして、検出した最初のグループを返します。前者を呼び出して値が返されなかった場合、後者の呼び出しに進みます（グローバルグループ）。PBSが後者の呼び出しから何も出力を得られなかった場合、デフォルトの“Everyone”を使用します。

この場合、必ず“Users”を取得できると考えることはお勧めできません。場合によっては、`-Wgroup_list`オプションを指定しないでジョブを投入して、デフォルトグループの“None”がジョブに割り当てられる可能性があります。

### 2.5.6 アカウンティング文字列の指定

Account_Nameジョブ属性の値を設定することにより、ジョブにアカウンティング文字列を関連付けることができます。この属性にはデフォルト値はありません。ユーザーはコマンドラインで、またはPBS指示文で、Account_Nameの値を設定できます。

**`qsub -A <アカウンティング文字列>`**

`#PBS Account_Name=<アカウンティング文字列>`

<*アカウンティング文字列*>には、任意の文字列を指定できます。PBSはこの文字列を解釈しようとはしません。

### 2.5.7 サーバーまたはキュー、あるいはその両方の指定

PBSはデフォルトでデフォルトサーバーとデフォルトキューを提供するので、サーバーまたはキューを指定しないで投入されたジョブは、最終的にデフォルトサーバーでデフォルトキューに追加されます。

管理者がPBSサーバーに複数のキューを構成し、それらのキューがユーザーからのジョブを受け入れるように構成している場合、ユーザーはそのジョブをデフォルト以外のキューに投入できます。

- ジョブを主に1つのデフォルト以外のサーバーに投入する場合、優先するそのサーバーの名前をPBS_SERVER環境変数に設定します。この環境変数が優先サーバーに設定された後は、ジョブを優先サーバーに投入するときに、その名前を指定する必要はありません。
- ジョブを通常はデフォルトサーバーに投入しているが、今回だけデフォルト以外のサーバーの特定のキューに投入する必要がある場合：
  - `qsub -q <キュー名>@<サーバー名>`を使用
  - `#PBS -q <キュー名>@<サーバー名>`を使用
- ジョブを通常はデフォルトサーバーに投入しているが、今回だけデフォルト以外のサーバーでデフォルトキューに投入する必要がある場合：
  - `qsub -q @<サーバー名>`を使用
  - `#PBS -q @<サーバー名>`を使用
- デフォルトサーバーまたはPBS_SERVER環境変数で指定されているサーバー（定義されている場合）でデフォルト以外のキューにジョブを投入する場合：
  - `qsub -q <キュー名>`を使用
  - `#PBS -q <キュー名>`を使用

PBSサーバーにデフォルトキューがない場合、キューを指定しないでジョブを投入すると、`qsub`コマンドからエラーが返されます。

PBSまたは管理者が、キューにあるジョブを別のキューに移動する場合があります。qstat [ジョブID]を使用して、ジョブが存在するキューを確認できます。ジョブのqueue属性には、ジョブが存在するキューの名前が格納されます。

例：

```
qsub -q queue my_job
qsub -q @server my_job
#PBS -q queue1
qsub -q queue1@myserver my_job
qsub -q queue1@myserver.mydomain.com my_job
```

### 2.5.7.1 専用時間の使用または回避

*専用時間*とは、管理者が定義する1つまたは複数の特定の期間のことです。繰り返しの期間はありません。それぞれが個別に定義されます。

専用時間内は、PBSが開始するジョブは特別の専用時間キューにあるもののみです。PBSは非専用ジョブが専用時間内に実行されないようにスケジュールします。専用時間キュー内にあるジョブも、非専用時間内に実行されないようにスケジュールされます。PBSは専用時間と非専用時間の境界でバックフィルを試行します。

PBSはwalltimeを使用して専用時間内およびその周辺のスケジュールを行います。ジョブがwalltimeなしに非専用時間キューに投入された場合、専用時間がすべて終了するまではそのジョブは開始されません。ジョブがwalltimeなしに専用時間キューに投入された場合、そのジョブが実行されることはありません。

専用時間内に実行されるようにジョブを投入するには、*qsub*で *-q <キュー名>*オプションを使用してキュー名として使用する専用時間キューの名前を与えます。キューは管理者が作成します。キュー名については管理者に問い合わせてください。

## 2.5.8 ジョブ識別子のstdoutへの出力の抑制

ジョブ識別子の標準出力への出力を抑制するには、qsubの-zオプションを使用します。このオプションは、コマンドラインで、またはPBS指示文で使用できます。

```
qsub -z my_job
#PBS -z
```

このオプションに関連するジョブ属性はありません。

## 2.5.9 フォアグラウンドでのqsubの実行

通常は、qsubはバックグラウンドで動作します。-fオプションを使用すると、フォアグラウンドで実行できます。デフォルトでは、qsubは、以前の呼び出しによってインスタンス化されている可能性があるバックグラウンドのqsubデーモンと通信しようとします。このバックグラウンドデーモンは、認証済みのサーバー接続を維持して、性能を向上させることができます。

このオプションは、別のジョブを投入する非常に短いジョブを投入する場合、またはWindowsで社内開発したコードを実行する場合に役立つことがあります。

# 2.6 HPE Cray システム管理でのPBSジョブ

HPE Crayシステムの管理システムでのPBSジョブの投入は、標準のLinuxマシンでのジョブの投入とまったく同じです。

# 2.7 ジョブ投入の注意事項

## 2.7.1 混在Linux-Windows操作の注意事項

- WindowsクライアントからLinuxジョブを投入することはできません。
- Windowsジョブを投入するには、アーキテクチャがWindowsであることを指定します。以下に例を示します。

```
export PBS_AUTH_METHOD=pwd; qsub -lselect=1:arch=windows -- pbs-sleep 100
```

# 3 ジョブの入力ファイルと出力ファイル

## 3.1 PBSのジョブのファイルI/Oの概要

PBSでは、ユーザーが、入力ファイル、出力ファイル、標準出力、および標準エラーを管理できます。PBSには、ジョブファイルを処理する２つのメカニズムがあります。１つは入力ファイルと出力ファイルのステージングの使用、もう１つはKeep_Filesジョブ属性を使用して*stdout*や*stderr*をコピーするかどうかの選択です。

## 3.2 入力 / 出力ファイルのステージング

ファイルステージングは、ジョブを開始する前に実行ホストにコピーする入力ファイルと、ジョブが終了したときに実行ホストからコピーする出力ファイルを指定する方法です。

### 3.2.1 ステージングおよび実行ディレクトリ：ユーザーのホームディレクトリとジョブ固有ディレクトリ

ジョブの*ステージングと実行のディレクトリ*は、入力ファイルのステージング先であり、出力ファイルのステージング元であるディレクトリです。これは、ジョブスクリプト、pbs_tm() APIによって開始されたタスク、およびepilogueに対する現在の作業ディレクトリでもあります。このディレクトリは、ユーザーのホームディレクトリか、PBSによってこのジョブ専用に作成されたジョブ固有ディレクトリのいずれかです。

PBSでは、ジョブのステージングと実行のディレクトリとして使用される一時ディレクトリが各ジョブ固有に作成されます。各ジョブに専用のディレクトリがあれば、ファイル名の競合を防止できます。実行ホストの設定に応じて、これらのディレクトリはホームディレクトリの下位または他の場所に作成されます。

ジョブ固有のステージングと実行のディレクトリを使用する場合、実行ホストが適切に設定されている限り、実行ホストごとのホームディレクトリは必要ありません。

次の表は、ホームディレクトリをステージングおよび実行ディレクトリとして使用する方法と、PBSによって作成されたジョブ固有のステージングおよび実行ディレクトリを使用する方法の相違点を示しています。

**表3-1：ステージングおよび実行ディレクトリとしてのユーザーのホームディレクトリとジョブ固有ディレクトリの相違点**

| 処理、要件、または設定に関する確認事項 | ユーザーのホームディレクトリ | ジョブ固有ディレクトリ |
|---|---|---|
| PBSでは、ジョブ固有のステージングと実行のディレクトリを作成する必要がありますか。 | No | ホームディレクトリにない場合は作成する必要があります。 |
| 実行ホスト上にユーザーのホームディレクトリが必要ですか。 | Yes | No |
| `qsub -k`オプションを使用した場合、標準出力と標準エラーは自動的に削除されますか。 | No | Yes |
| ステージアウトファイルをどのようなタイミングで削除しますか。 | 正常にステージアウトされたファイルは削除し、それ以外は“undelivered”に移動する | すべてのファイルが正常にステージアウトされた後にのみ削除する |
| ジョブの終了後にステージングおよび実行ディレクトリは削除されますか。 | No | Yes |
| ジョブのsandbox属性はどのように設定されていますか。 | *HOME*または未設定 | *PRIVATE* |

## 3.2.2 ジョブ固有のステージングおよび実行ディレクトリの使用

### 3.2.2.1 ジョブのステージングおよび実行ディレクトリの設定

ジョブ固有のステージングと実行のディレクトリがPBSによって作成されるかどうかは、そのジョブのsandbox属性によって制御されます。

- ジョブのsandbox属性が*PRIVATE*に設定されている場合、PBSではジョブごとにステージングと実行のディレクトリが作成されます。
- ジョブのsandbox属性が*HOME*に設定されているか未設定の場合は、ジョブ固有のステージングと実行のディレクトリは作成されません。代わりに、PBSではジョブ投入者のホームディレクトリが使用されます。

ユーザーはqsubまたはPBS指示文を使用して、`sandbox`属性を設定できます。以下に例を示します。

```
qsub -Wsandbox=PRIVATE
```

ジョブのsandbox属性は、ジョブの実行中は変更できません。

### 3.2.2.2 ステージングと実行のディレクトリの場所

ジョブのjobdir属性は、プライマリホスト上にある、ジョブのステージングと実行のディレクトリのパス名に設定されます。ユーザーは、ジョブが実行中の場合に限り、`qstat -f`を使用してこの属性を表示できます。jobdirの値は、ジョブが再実行される場合はそのまま保持されません。ジョブが実行中でないときにjobdirが表示可能かどうかは未定義です。これは読み取り専用属性です。

PBSはPBS_JOBDIR環境変数を、プライマリ実行ホスト上のステージングと実行のディレクトリのパス名に設定します。PBS_JOBDIRは、ジョブスクリプトプロセス、任意のジョブタスク、およびprologueとepilogueに追加されます。

### 3.2.3 ステージングに影響を与える属性および環境変数

以下の属性および環境変数はステージングおよび実行に影響を与えます。

**表3-2：ステージングに影響を与える属性および環境変数**

| ジョブの属性または環境変数 | 影響 |
|---|---|
| sandbox属性 | PBSでステージングと実行のディレクトリとして、ユーザーのホームディレクトリを使用するか、ジョブ固有のディレクトリを作成するかを決定します。*PRIVATE*に設定している場合、ジョブ固有のディレクトリが作成されます。値が*HOME*または未設定の場合は、ユーザーのホームディレクトリがステージングおよび実行に使用されます。`qsub -W`またはPBS指示文によって、ジョブごとにユーザーが設定できます。 |
| stagein属性 | ステージインされるファイルまたはディレクトリのリストを設定します。`qsub -W`によって、ジョブごとにユーザーが設定できます。フォーマットは次のとおりです。<br>*execution_path@storage_host:storage_path*<br>*execution_path*は、ステージングと実行のディレクトリへのパスです。ステージインの場合は、*storage_path*は入力ファイルが通常配置される場所のパスです。 |
| stageout属性 | ステージアウトされるファイルまたはディレクトリのリストを設定します。`qsub -W`によって、ジョブごとにユーザーが設定できます。フォーマットは次のとおりです。<br>*execution_path@storage_host:storage_path*<br>*execution_path*は、ステージングと実行のディレクトリへのパスです。ステージアウトの場合は、*storage_path*は出力ファイルが配置される場所のパスです。 |
| Keep_Files属性 | 出力ファイルまたはエラーファイル（あるいはその両方）が実行ホストにそのまま残されるかを決定します。`qsub -k`またはPBS指示文によって、ジョブごとにユーザーが設定できます。Keep_Files属性が*o*と*e*の一方または両方（出力ファイルまたはエラーファイル、あるいはその両ファイルがステージングと実行のディレクトリに残される）に設定され、ジョブのsandbox属性が*PRIVATE*に設定されている場合、ステージングと実行のディレクトリがジョブ終了時にその内容と共に削除されると、標準出力ファイルまたはエラーファイル（あるいはその両方）が削除されます。`-k`引数の`-d`サブオプションを通じてファイルの直接書き込みが指定されている場合は、ファイルは削除されません。48ページの第3.3.5節「実行ホストでの出力ファイルとエラーファイルの保存」をご参照ください。 |
| jobdir属性 | プライマリ実行ホスト上のステージングおよび実行ディレクトリのパス名に設定されます。読み取り専用。`qstat -f`で表示できます。 |
| Remove_Files属性 | ジョブの完了時に標準出力や標準エラーファイルを自動的に削除するかどうかを指定します。 |
| PBS_JOBDIR環境変数 | プライマリ実行ホスト上のステージングおよび実行ディレクトリのパス名に設定されます。ジョブスクリプトプロセス、pbs_tmジョブタスク、およびprologueとepilogueの環境に追加されます。 |
| TMPDIR環境変数 | ジョブ固有のスクラッチディレクトリの場所。 |

## 3.2.4 ファイルのステージインまたはステージアウトの指定

ジョブのstagein属性とstageout属性を設定すると、ジョブの実行前にステージインするファイルおよびジョブの実行後にステージアウトするファイルを指定できます。qsubのオプションまたはジョブスクリプト内の指示文を使用できます。

*qsub -Wstagein=<実行パス>@<入力ファイルストレージホスト>:<入力ファイルストレージパス>[,...]-Wstageout=<実行パス>@<出力ファイルストレージホスト>:<出力ファイルストレージパス>[,...]*

*#PBS -W stagein=<実行パス>@<入力ファイルストレージホスト>:<入力ファイルストレージパス>[,...]*

*#PBS -W stageout=<実行パス>@<出力ファイルストレージホスト>:<出力ファイルストレージパス>[,...]*

*実行パス*は、（実行ホスト上の）ジョブのステージングと実行のディレクトリにあるファイルの名前です。*実行パス*は、ジョブのステージングと実行のディレクトリを基準とした相対パスまたは絶対パスで指定できます。

文字'@'で実行の指定と保存の指定を区切ります。

*ストレージパス*は、*ストレージホスト*で指定されたホスト上のファイル名です。ステージインの場合は、入力ファイルの取得元の場所を示します。ステージアウトの場合は、ジョブの終了時における出力ファイルの格納先です。hostnameを指定する必要があります。このパスは、絶対パスまたは*ストレージホスト*という名前のマシン上にあるユーザーのホームディレクトリを基準とした相対パスで表すことができます。

ステージインの場合、方向は*ストレージパス*から*実行パス*になります。

ステージアウトの場合、方向は*実行パス*から*ストレージパス*になります。

次の例では、指示文を使用して、serverAというホストのディレクトリ/u/user1にあるgrid.datという名前のファイルをステージインする方法を示しています。ステージインファイルは、data1という名前でステージングと実行のディレクトリにコピーされます。*実行パス*はステージングおよび実行ディレクトリを基準に評価されるため、data1のフルパス名を指定する必要はありません。

```
#PBS -W stagein=data1@serverA:/u/user1/grid.dat ...
```

qsubオプションを使用して、myhost（/Users/myhome/mydata/data1）に保存されているファイルをステージインし、ファイル名をinput_data1としてステージングおよび実行ディレクトリに格納するには、次のように指定します。

```
qsub -W stagein=input_data1@myhost:/Users/myhome/mydata/data1
```

複数のファイルまたはディレクトリをステージングする場合は、コンマで区切られたパスのリストを使用し、そのリストを二重引用符で囲みます。たとえば、data1とdata2の２つのファイルをステージングする場合は、次のように指定します。

```
qsub -W stagein="input1@hostA:/myhome/data1,input2@hostA:/myhome/data1"
```

## 3.2.5 ステージングの注意事項と要件

### 3.2.5.1 Linux：ステージングおよび特殊文字

ファイル名やディレクトリ名に、括弧などの特殊文字を使用する必要がある場合、パスのその部分を引用符の追加レイヤで囲みます。構文：

*-W stageout="<実行パス>@<ストレージホスト>:'<ストレージパス>'"*

例：

```
-W stageout="myoutfile@myhost:'/home/user1/outfile(1234)'"
```

### 3.2.5.2 Windows：ステージングおよび特殊文字またはパス

#### 3.2.5.2.i 特殊文字

Windowsでは、パスに空白、バックスラッシュ（¥）、コロン（:）などの特殊文字やドライブ文字が含まれる場合、ステージング指定を二重引用符で囲みます。たとえば、hostBにあるドライブDのgrid.datファイルをドライブCの実行ファイル“dat1”にステージングする場合、以下のように指定します。

```
qsub -W stagein="dat1@hostB:D¥Documents and Settings¥grid.dat"
```

#### 3.2.5.2.ii UNCパスの使用

UNCパスを使用してステージインまたはステージアウトする場合、ホスト名はオプションです。UNC以外のパスを使用する場合、ホスト名は必須です。

### 3.2.5.3 ステージングのパス名

- *ストレージパス*には、絶対パス名を使用することをお勧めします。ユーザーのホームディレクトリへのパスはマシンごとに異なる場合があります。また、sandbox = *PRIVATE*を使用している場合は、必ずしもすべての実行マシン上にホームディレクトリを置く必要があるとは限りません。
- PBSによってジョブのステージングと実行のディレクトリが作成される場合、*実行パス*には必ず相対パス名を使用します。これは、ジョブ固有のステージングと実行のディレクトリを使用している場合は、実行パスに絶対パスを指定しないことを意味します。

### 3.2.5.4 必要なアクセス権

ステージインするファイルまたはディレクトリの読み取りアクセス許可およびステージアウトするファイルまたはディレクトリの書き込みアクセス許可が必要です。

### 3.2.5.5 アンパーサンドに関する警告

ステージングパスには、アンパーサンド（“&”）を使用できません。ステージングが失敗します。

### 3.2.5.6 インタラクティブジョブとファイルI/O

インタラクティブジョブの終了時に、ステージングされていたファイルがまだコピーされていない可能性があります。

### 3.2.5.7 ステージングおよび実行ディレクトリに対するディレクトリのコピー

ファイルのステージングと同じ方法で、ディレクトリをステージングおよび実行ディレクトリにステージインまたはステージアウトできます。ステージインとステージアウトの両方で、*ストレージパス*および*実行パス*にディレクトリを指定できます。ディレクトリをステージインまたはステージアウトする場合、PBSでは対象のディレクトリとその配下のファイルおよびサブディレクトリをすべてコピーします。ジョブの終了時に、ディレクトリは、すべてのファイルおよびサブディレクトリも含めて削除されます。この動作は、複数のジョブで同じディレクトリを使用していると問題になることがあります。そのような問題を回避するには、ジョブ固有のステージングと実行のディレクトリがPBSで作成されるようにします。そのためには、ジョブに対してsandbox=*PRIVATE*を設定します。

### 3.2.5.8 ファイルステージングにおけるワイルドカード

ファイルやディレクトリのステージングでは、以下の規則に従って、ワイルドカードを使用できます。

- アスタリスク“*”は1つまたは複数の文字に一致します。
- 疑問符“?”は1文字に一致します。
- その他の文字は、それぞれの文字にのみ一致します。
- 引用符内のワイルドカードは展開されます。
- ピリオド（“.”）で始まるLinuxファイル名または“SYSTEM”属性や“HIDDEN”属性を持つWindowsファイル名では、ワイルドカードを使用した一致操作はできません。
- Linux上で`qsub`コマンドラインを使用する場合は、シェルがワイルドカードを展開しないようにする必要があります。一部のシェルでは、パス名を二重引用符で囲むことができます。一部のシェルでは、ワイルドカードの前にバックスラッシュを使用できます。
- ワイルドカードを使用できるのは、ステージング指定のソース側のみに限られます。つまり、ステージインの*ストレージパス*指定およびステージアウトの*実行パス*指定で使用可能です。
- ワイルドカードを使用してステージングする場合は、宛先をディレクトリにする必要があります。宛先がディレクトリでない場合は、結果が定義されません。したがって、すべての`.out`ファイルをステージアウトする場合、*ストレージパス*にディレクトリを指定する必要があります。
- ワイルドカードは、パスの最後の部分（つまり基本名）にのみ使用できます。
- ジョブ固有のステージングと実行のディレクトリがPBSで作成されていない場合、ステージインでワイルドカードを使用すると、ステージングしたファイルがジョブの終了時に自動的には削除されません。ステージングと実行のディレクトリが作成されていれば、そのディレクトリと配下のすべての内容がジョブの終了時に削除されます。

## 3.2.6 ファイルステージングの例

例3-1： 実行ディレクトリから特定のディレクトリにすべてのファイルをステージアウトします。

Linux

```
-W stageout=*@myworkstation:/user/project1/case1
```

Windows

```
-W stageout=*@mypc:E:¥project1¥case1
```

例3-2： ジョブの終了後は、特定のタイプの結果ファイルをステージアウトし、スクラッチファイルおよびその他の一時ファイルは無視します。この例で使用する結果ファイルの末尾は'.dat'です。

Linux

```
-W stageout=*.dat@myworkstation:project3/data
```

Windows

```
-W stageout=*.dat@mypc:C:¥project¥data
```

例3-3： アプリケーションデータディレクトリからサブディレクトリにすべてのファイルをステージインします。

Linux

```
-W stagein=jobarea@myworkstation:crashtest1/*
```

Windows

```
-W stagein=jobarea@mypc:E:¥crashtest1¥*
```

例3-4： “wing*”と一致するファイルおよびディレクトリからデータをステージインします。

Linux

```
-W stagein=.@myworkstation:848/wing*
```

Windows

```
-W stagein=.@mypc:E:¥flowcalc¥wing*
```

例3-5： .batおよび.datファイルをジョブエリアにステージインします。

Linux：

```
-W stagein=jobarea@myworkstation:/users/me/crash1.?at
```

Windows：

```
-W stagein=jobarea@myworkstation:C:¥me¥crash1.?at
```

#### 3.2.6.1 ジョブ固有のステージングおよび実行ディレクトリの使用例

この例では、ホスト“submithost”からコピーすることにより、PBS_JOBDIRで指定されているジョブ固有のステージングと実行のディレクトリに“jay.fem”という名前のファイルを送達します。ジョブスクリプトはPBS_JOBDIR内で実行され、“jay.out”はPBS_JOBDIRから投入ホスト（“ストレージホスト”）上のホームディレクトリにステージアウトされます。

```
qsub -Wsandbox=PRIVATE -Wstagein=jay.fem@submithost:jay.fem -Wstageout=jay.out@submithost:jay.out
```

### 3.2.7 ジョブのライフサイクルの概要

ここでは、PBSによって実行される手順の概要を示します。これらの手順は、必ずしも記載どおりの順序で実行する必要はありません。

- PBSにより、各実行ホスト上（指定されている場合）で、ジョブ固有のステージングおよび実行ディレクトリが作成されます。
- PBS_JOBDIRとジョブのjobdir属性がジョブのステージングと実行のディレクトリのパスに設定されます。
- ジョブに割り当てられている各実行ホスト上で、一時スクラッチディレクトリが作成されます。
- TMPDIR環境変数は一時スクラッチディレクトリのパス名に設定されます。
- ディレクトリの作成時または変数の設定時に何らかのエラーが発生すると、ジョブは再キューイングされます。
- ファイルまたはディレクトリがステージインされます。
- prologue がプライマリ実行ホスト上で実行されます。この際、PBS_HOME/mom_priv をカレントディレクトリとし、PBS_JOBDIRおよびTMPDIRは環境変数として設定されます。
- ジョブがプライマリ実行ホスト上でユーザーとして実行されます。
- ジョブの関連するタスクが実行ホスト上でユーザーとして実行されます。
- epilogueがプライマリ実行ホスト上で実行されます。この場合は、ジョブのステージングと実行のディレクトリがカレントディレクトリになり、PBS_JOBDIRおよびTMPDIRは環境変数として設定されます。
- ファイルまたはディレクトリがステージアウトされます。
- PBSはジョブのRemove_Files属性の値に従って、標準エラーまたは標準出力、あるいはその両方を削除します。
- ステージングされたファイルまたはディレクトリが削除されます。
- ジョブ固有のステージングと実行のディレクトリがPBSで作成されている場合は、そのディレクトリとその内容、およびすべてのTMPDIRとその内容が削除されます。
- 最終ジョブアカウンティングレコードが書き込まれ、サーバーのデータベースからジョブ情報が削除されます。

## 3.2.8 ジョブのライフサイクルの詳細

### 3.2.8.1 TMPDIRの作成

ジョブに割り当てられた各ホストに対して、PBSではこのジョブに対するジョブ固有の一時スクラッチディレクトリが作成されます。一時スクラッチディレクトリを作成できないと、ジョブは中止されます。

### 3.2.8.2 ステージングおよび実行ディレクトリの選択肢

ジョブのsandbox属性を*PRIVATE*に設定している場合、そのジョブに対して、ジョブ固有のステージングと実行のディレクトリがPBSによって作成されます。ジョブのsandbox属性を*HOME*に設定しているか、何も設定していない場合は、ユーザーのホームディレクトリがステージングと実行に使用されます。

#### 3.2.8.2.i ジョブ固有のステージングおよび実行ディレクトリ

ステージングおよび実行ディレクトリを作成できないと、ジョブは中止されます。PBSでステージングおよび実行ディレクトリの作成に失敗する場合は、システム管理者に確認してください。

PBSが作成するステージングおよび実行ディレクトリに対しては特定の命名規則を想定するべきではありません。

#### 3.2.8.2.ii ステージングおよび実行ディレクトリとして使用されるユーザーのホームディレクトリ

ユーザーは各実行ホスト上にホームディレクトリを保有する必要があります。ユーザーのホームディレクトリが存在しないとエラーが発生し、この結果、ジョブが中止されます。

### 3.2.8.3 環境変数および属性の設定

PBS_JOBDIRおよびジョブのjobdir属性は、プライマリホスト上にある、ジョブのステージングと実行のディレクトリのパス名に設定されます。TMPDIR環境変数は、ジョブ固有の一時スクラッチディレクトリのパス名に設定されます。

### 3.2.8.4 ステージングおよび実行ディレクトリへのファイルのステージング

PBSでは、プライマリ実行ホストにファイルがステージインされます。PBSでは、PBS_JOBDIRで指定されたステージングおよび実行ディレクトリを基準に実行パスおよびストレージパスを評価し、このディレクトリがユーザーのホームディレクトリであるか、またはPBSによって作成されたジョブ固有ディレクトリであるかを判断します。指定したファイルまたはディレクトリ（あるいはその両方）がジョブのステージングと実行のディレクトリにコピーされます。

### 3.2.8.5 prologueの実行

MoMのprologueは、プライマリホスト上でPBS_HOME/mom_privを作業ディレクトリとしてroot権限で実行されます。この際に、PBS_JOBDIRとTMPDIRは環境変数として設定されます。

### 3.2.8.6 ジョブの実行

PBSではジョブスクリプトをプライマリホスト上でユーザーとして実行します。また、ジョブで作成されたタスクもユーザーとして実行します。ジョブスクリプトおよびタスクは、ジョブのステージングおよび実行ディレクトリをカレントディレクトリとして実行され、PBS_JOBDIRおよびTMPDIRは環境変数として設定されます。

#### 3.2.8.7 標準出力と標準エラー

-kオプションの-dサブオプションを使用して、ジョブのstdoutファイルとstderrファイルが最終的な目的の場所に直接書き込まれるように指定していない限り、プライマリ実行ホスト上のジョブのステージングと実行のディレクトリにこれらのファイルが直接作成されます。

##### 3.2.8.7.i ジョブ固有のステージングおよび実行ディレクトリ

sandboxを*PRIVATE*に設定し、qsub -kオプションを指定していると、stdoutファイルとstderrファイルはジョブ終了時にステージングと実行のディレクトリから**自動的にはコピーされません**。これらのファイルは、ステージングと実行のディレクトリが自動的に削除される際に削除されます。-kオプションの-dサブオプションを使用して、ファイルが最終的な目的の場所に書き込まれるようにしていると、最初の場所のステージングと実行のディレクトリにはそれらのファイルが作成されません。

##### 3.2.8.7.ii ステージングおよび実行ディレクトリとして使用されるユーザーのホームディレクトリ

sandboxを*HOME*に設定するか何も設定せず、qsubに-kオプションを指定していると、標準出力ファイルおよび/または標準エラーファイルは、投入ホストには返されずにプライマリ実行ホストに保持され、ジョブが終了しても削除されません。

#### 3.2.8.8 epilogueの実行

PBSはプライマリホスト上でepilogueをrootとして実行します。epilogueはジョブのステージングおよび実行ディレクトリを現在の作業ディレクトリとして実行され、PBS_JOBDIRおよびTMPDIRが環境変数として設定されます。

#### 3.2.8.9 ファイルのステージアウトと実行ディレクトリの削除

PBSはファイルをステージアウトする際、PBS_JOBDIRを基準に実行パスおよびストレージパスを評価します。ステージアウトできないファイルは、PBS_HOME/undeliveredに保存されます。

##### 3.2.8.9.i ジョブ固有のステージングおよび実行ディレクトリ

ジョブ固有のステージングと実行のディレクトリがPBSによって作成されている場合、それらのディレクトリはジョブが終了するとクリーンアップされます。すべての実行ホストにあるステージングと実行のディレクトリおよびその内容が削除されます。

#### 3.2.8.10 TMPDIRおよびファイルの削除

PBSはすべてのTMPDIRをその内容と共に削除します。Remove_Filesで出力ファイルまたはエラーファイル（あるいはその両方）が指定されている場合、これらのファイルは削除されます。

### 3.2.9 ジョブアレイのステージング

ファイルステージングはジョブアレイについてサポートされています。164ページの「ジョブアレイのためのファイルステージング」をご参照ください。

## 3.2.10 ステージインとステージアウトの失敗

### 3.2.10.1 ファイルのステージインの失敗

ステージインに失敗すると、問題を修正する猶予のため、ジョブは３０分間の待機状態になります。通常、ファイルが見つからない場合か、ネットワーク障害で発生します。問題が検出された場合、電子メールがジョブオーナーに送信されます。問題を解決した後、ジョブオーナーまたはPBSオペレータは、qalterに-aオプションを付けてジョブを実行可能に設定し、時間を再設定することで待機状態を解除できます。サーバーはジョブが待機状態になった理由に関する情報でジョブのコメントを更新します。ジョブが実行可能になると、他のvnodeで実行されます。

### 3.2.10.2 ファイルのステージアウトの失敗

ステージアウトでエラーが発生した場合、３度の再試行が行われます。PBSは1秒待機後再試行し、11秒待機後３回目を試行し、最後にさらに21秒待機後４回目を試行します。すべての試行が失敗すると、電子メールがジョブオーナーに送信されます。ステージアウトできないファイルは、PBS_HOME/undeliveredに保存されます。50ページの第3.3.8.1節「出力結果が送信されない」をご参照ください。

# 3.3 出力ファイルとエラーファイルの管理

## 3.3.1 出力ファイルとエラーファイルのデフォルトの動作

デフォルトで、PBSでは、ジョブの終了時に標準出力（stdout）ファイルおよび標準エラー（stderr）ファイルのコピーを投入ホスト上の$PBS_O_WORKDIRに返します。qsubの実行時に、$PBS_O_WORKDIRはqsubコマンドが実行されるカレントディレクトリに設定されます。これは、ジョブのstdoutファイルとstderrファイルを投入ディレクトリに送達する必要がある場合、何もする必要がないことを意味します。

qsubコマンドの次のオプションで、stdoutとstderrの作成先およびジョブ終了時のそれらのファイルのコピー先を制御します。

sandbox

デフォルトでは、PBSはジョブスクリプトをオーナーのホームディレクトリで実行します。sandboxを*PRIVATE*に設定している場合、ジョブ固有のステージングと実行のディレクトリがPBSによって作成され、そこでジョブスクリプトが実行されます。36ページの第3.2.2.1節「ジョブのステージングおよび実行ディレクトリの設定」をご参照ください。

k

k {e | o | eo | oe | n}

-e、-o、-eo、-oe、-nの各サブオプションとともに使用して、stdoutまたはstderrを保持するかどうか、保持する場合はどちらを保持するかを指定します。設定すると、このオプションは-o <出力パス>と-e <エラーパス>より優先されます。48ページの第3.3.5節「実行ホストでの出力ファイルとエラーファイルの保存」をご参照ください。

kd {e | o | eo | oe}

-dサブオプションとともに使用し、出力ファイルおよび/またはエラーファイルを最終的な目的の場所に書き込むことを指定します。eサブオプションとoサブオプション（またはそのどちらか）を指定する必要があります。49ページの第3.3.6節「最終宛先へのファイルの直接書き込み」をご参照ください。

o

stdoutの宛先を指定します。kが設定されている場合は、kのほうが優先されます。46ページの第3.3.2節「出力ファイルとエラーファイルのパス」をご参照ください。

e

stderr の宛先を指定します。k が設定されている場合は、k のほうが優先されます。46 ページの第 3.3.2 節「出力ファイルとエラーファイルのパス」をご参照ください。

R

ジョブの完了時に標準出力や標準エラーを削除するかどうかを指定します。47 ページの第 3.3.3 節「stdout と stderr の作成の回避」をご参照ください。

以下の表に、これらのオプションによる stdout と stderr の作成とコピーの制御方法を示します。

### 表 3-3：qsub の k、sandbox、o、および e の各オプションが stdout と stderr に与える影響

| sandbox | -k (o, e, eo, oe) | -e、-o | -R | -k d | stdout と stderr の作成先 | stdout と stderr のコピー先 |
|---|---|---|---|---|---|---|
| *HOME* または未設定 | 設定なし | 設定なし | 設定なし | 設定なし | PBS_HOME/spool | PBS_O_WORKDIR（ジョブ投入ディレクトリ） |
| *HOME* または未設定 | 設定なし | *<パス>* | 設定なし | 設定なし | PBS_HOME/spool | -o <パス>や-e <パス>で指定する宛先 |
| *HOME* または未設定 | *e*、*o*、*eo*、*oe* | 設定なし | 設定なし | 設定なし | 実行ホスト上の、ジョブ投入者のホームディレクトリ | コピーされずに、実行ホスト上の投入者のホームディレクトリに残り、削除されない。 |
| *HOME* または未設定 | *e*、*o*、*eo*、*oe* | *<パス>* | 設定なし | 設定なし | 実行ホスト上の、ジョブ投入者のホームディレクトリ | コピーされずに、実行ホスト上の投入者のホームディレクトリに残り、削除されない。 |
| *PRIVATE* | 設定なし | 設定なし | 設定なし | 設定なし | PBS によって作成されるジョブ固有の実行ディレクトリ | PBS_O_WORKDIR（ジョブ投入ディレクトリ） |
| *PRIVATE* | 設定なし | *<パス>* | 設定なし | 設定なし | PBS によって作成されるジョブ固有の実行ディレクトリ | -o <パス>や-e <パス>で指定する宛先 |
| *PRIVATE* | *e*、*o*、*eo*、*oe* | 設定なし | 設定なし | 設定なし | PBS によって作成されるジョブ固有の実行ディレクトリ | コピーされずに、ジョブ固有実行ディレクトリに残る。ジョブ固有実行ディレクトリが削除されると削除される。 |
| *PRIVATE* | *e*、*o*、*eo*、*oe* | *<パス>* | 設定なし | 設定なし | PBS によって作成されるジョブ固有の実行ディレクトリ | コピーされずに、ジョブ固有実行ディレクトリに残る。ジョブ固有実行ディレクトリが削除されると削除される。 |
| 任意 | 任意 | 任意 | *-R e/o* | 任意 | 作成先に関係なく削除される。 | 存在せず、コピーされない。 |
| 任意 | 任意 | 任意 | 設定なし | *-k d <o または e（あるいは両方）>* | MoM がアクセス可能な場合、-o <出力パス>または-e <エラーパス>で指定された最終宛先 | 存在せず、コピーされない。 |

- stdout と stderr のパスを指定できます。46 ページの第 3.3.2 節「出力ファイルとエラーファイルのパス」をご参照ください。
- stdout と stderr を結合できます。48 ページの第 3.3.4 節「出力ファイルとエラーファイルの結合」をご参照ください。
- stdout と stderr の作成を回避できます。47 ページの第 3.3.3 節「stdout と stderr の作成の回避」をご参照ください。
- 実行ホスト上に stdout と stderr を保持するかどうかを指定できます。48 ページの第 3.3.5 節「実行ホストでの出力ファイルとエラーファイルの保存」をご参照ください。
- 出力ファイルまたはエラーファイル、あるいはその両方を最終宛先に直接書き込むように指定できます。49 ページの第 3.3.6 節「最終宛先へのファイルの直接書き込み」をご参照ください。
- 出力ファイルまたはエラーファイル、あるいはその両方をジョブ終了時に削除するように指定できます。47 ページの第 3.3.3 節「stdout と stderr の作成の回避」をご参照ください。

## 3.3.2 出力ファイルとエラーファイルのパス

### 3.3.2.1 出力ファイルとエラーファイルのデフォルトパス

デフォルトでは、PBS はジョブ名とジョブのシーケンス番号を使用して、ジョブの出力ファイルとエラーファイルの名前を付けます。出力ファイル名は Output_Path ジョブ属性で指定され、エラーファイル名は Error_Path ジョブ属性で指定されます。

デフォルトの出力ファイル名は以下の形式になります。

*<ジョブ名>.o<シーケンス番号>*

デフォルトのエラーファイル名は以下の形式になります。

*<ジョブ名>.e<シーケンス番号>*

*ジョブ名*は、指定されない場合は、デフォルトでスクリプト名になります。たとえば、ジョブ ID が 1234.exampleserver、スクリプト名が "myscript" の場合、エラーファイルの名前は myscript.e1234 です。ジョブの名前を指定した場合、スクリプト名の代わりにジョブ名が使用されます。たとえば、ジョブの名前として "fixgamma" を指定した場合、出力ファイルの名前は fixgamma.o1234 です。

ジョブ名の指定の詳細については、29 ページの第 2.5.2 節「ジョブ名の指定」をご参照ください。

### 3.3.2.2 パスの指定

ジョブごとに Output_Path と Error_Path の各ジョブ属性の値を設定することによって、ジョブの出力ファイルとエラーファイルのパスと名前を指定できます。これらの属性は、以下の方法で設定できます。

- qsub の -o <出力パス> オプションと -e <エラーパス> オプションを使用
- ジョブスクリプト内で #PBS Output_Path=<パス> 指示文と #PBS Error_Path=<パス> 指示文を使用

パス引数の形式は、以下のとおりです。

*[<ホスト名>:]<パス名>*

*ホスト名*はホストの名前、*パス名*はそのホスト上のパス名です。

相対パスまたは絶対パスのいずれかを指定できます。ファイル名のみを指定した場合は、ホームディレクトリを基準とする相対パスであるとみなされます。パスに変数は使用しないでください。

次の例は、パスの指定方法を示しています。

```
#PBS -o /u/user1/myOutputFile
#PBS -e /u/user1/myErrorFile

qsub -o myOutputFile my_job
qsub -o /u/user1/myOutputFile my_job
qsub -o myWorkstation:/u/user1/myOutputFile my_job
qsub -e myErrorFile my_job
qsub -e /u/user1/myErrorFile my_job
qsub -e myWorkstation:/u/user1/myErrorFile my_job
```

### 3.3.2.3 Windows ホストからのパスの指定

#### 3.3.2.3.i パスでの特殊文字の使用

Windows ホストからジョブを投入する場合、空白、バックスラッシュ（“¥”）、コロン（“:”）などの特殊文字を使用したパス名の指定が必要になることがあります。また、ドライブ文字の指定が必要なことも考えられます。以下の例は許可されます。

```
qsub -o ¥temp¥my_out job.scr
qsub -e "myhost:e:¥Documents and Settings¥user¥Desktop¥output"
```

この例のジョブのエラー出力は、myhost の e: ドライブのパス "¥Documents and Settings¥user¥Desktop¥output" にコピーされます。

#### 3.3.2.3.ii UNC パスの使用

出力ファイルまたはエラーファイルに UNC パスを使用する場合、ホスト名はオプションです。UNC 以外のパスを使用する場合、ホスト名は必須です。

### 3.3.2.4 パスの注意事項

引数に空白が含まれる場合は、qsub の引数を引用符で囲みます。

## 3.3.3 stdout と stderr の作成の回避

PBS は常に、ジョブごとにジョブの出力ファイルとエラーファイルを作成します。ファイルが作成される場所は、45 ページの表 3-3「qsub の k、sandbox、o、および e の各オプションが stdout と stderr に与える影響」に示されています。

stdout や stderr が必要ない場合は、次のいずれかを実行できます。

- qsub または qalter に -R オプションを使用して、ジョブの終了時に PBS でファイルを削除するように指定する。-R オプションは、サブオプションとして *o*、*e*、*eo*、*oe* を取ります。たとえば、PBS にエラーファイルを削除させるには、以下のように指定します。
  **qsub -R e job.sh**
- ジョブスクリプト内でそれらを /dev/null にリダイレクトする。たとえば、stdout と stderr を /dev/null にリダイレクトするには、以下を実行します。
  exec >&/dev/null 1>&2
- 標準出力と標準エラーは、通常 /var/spool などの場所に書き込まれた後、最終的な場所にコピーされます。これらのファイルの作成を回避し、コピーもされないようにするには、直接書き込みを使用してこれらを /dev/null に送信します。
  **qsub -koed -o /dev/null -e /dev/null**

これをサポートするには、管理者がMoMの設定ファイルを設定することも必要です。

## 3.3.4 出力ファイルとエラーファイルの結合

デフォルトでは、PBSは、ジョブごとに個別に標準出力ファイルと標準エラーファイルを作成します。ジョブのJoin_Path属性を設定することによって、stdoutとstderrを結合することを指定できます。この属性のデフォルトは*n*で、結合は行われないことを意味します。この属性は、以下の方法で設定できます。

- qsub -j <結合オプション>を使用
- #PBS Join_Path=<結合オプション>を使用

以下のいずれかの*結合オプション*を指定できます。

oe
: 標準出力と標準エラーが１つのストリームに結合され、これが標準出力になります。

eo
: 標準出力と標準エラーが１つのストリームに結合され、これが標準エラーになります。

n
: 標準出力と標準エラーは結合されません。

たとえば、my_jobの標準出力と標準エラーを結合して標準出力にするには、以下のように指定します。

```
qsub -j oe my_job
#PBS -j oe
```

## 3.3.5 実行ホストでの出力ファイルとエラーファイルの保存

デフォルトでは、PBSはstdoutとstderrをジョブの投入ディレクトリにコピーします。PBSがstdoutまたはstderr、あるいは両方を実行ホスト上のジョブの実行ディレクトリに保持することを指定できます。この動作は、ジョブのKeep_Files属性によって制御されます。この属性は、以下のいずれかの値に設定できます。

e
: PBSは、stderrをプライマリ実行ホスト上のジョブのステージングと実行のディレクトリに保持します。

o
: PBSは、stdoutをプライマリ実行ホスト上のジョブのステージングと実行のディレクトリに保持します。

eo, oe
: PBSは、標準出力と標準エラーの両方をプライマリ実行ホスト上のジョブのステージングおよび実行ディレクトリに保存します。

n
: PBSは、どちらのファイルも実行ホスト上に保存しません。

d
: PBSはstdoutとstderrの両方をそれらの最終宛先に書き込みます。-o <出力パス>オプションおよび/または-e <エラーパス>オプションを指定する必要があります。49ページの第3.3.6節「最終宛先へのファイルの直接書き込み」をご参照ください。

Keep_Filesのデフォルト値は“*n*”です。

Keep_Filesジョブ属性の値は、以下の方法で設定できます。

- qsub -k <破棄オプション>を使用
- #PBS Keep_Files=<破棄オプション>を使用

たとえば、以下のどちらかを使用して、標準出力と標準エラーの両方を実行ホスト上に保存できます。

```
qsub -k oe my_job
#PBS -k oe
```

### 3.3.5.1 実行ホストでのファイルの保存の注意事項

- PBS によってジョブ固有のステージングと実行のディレクトリが作成されていると、ジョブが終了したときにそのディレクトリとそこにあるすべてのファイルが削除されます。stdout または stderr、あるいはその両方を実行ホスト上に保存する必要があることを指定していた場合は、指定したファイルも同様に削除されます。
- qsub -kオプションは、-oオプションと-eオプションよりも優先されます。たとえば、qsub -k o -o <パス>を指定する場合、stdoutは実行ホスト上で保存され、指定したパスにはコピーされません。

## 3.3.6 最終宛先へのファイルの直接書き込み

プライマリ実行ホスト上のMoMが最終宛先にアクセスできる場合、このMoMはその宛先にジョブの標準出力と標準エラーファイルを書き込むことができます。アクセス可能にするには、最終宛先ホストおよびパスが実行ホスト上にあるか、MoMの設定ファイル内の$usecp指示文でプライマリ実行ホストからマップされる必要があります。標準出力や標準エラーを直接その最終宛先に書き込むように指定するには、qsubまたはqalterの-kオプションにdサブオプションを使用します。eサブオプションまたはoサブオプションを使用して、どちらのファイルを書き込むかを指定します。

たとえば、出力とエラーの両方をそれらの最終宛先に直接書き込むようにするには、以下のように指定します。

```
qsub -koed -o <出力パス> -e <エラーパス> job.sh
```

出力を直接その最終宛先に書き込み、エラーは通常のスプーリングおよびステージングを通るようにするには、以下のように指定します。

```
qsub -kod -o <出力パス> job.sh
```

## 3.3.7 Linuxジョブのumaskの変更

Linuxでは、ジョブがファイルやディレクトリを実行ホストにステージングまたはコピーするか、実行ホスト上でstdoutまたはstderrを書き込むたびに、MoMがumaskを使用してそのファイルまたはディレクトリのアクセス権を決定します。umaskに値が指定されていない場合、MoMはシステムデフォルトを使用します。値は、以下の方法で指定できます。

- qsub -W umask=<値>を使用
- #PBS umask=<値>を使用

ジョブスクリプトでファイルまたはディレクトリを作成する場合、この方法は使用できません。

次の例は、umaskを*022*に設定して、オーナーのみが書き込みアクセス許可を持つようにファイルを作成します。希望するアクセス許可は、-rw-r--r--です。

```
qsub -W umask=022 my_job
#PBS -W umask=022
```

### 3.3.7.1 注意事項

この機能は、Windowsには適用されません。

## 3.3.8 ファイル送達のトラブルシューティング

ファイル送達は、実行ホスト上でMoMによって処理されます。ファイル送達の機能については、『PBS Professional Administrator's Guide』の「Setting File Transfer Mechanism」（441ページ）をご参照ください。

ファイル送達のトラブルシューティングについては、『PBS Professional Administrator's Guide』の「Troubleshooting File Transfer」（446ページ）をご参照ください。

### 3.3.8.1 出力結果が送信されない

ジョブの出力結果をユーザーに送達できない場合、出力結果は`PBS_HOME/undelivered`という名前の特別なディレクトリに保存され、ユーザーには通知メールが送信されます。送信の失敗には、一般に以下のような原因が考えられます。

1. 宛先ホストが信頼されておらず、ユーザーが`.rhosts`ファイルを持っていない。
2. 指定されたパスが不適切。
3. 指定の宛先パス内のディレクトリへの書込みが不可能。
4. 宛先ホスト上のユーザーの`.cshrc`が、実行時に出力を生成する。
5. `pbs.conf`内のPBS_SCPに指定されたパスが不正。
6. 実行ホスト上の `PBS_HOME/spool` ディレクトリに、適切なアクセス許可が与えられていない。このディレクトリは、モード *1777 drwxrwxrwxt*（Linuxの場合）、または“Everyone”に対して“フルコントロール”（Windowsの場合）である必要があります。

## 3.3.9 出力ファイルとエラーファイルの注意事項

### 3.3.9.1 実行ホストでのファイルの保持

PBSがジョブ固有のステージングおよび実行ディレクトリを作成し、ユーザーが`qsub`の-kオプションを使用するか、Keep_Files属性で*o*や*e*を指定する場合、ユーザーが実行ホスト上に保存するように要求したファイルは、ジョブ終了時にジョブ固有のステージングおよび実行ディレクトリが削除される際に、削除されます。

### 3.3.9.2 ジョブの再実行時に追加される標準出力と標準エラー

ジョブを実行して`stdout`または`stderr`に書き込んだ後に再実行すると、別のジョブが同じ名前で実行されているとみなされ、PBSでは２回目の実行時の`stdout`が初回のそれに追加され、２回目の実行時の`stderr`が初回のそれに追加されます。

### 3.3.9.3 Windowsドライブマップと PBS

Windowsでは、ドライブをマップする際、ユーザーのセッションにローカルにマップされます。マップされたドライブは、ユーザーのセッション外の他のプロセスによって見ることはできません。１つのセッションにマップされたドライブは、同じユーザーであっても、他のセッション内でマップ解除できません。これは、PBSでのジョブ実行に影響します。特に、あるドライブをマップし、マップしたドライブに移動してジョブを投入する場合、ジョブを実行する`vnode`は、`qsub`を発行した同じ場所にファイルを送り戻すことはできません。回避策として、PBSにファイルをローカルのマップされていないディレクトリに送達するように指示します。`qsub`に“`-o`”または“`-e`”オプションを使用し、ジョブ出力とエラーファイル用にディレクトリの場所を指定します。詳細については、46ページの第3.3.2節「出力ファイルとエラーファイルのパス」をご参照ください。

### 3.3.9.4 安全性には問題のないcshエラーメッセージ

ユーザーのログインシェルがcshである場合、ジョブの標準出力に次のメッセージが表示されることがあります。

```
Warning: no access to tty, thus no job control in this shell
```

多くのcshバージョンで、入力が端末ではないとシェルが判断した場合にこのメッセージが生成されます。cshを修正する以外に、このメッセージを抑制する方法はありません。ただし、単にこのようなメッセージが表示されるだけで、ジョブには何の影響もありません。

### 3.3.9.5 インタラクティブジョブとファイルI/O

インタラクティブジョブの終了時に、stdoutやstderrがまだコピーされていない可能性があります。

### 3.3.9.6 必要な書き込みアクセス許可

- stdoutまたはstderrのコピー先のディレクトリの書き込みアクセス許可が必要です。
- rootはPBS_HOME/spoolに書き込み可能である必要があります。

# 4 ジョブへのリソースの割り当てとジョブの配置

## 4.1 vnodeとは

仮想ノード、またはvnodeとは、マシンの利用可能なリソースの集合を表した概念上のオブジェクトです。vnodeは、１ホスト全体を指す場合があります。また、１ノードボードあるいは１ブレードである場合も考えられます。１ノードボードあるいは１ブレードを指す場合は、通常、複数のvnodeの集合が１ホストとなります。

ホストは任意のコンピュータです。実行ホストは以前はノードと呼ばれていました。いまでもPBSドキュメント以外では、ノードと呼ばれることがよくあります。PBSでは、ホストは１つまたは複数のvnodeで構成されているものとみなします。

PBSでは、各vnodeは個別に管理およびスケジューリングされます。ジョブは１つまたは複数のvnode上で実行されます。各vnodeには、固有の属性があります。『PBS Professional Reference Guide』の「Vnode Attributes」（320ページ）をご参照ください。

### 4.1.1 非推奨のvnodeタイプ

すべてのvnodeが同等に扱われ、“タイムシェアドノード”と呼ばれていたノードと同じように処理されます。“time-shared”と“cluster”のタイプは推奨されません。:ts接尾辞は推奨されません。この接尾語は無視され、変換中に削除されます。

vnode属性のntypeは、PBSノードとGlobus vnodeを区別する場合に限り使用されていました。GlobusからPBSにジョブを送信することはできますが、PBSからGlobusにジョブを送信することはできません。ntype属性は読み取り専用です。

## 4.2 PBSのリソース

### 4.2.1 PBSリソースの概要

本節「PBSリソースの概要」では、PBSリソースの基本について簡単に説明します。詳細については、『PBS Professional Administrator's Guide』の「Using PBS Resources」（227ページ）の特に第5.4節と第5.5節をご参照ください。各PBSリソースの詳細については、259ページの第5章「List of Built-in Resources」をご参照ください。

PBSリソースは、CPU、メモリ、アプリケーションライセンス、スイッチ、スクラッチスペース、時間などを表します。また、マシンが特定のプロジェクト専用かなど、ある事項がtrueであるかどうかも表します。

PBSでは複数の組み込みリソースが提供され、管理者は追加のカスタムリソースを定義できます。カスタムリソースはアプリケーションライセンスやスクラッチスペースなどに使用され、管理者によって定義されます。カスタムリソースは組み込みリソースと同じ方法で使用されます。PBSには次のタイプのリソースが用意されています。

### Boolean

ブーリアンリソースの名前は文字列です。

値：

*TRUE*、*True*、*true*、*T*、*t*、*Y*、*y*、*1*

*FALSE*、*False*、*false*、*F*、*f*、*N*、*n*、*0*

### Duration

期間。次のいずれかで表されます。

*秒単位の整数*

または

*[[hours:]minutes:]seconds[.milliseconds]*

形式は以下のとおりです。

*[[[HH]HH:]MM:]SS[.milliseconds]*

ミリ秒は秒に四捨五入されます。

### Float

浮動小数点。指定可能な値：[+-] 0-9 [[0-9] ...][.][[0-9] ...]

### Long

長整数。指定可能な値：0-9 [[0-9] ...]、+、-

*<キュー名>@<サーバー名>*

### Size

バイトまたはワード数。ワードのサイズは64ビットです。

フォーマット：*<整数>[<接尾辞>]*

*接尾辞*は、次のいずれかを使用できます。

**表4-1：バイト単位のサイズ**

| 接尾辞 | 意味 | サイズ |
|---|---|---|
| bまたはw | バイトまたはワード | 1 |
| kbまたはkw | キロバイトまたはキロワード | 2の10乗（1024） |
| mbまたはmw | メガバイトまたはメガワード | 2の20乗（1,048,576） |
| gbまたはgw | ギガバイトまたはギガワード | 2の30乗（1,073,741,824） |
| tbまたはtw | テラバイトまたはテラワード | 2の40乗（1024ギガバイト） |
| pbまたはpw | ペタバイトまたはペタワード | 2の50乗（1,048,576ギガバイト） |

デフォルト：*バイト*

スケジューラは、sizeタイプのすべてのリソースの端数を切り上げて最も近いKBにします。

### 文字列

スペース文字を含む任意の文字。

どの値にも、2つの引用符文字タイプ"または'のうち1つのみを使用できます。

値：[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ´ ( ) * + , - ./ : ; < = > ? @ [ \ ] ^ _ ' { | } ~] ...]

文字列リソースの値は大文字と小文字が区別されます。長さに制限はありません。

### String Array

文字列のコンマで区切られたリスト。

string_array内の文字列にはコンマが含まれない場合があります。長さに制限はありません。

Pythonタイプは*str*です。

値が1つの文字列アレイリソースは、文字列リソースとまったく同じように機能します。

組み込みリソースのリストについては、『PBS Professional Reference Guide』の「Resources Built Into PBS」（265ページ）をご参照ください。

一部のシステムでは、PBSで特殊なカスタムリソースが作成されます。

管理者は、サーバー、各キュー、および各vnode上でどのリソースを使用可能にするかを指定できます。キューまたはサーバーレベルで定義されたリソースはジョブ全体に適用されます。vnodeレベルで定義されたリソースは、そのvnodeで実行されるジョブの部分にしか適用されません。

ジョブはリソースを要求できます。スケジューラは、管理者によって定義された規則に従って、要求されたリソースと使用可能リソースを一致させます。PBSは、必ず、ジョブから要求されたリソースが見つかった場所にジョブを配置します。PBSでは、リソースが不足する可能性のある場所にジョブは配置されません。たとえば、それぞれCPUを1つ要求する2つのジョブがあり、CPUを1つ備えたvnodeが1つある場合、そのvnode上ではこれらのジョブのうち一度に1つのみが実行されます。

PBSは、ジョブ別のリソース使用の制限を強制できます。66ページの第4.5節「リソース使用量の制限」をご参照ください。

## 4.2.2 用語集

### チャンク

チャンクは、ジョブに割り当てられるリソースの単位です。これは、-l select書式で指定されます。各チャンクに指定されたリソースは、チャンクごとに同じホストに割り当てられます。通常MPIジョブの実行には、ジョブ内の1プロセスに対して1チャンクを用いてリソースを指定します。

### チャンクレベルリソース、ホストレベルリソース

CPUやメモリなど、ホストレベルで使用可能なリソース。チャンクリソースは、-l select内部で要求されます。チャンクのリソースは、そのチャンクで動作するジョブの部分に適用されます。

チャンクリソースは、-l select内部で要求されます。

### ジョブワイドのリソース、サーバーリソース、キューリソース

ジョブワイドのリソースは、サーバーレベルまたはキューレベルのリソースとも呼ばれ、サーバーまたはキューでジョブ全体から使用可能なリソースです。

ジョブワイドのリソースは、サーバーまたはキューのresources_available.<リソース名>属性を使用可能な値または照合する値に設定した場合に、サーバーで消費または照合できます。たとえば、*FloatingLicenses*と呼ばれるカスタムリソースを定義し、サーバーのresources_available.FloatingLicenses属性を、使用可能なフローティングライセンスの数に設定します。

ジョブワイドのリソースには、共有スクラッチスペース、アプリケーションライセンス、walltimeなどがあります。

ジョブでは、ジョブ全体でジョブワイドのリソースを要求できますが、個々のチャンクでは要求できません。

# 4.3 リソースの要求

ジョブでは、ジョブ全体に適用されるリソースまたはジョブチャンクに適用されるリソースを要求できます。たとえば、ジョブ全体でアプリケーションライセンスが必要な場合、ジョブはジョブワイドのライセンスを１つ要求できます。一方、１つのジョブプロセスが２つのCPUを必要とし、別のジョブプロセスが８つのCPUを必要とする場合、ジョブとしては２つのチャンク（２つのCPUのチャンクと８つのCPUのチャンク）を要求できます。ジョブは、ジョブワイドのリクエストとチャンクレベルリクエストで同じリソースを要求することはできません。

PBSは、ジョブワイドのリソースとしてのみ使用できるwalltimeなどのリソースおよびチャンクリソースとしてのみ使用できるncpusやmemなどのリソースを提供します。リソースはジョブワイドまたはチャンクレベルのどちらかであり、両方ではありません。各リソースの説明には、リソースをどちらの方法で使用するかが記述されています。『PBS Professional Reference Guide』の「List of Built-in Resources」（259ページ）をご参照ください。

リソースの要求の詳細については、57ページの第4.3.2節「ジョブワイドのリソースの要求」および57ページの第4.3.3節「チャンク内のリソースの要求」をご参照ください。

## 4.3.1 リソースの要求の一覧

ジョブワイドのリソースは、<*リソース名*>=<*値*>の組み合わせで要求します。ジョブワイドのリソースは、以下のどちらかの方法で要求できます。

- `qsub -l` <リソース名>=<値>オプション

  複数のリソースは、以下のどちらかの形式で要求できます。

  **`-l <リソース>=<値>,<リソース>=<値>`**

  **`-l <リソース>=<値> -l <リソース>=<値>`**

- １つまたは複数の`#PBS -l` <リソース>=<値>指示文

チャンクリソースは、*-l selectのチャンク指定*で要求します。チャンクリソースは、以下のどちらかの方法で要求できます。

- `qsub -l select=[N:][`<チャンク指定>`][+[N:]`<チャンク指定>`]` オプション
- `#PBS -l select=[N:][`<チャンク指定>`][+[N:]`<チャンク指定>`]` 指示文

ジョブワイドのリソースとチャンクリソースの両方を要求する形式は、以下のとおりです。

**`qsub`** ...（ジョブのリソース以外の部分）
**`-l <リソース>=<値>`**　（ジョブワイドのリクエスト）
**`-l select=<チャンク>[+<チャンク>]`**　（-l select）

PBSは、リソースの要求またはリソース要求の変更に使用できるさまざまなコマンドを提供します。

- qsubコマンド（コマンドラインまたはPBS指示文の両方）
- pbs_rsubコマンド（コマンドラインのみ）
- qalterコマンド（コマンドラインのみ）

## 4.3.2 ジョブワイドのリソースの要求

ジョブは、*ジョブワイド*のリソース要求で、ジョブ全体に適用されるリソースを要求できます。ジョブワイドのリソースは、ジョブ全体で使用するように設計されており、サーバーまたはキューでは使用できますが、ホストレベルでは使用できません。ジョブワイドのリソースは、フローティングアプリケーションライセンスやその他の特定のvnodeに関連していないリソース（cput、walltimeなど）を要求する際に使用されます。

ジョブワイドのリソースは、以下のように-l select外部で要求されます。

*-l <リソース名>=<値>[,<リソース名>=<値> ...]*

“-l select外部”のリソース要求では、リソース要求が“-lselect=”の後ではなく、“-l”の後に出現します。言い換えれば、チャンクでジョブワイドのリソースを要求することはできません。

たとえば、ジョブで1時間のwalltimeを要求するには、以下のように指定します。

*-l walltime=1:00:00*

ジョブワイドのリソースは、以下のどちらかの方法で要求できます。

- qsub -l <リソース名>=<値>オプション

  複数のリソースは、以下のどちらかの形式で要求できます。

  **-l <リソース>=<値>,<リソース>=<値>**

  **-l <リソース>=<値> -l <リソース>=<値>**
- １つまたは複数の#PBS -l <リソース>=<値>指示文

## 4.3.3 チャンク内のリソースの要求

*チャンク*は、各リソースの値をリソースセットで指定します。リソースセットは、ジョブにユニットとして割り当てられます。また、チャンクはジョブに割り当てられる最小のリソースセットです。すべてのチャンクが１台のホストから適用されます。１つのチャンクを複数のvnodeに分割できますが、それらすべてのvnodeが同じホストに属する必要があります。

ジョブはチャンクリソースを要求できます。チャンクリソースは、ジョブのホストレベルの部分に適用されるリソースです。ホストレベルのリソースはチャンクの一部としてのみ要求できます。サーバーまたはキューリソースはチャンクの一部としては要求できません。チャンクリソースは、ジョブのうちそのチャンクで動作する部分によって使用され、ホストレベルで使用できます。チャンクは、CPU、メモリ、アーキテクチャなど、ホスト関連のリソースを要求する場合に使用します。

チャンクリソースは、-l select内部で要求されます。-l selectの形式は以下のとおりです。

*-l select=[N:]<チャンク>[+[N:]<チャンク> ...]*

ここで、詳細について説明します。１つのチャンクを要求するには、以下の形式を使用します。

*-l select=<リソース名>=<値>[:<リソース名>=<値>...]*

たとえば、１つのチャンクで２つのCPUと4GBのメモリを持つことができます。

```
-l select=ncpus=2:mem=4gb
```

特定のチャンクを複数要求するには、チャンク指定の前にチャンク数を指定します。

*-l select=[<チャンク数>]<チャンク指定>*

たとえば、前の例のチャンクを６つ要求するには、以下のように指定します。

```
-l select=6:ncpus=2:mem=4gb
```

*N*を指定しない場合、*1*とみなされます。

異なるチャンクを要求する場合、それらのチャンクをプラス記号（“+”）で連結します。

*-l select=[<チャンク数>]<チャンク指定>+[<チャンク数>]<チャンク指定>*

たとえば、２セットのチャンクを要求するとします。１つは２つのCPUを持つチャンクが６つのセット、もう１つは８つのCPUを持つチャンクが３つのセットであり、さらにどちらのセットもチャンク１つにつき4GBのメモリがある場合、以下のように指定します。

```
-l select=6:ncpus=2:mem=4gb+3:ncpus=8:mem=4GB
```

チャンク間で空白は使用できません。

すべてのチャンクを１つの-l selectで指定する必要があります。

チャンクリソースは、以下のどちらかの方法で要求できます。

- `qsub -l select=[N:][<チャンク指定>][+[N:]<チャンク指定>]` オプション
- `#PBS -l select=[N:][<チャンク指定>][+[N:]<チャンク指定>]` 指示文

## 4.3.4 ブーリアンリソースの要求

リソース要求では、ブーリアンリソースが*True*か*False*かについて指定できます。

例4-1： green=*True*とred=*True*のvnodeが複数あり、それぞれ１個のCPUを装備し、redを除外してgreenだけのvnodeを２つ要求する場合：

```
-l select=2:green=true:red=false:ncpus=1
```

例4-2： 次のジョブスクリプトの一部は、walltime（経過時間）に対するジョブワイドの要求およびブーリアンリソースHasMyAppが*True*であるCPUとメモリに対するチャンク要求です。

```
#PBS -l walltime=1:00:00
#PBS -l select=ncpus=4:mem=400mb:HasMyApp=true
```

vnodeレベルのブーリアン要求とジョブワイドのブーリアン要求の違いにご留意ください。

```
qsub -l select=1:green=True
```

は、greenが*True*に設定されているvnodeを要求します。一方、

```
qsub -l green=True
```

はブーリアンリソース、greenが*True*に設定されているサーバーやキューを要求します。

## 4.3.5 アプリケーションライセンスの要求

アプリケーションライセンスは、PBS 管理者が定義するリソースとして管理されます。実際には PBS はライセンスをチェックアウトしないで、ジョブのセッション内で実行されるアプリケーションが行います。

### 4.3.5.1 フローティングアプリケーションライセンスの要求

サイト全体のフローティングライセンスは、一般的にジョブワイドのサーバーレベルリソースとして構成されます。

AppF という名前のジョブワイドのアプリケーションライセンスを要求するには、以下を使用します。

```
qsub -l AppF=<ライセンス数> <他のqsub引数>
```

特定のホストのみがそのアプリケーションを実行できる場合、一般的にそのホストではホストレベルのブーリアンリソースが *True* に設定されています。

ジョブワイドのリソース AppF は、サイトで使用可能なライセンス数を示す数値リソースです。ホストレベルブーリアンリソース haveAppF は、指定されたホストがアプリケーションを実行できるかどうかを示します。アプリケーションライセンスおよびアプリケーションを実行する vnode を要求するには、以下を使用します。

```
qsub  -l AppF=<ライセンス数> <他のqsub引数>
      -l select=haveAppF=True
```

PBS は、スケジューリングサイクルが開始されるごとに利用可能なフローティングライセンスの数をサーバーに照会します。実際には PBS はライセンスをチェックアウトしないで、ジョブのセッション内で実行されるアプリケーションが行います。

### 4.3.5.2 ノードロックアプリケーションライセンスの要求

ノードロックアプリケーションライセンスは、アプリケーションをライセンスされた vnode（複数可）で使用できます。これらは、-l select 内部で要求されるホストレベル（チャンク）リソースです。

#### 4.3.5.2.i ホスト別ノードロックアプリケーションライセンスの要求

ホスト別ノードロックアプリケーションライセンスは、通常は、必要なライセンスがそのホストで使用可能かどうかを示すブーリアンリソースとして構成されます。

ブーリアン値のホスト別ノードロックライセンスを要求するときは、ホスト 1 つにつき 1 つを要求します。フォーマット：

*qsub -l select=<ブーリアンリソース名>=true:<残りのチャンク指定>*

例 4-3： ブーリアンリソース runsAppA は、この vnode に必要なライセンスがあるかどうかを示します。1 つのチャンクで AppA 用のホスト別ノードロックライセンスを使用してホストを要求するには、以下のように行います。

```
qsub -l select=1:runsAppA=1 <ジョブスクリプト>
```

#### 4.3.5.2.ii 使用別ノードロックアプリケーションライセンスの要求

ホスト別ノードロックアプリケーションライセンスは、アプリケーションを実行するホストが一度に使用可能なライセンスの数を持つように、消費可能な数値リソースとして構成されます。

数値の使用別ノードロックライセンスを要求するときは、ホストごとに必要な数のライセンスを要求します。

*qsub -l select=<消費可能リソース名>=<必要な量>:<残りのチャンク指定>*

例 4-4： AppB という名前の消費可能リソースは、ホスト上で使用可能な使用別アプリケーションライセンスの数を示します。2 つの CPU 上で AppB のインスタンスを 1 つ実行する場合で、1 つのチャンクで AppB 用のホスト別ノードロックライセンスを使用してホストを要求するには、以下のように行います。

```
qsub -l select=1:ncpus=2:AppB=1
```

#### 4.3.5.2.iii CPU別ノードロックアプリケーションライセンスの要求

CPU別ノードロックライセンスは、通常、ライセンスされたCPUごとにホストがライセンスを１つ持つように配置されます。PBS管理者は、使用可能なライセンス数を示す消費可能な数値リソースを構成します。

CPUごとにライセンスを１つ要求する必要があります。数値の使用別ノードロックライセンスを要求するときは、ホストごとに必要な数のライセンスを要求します。

*qsub -l select=<CPU別リソース名>=<必要な量>:<残りのチャンク指定>*

例4-5： AppCという名前の数値の消費可能リソースは、使用可能なCPU別ライセンスの数を示します。2つのCPUを使用するジョブを実行する場合で、1つのチャンクでAppC用のCPU別ノードロックライセンスを使用してホストを要求するには、以下のように行います。

```
qsub -l select=1:ncpus=2:AppC=2
```

## 4.3.6 スクラッチスペースの要求

マシン上のスクラッチスペースはホストレベルの動的リソースとして構成されます。スクラッチスペースリソースの名前については、管理者に確認します。

スクラッチスペースを要求するには、チャンク要求にこのリソースを追加します。

*-l select=<スクラッチリソース名>=<必要なスクラッチの量>:<残りのチャンク指定>*

例4-6： 管理者はスクラッチリソースに“dynscratch”という名前を付けています。１つのチャンクで10MBのスクラッチスペースを要求する場合、以下のように行います。

```
-l select=1:ncpus=N:dynscratch=10MB
```

## 4.3.7 GPUの要求

PBSジョブではGPUを要求できます。GPUをどのように要求するかは、PBSがGPUを管理するためにcgroupを使用するかどうかにより異なります。管理者にご確認ください。

### 4.3.7.1 cgroupを介して管理されるGPUの要求

推奨：Linuxにおいてのみ、cgroupを使用してGPUを排除するようにPBSを設定し、ジョブがGPUを要求したときに自動的にそのGPUを排他的に使用するようにすることができます。専有性を要求する必要はありません。PBSがcgroupを使用してGPUを管理する場合、ngpusリソースを介して必要なGPUの数を要求します。

*qsub -l select=ngpus=<値>:<残りのチャンク指定>*

GPUがcgroupを介して管理される場合、メモリを要求するジョブは、その量を物理メモリとスワップの両方に使用します。たとえば、20GBを要求し、16GBを使用しているジョブが、50GBのファイルを読み込む場合、一度に4GBしかスワップできません。したがって、ジョブが32GBのアプリケーションメモリを必要としているが、適切に実行するには5GBのプライベートファイルキャッシュも必要である場合、そのジョブは37GBを要求する必要があります。

## 4.3.7.2 cgroupを介して管理されないGPUの要求

WindowsまたはLinuxで、PBSがcgroupを使用せずにGPUを管理している場合、管理者は、以下の動作をサポートするようにPBSを設定できます。

- (“基本的なGPUスケジューリング”) ジョブが不特定のGPUを使用し、ノードを排他的に使用する
- (“高度なGPUスケジューリング”) ジョブが不特定のGPUを使用し、ノードを共有して使用する
- (“高度なGPUスケジューリング”) ジョブが特定のGPUを使用し、ノードを共有するか排他的に使用する

### 4.3.7.2.i GPUへのバインド

PBS ProfessionalではGPUを割り当てますが、ジョブを特定のGPUにはバインドしません。実際には、アプリケーション自体またはCUDAライブラリで実際のバインドを行います。

### 4.3.7.2.ii 不特定のGPUの要求およびノードの排他的な使用

サイトで“基本的なGPUスケジューリング”を使用しているとき、ジョブにGPUは必要だが特定のGPUは必要ではなく、GPUノードの排他的な使用を要求できる場合、CPUの要求と同じ方法でGPUを要求することが可能です。

管理者は、ノード上のGPUを示すようにリソースを設定できます。推奨されるGPUリソース名は*ngpus*です。

このようにGPUを要求する場合は、ジョブがノードを排他的に使用できるように要求して、他のジョブが当該GPU上にスケジュールされるのを回避する必要があります。

*qsub -l select=ngpus=<値>:<残りのチャンク指定> -lplace=excl*

例4-7: 2つのGPUと1つのCPUを備えた1つのノードを要求し、ノードを排他的に使用する“*my_gpu_job*”という名前のジョブを投入する場合は、以下のように行います。

```
qsub -lselect=1:ncpus=1:ngpus=2 -lplace=excl my_gpu_job
```

アプリケーションまたはCUDAによって、GPUとアプリケーションプロセスがバインドされます。

### 4.3.7.2.iii 不特定のGPUの要求およびノードの共有使用

サイトで“高度なGPUスケジューリング”を使用している場合、管理者は、ジョブがノード上の不特定のGPUを使用して、GPUノードを共有できるようにPBSを設定することが可能です。この場合、管理者は各GPUをそれぞれ独自のvnodeに配置します。

リソースがGPUを示すように設定することが可能です。推奨されるGPUリソース名は*ngpus*です。

管理者は、それぞれのリソースにGPUのデバイス番号が含まれるよう、各GPU vnodeを設定することが可能です。推奨されるリソース名は*gpu_id*です。

例4-8: 2つのGPUと1つのCPUを要求し、ノードを共有して使用する“*my_gpu_job*”という名前のジョブを投入する場合は、以下のように入力します。

```
qsub -lselect=1:ncpus=1:ngpus=2 -lplace=shared my_gpu_job
```

任意のGPUを要求するジョブが投入されると、PBSスケジューラでは使用可能なGPUを備えたvnodeを探して、該当するvnodeをそのジョブに割り当てます。GPUとvnodeは1対1で対応しているため、ジョブは該当するvnodeのgpu_idを特定できます。これで、アプリケーションは適切なCUDA呼び出しを使用して、割り当てられたGPUにプロセスをバインドできます。

### 4.3.7.2.iv 特定のGPUの要求

サイトで“高度なGPUスケジューリング”を使用している場合、ジョブでは特定のGPUを1つまたは複数要求できます。これにより、特定のGPU上でそのGPUのために書かれたアプリケーションを実行できます。

管理者は、ジョブが特定のGPUを要求できるようにリソースを設定することが可能です。推奨されるGPUリソース名は*gpu_id*です。

特定のGPUを要求する場合、チャンクごとに使用するGPUを指定します。

*qsub -l select=gpu_id=<GPU ID>:<残りのチャンク指定>*

例4-9： それぞれがID *0*のGPUを備えた４つのvnodeを要求するには、以下のように行います。

```
qsub -lselect=4:ncpus=1:ngpus=1:gpu_id=gpu0 my_gpu_job
```

特定のGPUを要求するジョブが投入されると、PBSスケジューラでは、該当するgpu_idがリソースに含まれるvnodeをジョブに割り当てます。アプリケーションは適切なCUDA呼び出しを使用して、割り当てられたGPUにプロセスをバインドできます。

#### 4.3.7.3 ノードのGPU情報の表示

pbsnodesコマンドを使用すると、実行ホストに割り当てられている使用可能なGPUの数を調べることができます。68ページの第4.6節「リソースの表示」をご参照ください。

## 4.3.8 リソース要求に関する注意事項と制限事項

### 4.3.8.1 リソース値の指定の注意事項と制限事項

- リソース値にコンマ、引用符、正記号、等記号、コロン、または括弧が含まれている場合、PBSでは引用符で囲まれている必要があります。コマンド（qsub、qalterなど）で文字列を正しく解釈できるように、文字列を引用符で囲む必要があります。
- コマンドラインからリソースを指定する場合、引用符で囲まれた文字列はエスケープまたは別の引用符のセットで囲む必要があります。この２番目の引用符セットは、最初のセットとは異なる必要があります。つまり、二重引用符は一重引用符で囲み、一重引用符は二重引用符で囲む必要があります。
- 文字列リソース値に空白やシェルのメタキャラクターが含まれている場合、文字列を引用符で囲むか、空白とメタキャラクターをエスケープします。必ず、シェルおよび必要な動作に適した正しい引用符を使用してください。

### 4.3.8.2 walltimeを要求しないことに関する警告

ジョブでwalltimeを要求せず、walltimeのデフォルトも存在しない場合、ジョブは非常に長い時間を要求した場合と同様に扱われます。言い換えると、スケジューラがジョブに時間スロットを見つけるのが困難な状況になります。管理者は、アップグレードの場合など、１年に１回、PBSコンプレックス全体に専用時間をスケジュールする可能性があります。この場合、上記のジョブは決して実行されません。ジョブで合理的なwalltimeを要求することをお勧めします。

### 4.3.8.3 未定義のリソースを要求するジョブの注意事項

ジョブワイドまたはホストレベルのリソースを要求するジョブを投入し、そのリソースが未定義である場合、ジョブが投入時に拒否されることはありません。代わりに、実行キューに追加されるときにリソースが依然として未定義だった場合は、ジョブが中止されます。これは、ピアスケジューリングにより移動された先のサーバーにリソースが定義されている可能性を考慮したものです。これにより、下位互換性が確保されます。

### 4.3.8.4 リソース要求と未設定リソースとの一致

ジョブのリソース要求を使用可能なリソースと照合する際、ホスト上で未設定の数値リソースはゼロとして処理されます。未設定の文字列は要求を満たすことができません。未設定のブーリアンリソースは、“*False*” に設定された場合と同様に扱われます。サーバーまたはキューの未設定のリソースは無限とみなされます。

#### 4.3.8.5 非表示または要求不可のリソースの注意事項

管理者が制限付きのカスタムリソースを定義する可能性があり、そのようなリソースは非表示か、または表示されても要求不可です。非表示または要求不可の設定で作成されたカスタムリソースは要求または変更できません。リソースの表示や要求またはリソース要求の変更に通常使用するコマンドとその制限付きリソースに対する制限事項のリストを、以下に示します。

`pbsnodes`

ジョブ投入者は制限付きのホストレベルのカスタムリソースを表示できません。

`pbs_rstat`

ジョブ投入者は制限付きの予約リソースを表示できません。

`pbs_rsub`

ジョブ投入者は予約に制限付きのカスタムリソースを要求できません。

`qalter`

ジョブ投入者は制限付きのリソースを変更できません。

`qmgr`

ジョブ投入者は制限付きのリソースを出力または一覧表示できません。

`qselect`

ジョブ投入者は制限付きのリソースを`-l Resource_List.`で指定できません。

`qsub`

ジョブ投入者は制限付きのリソースを要求できません。

`qstat`

ジョブ投入者は制限付きのリソースを表示できません。

#### 4.3.8.6 わずかな量のメモリを要求することに関する警告

要求できるメモリの最小単位は1KBです。400バイトを要求すると、1KBを取得します。1400バイトを要求すると、2KBを取得します。

#### 4.3.8.7 ジョブ投入コマンドラインの最大長

PBSでは、コマンドラインの最大長は4095文字です。コマンドラインでジョブを投入する場合、selectおよびplaceステートメントおよびコマンドラインの残りの部分を含めて4095文字以内に収める必要があります。

#### 4.3.8.8 ジョブごとに-l selectは１つのみ

１つのジョブ投入で指定できる-l selectは最大１つです。

#### 4.3.8.9 softwareリソースはジョブワイド

組み込みリソース"software"はvnodeレベルのリソースではありません。『PBS Professional Reference Guide』の「Resources Built Into PBS」（265ページ）をご参照ください。

#### 4.3.8.10 古い構文と新しい構文の混在の防止

リソースを要求するときに、新しい構文と古い構文を混在させないでください。古い構文の説明については、75ページの第4.8節「下位互換性」をご参照ください。

# 4.4 ジョブへのリソースの割り当て方法

ジョブへのリソースの割り当ては、ジョブが明示的にリソース要求したとき、およびPBSがデフォルトを適用するときに、行われます。

ジョブでは、-l select内で定義されたチャンクの中のvnodeレベル、またはジョブワイドのリソース要求のいずれかでリソースが明示的に要求されます。リソースの要求については、57ページの第4.3.3節「チャンク内のリソースの要求」および57ページの第4.3.2節「ジョブワイドのリソースの要求」をご参照ください。

管理者はサーバーおよびキューにデフォルトリソースを設定できるので、投入時にリソースを要求しないジョブは、そのリソースのデフォルト値が割り当てられます。デフォルトリソースについては、64ページの第4.4.1節「デフォルトリソースの適用」をご参照ください。

管理者は、ジョブが自動的に特定のリソースを要求するように、qsubのデフォルト引数を指定することもできます。ジョブが明示的に要求したリソース値は、qsubのデフォルトより優先されます。『PBS Professional Reference Guide』の「qsub」（216ページ）をご参照ください。

## 4.4.1 デフォルトリソースの適用

PBSは、ジョブが明示的にリソースの値を要求しなかった場合のみリソースのデフォルト値を適用します。

ジョブワイドまたはチャンクごとのリソースは、以下の優先順序で、以下の方法で適用されます。

1. -l <resource>=<値>および-l select=<チャンク>で明示的に要求されたリソース
2. デフォルトのqsub引数
3. キューのdefault_chunk.<リソース>
4. サーバーのdefault_chunk.<リソース>
5. キューのresources_default.<リソース>
6. サーバーのresources_default.<リソース>
7. キューのresources_max.<リソース>
8. サーバーのresources_max.<リソース>

### 4.4.1.1 ジョブワイドのデフォルトリソースの適用

明示的なジョブワイドのリソース要求は、まずqsubのデフォルト引数に対してチェックされ、次にキューリソースのデフォルトに対してチェックされ、その後、サーバーリソースのデフォルトに対してチェックされます。ジョブのリソース要求に存在していないデフォルトのジョブワイドのリソースがあれば、それが追加されます。PBSは、以下の場所で定義されているデフォルトのジョブワイドのリソースを、以下の順序で適用します。

- qsubを使用：サーバーのdefault_qsub_arguments属性に、任意の要求可能なジョブワイドのリソースを設定できます。
- キューを使用：各キューの resources_default 属性では、キューレベルのジョブワイドのリソースごとのデフォルト値をresources_default.<リソース>に定義します。
- サーバーを使用：各サーバーのresources_default属性では、サーバーレベルのジョブワイドのリソースごとのデフォルト値をresources_default.<リソース>に定義します。

### 4.4.1.2 チャンクごとのデフォルトリソースの適用

ジョブの-l select内のチャンクごとに、最初にqsubのデフォルトが、次にキューチャンクのデフォルトが、その後サーバーチャンクのデフォルトが適用されます。チャンク要求で、デフォルト内にリストされたリソースが含まれていない場合は、デフォルトが追加されます。PBSは、デフォルトのチャンクリソースを、以下の順序で適用します。

- qsubを使用：サーバーのdefault_qsub_arguments属性に、任意の要求可能なチャンクリソースを設定できます。
- キューを使用：各キューのdefault_chunk属性では、キューレベルのチャンクリソースごとのデフォルト値をdefault_chunk.<リソース>に定義します。
- サーバーを使用：各サーバーのdefault_chunk属性では、サーバーレベルのチャンクリソースごとのデフォルト値をdefault_chunk.<リソース>に定義します。

例4-10： チャンクデフォルトの適用：ジョブがエンキューされたキューに次のようなデフォルト定義がある場合を考えます。

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

以下の-l selectで投入されたジョブは、

```
select=2:ncpus=4+1:mem=9gb
```

default_chunk要素が適用された後に上記の指定が追加されます。

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb.
```

この例では、*mem=2gb*および*ncpus=1*がdefault_chunkから継承されます。

### 4.4.1.3 キュー間でのジョブの移動の注意事項

現在のキューから新しいキューにジョブが移動された場合、現在のキューが提供しているジョブのリソースリストにあるデフォルトリソースは削除されます。これには、古い構文から変換するためにルールによって生成されたselect指定およびplace指示文も含まれます。ジョブのリソースが設定（定義）されてなく、新しいキューまたはサーバーにデフォルト値が存在する場合、そのデフォルト値がジョブのリソースリストに適用されます。selectまたはplaceのいずれかがジョブの新しいリソースリストに存在しない場合、新しく継承されたデフォルト値を使用して自動的に生成されます。

以下のようなキューおよびサーバーのデフォルト値セットを考えます。

サーバー

```
resources_default.ncpus=1
```

キュー QA

```
resources_default.ncpus=2
default_chunk.mem=2gb
```

キュー QB

```
default_chunk.mem=1gb
```

ncpusのデフォルト値なし

以下の例では、キュー QAに投入され、その後キュー QBに移動される（または直接投入された）ジョブの同等のselect指定を示しています。

**qsub -l ncpus=1 -lmem=4gb**

QA：select=1:ncpus=1:mem=4gb

デフォルト値の適用不要

QB：select=1:ncpus=1:mem=4gb

デフォルト値の適用不要

**qsub -l ncpus=1**

QA：select=1:ncpus=1:mem=2gb

キューのチャンクのデフォルトから2gbを取得し、qsubから1 ncpusを取得

QB：select=1:ncpus=1:mem=1gb

キューのチャンクのデフォルトから1gbを取得し、qsubから1 ncpusを取得

**qsub -lmem=4gb**

QA：-l select=1:ncpus=2:mem=4gb

キューレベルのジョブワイドのリソースデフォルトから2 ncpusを取得し、qsubから4gb memを取得

QB：select=1:ncpus=1:mem=4gb

サーバーレベルのジョブワイドのデフォルトから1 ncpusを取得し、qsubから4gb memを取得

**qsub -lnodes=4**

QA：select=4:ncpus=1:mem=2gb

キューレベルのチャンクのデフォルトメモリを2gb取得（特に明示的に記載されていない場合は、前のバージョンでは"nodes=x"はノードあたり1CPUを意味したため、これは4:ncpus=2ではない）

QB：select=4:ncpus=1:mem=1gb

（特に明示的に記載されていない場合は、前のバージョンでは"nodes=x"はノードあたり1CPUを意味したため、ncpus=1はサーバーデフォルトから継承されない）

**qsub -l mem=16gb -lnodes=4**

QA：select=4:ncpus=1:mem=4gb

（特に明示的に記載されていない場合は、前のバージョンでは"nodes=x"はノードあたり1CPUを意味したため、これは4:ncpus=2ではない）

QB：select=4:ncpus=1:mem=4gb

（特に明示的に記載されていない場合は、前のバージョンでは"nodes=x"はノードあたり1CPUを意味したため、ncpus=1はサーバーデフォルトから継承されない）

# 4.5 リソース使用量の制限

ジョブは使用できるリソース量のリミットを割り当てられます。これらのリミットは、ジョブワイドで使用できる量（ジョブワイドのリミット）と各ホストでジョブが使用できる量（ホストリミット）に適用されます。リミットはジョブが要求または継承したリソースにのみ適用されます。

管理者は、memとncpusに対してリミットを強制するようにPBSを構成できますが、他のリミットは常に強制されます。

ジョブが使用するリソースが一定量を超過しないことを保証する必要がある場合は、その量のリソースを要求します。

### 4.5.1 強制可能なリソースリミット

以下のリソースには、リミットを強制できます。

**表4-2：強制可能なリソースリミット**

| リソース名 | 指定される場所 | 強制される場所 | 常に強制？ |
|---|---|---|---|
| cput | ホスト | ホスト | 常に強制 |
| mem | ホスト | ホスト | オプション |
| ncpus | ホスト | ホスト | オプション |
| pcput | ジョブワイド | プロセスごと | 常に強制 |
| pmem | ジョブワイド | プロセスごと | 常に強制 |
| pvmem | ジョブワイド | プロセスごと | 常に強制 |
| vmem | ホスト | ホスト | 常に強制 |
| walltime | ジョブワイド | ジョブワイド | 常に強制 |

### 4.5.2 リソースリミットの導出元

リミットは要求されたリソースと適用されたデフォルトリソースの両方から導出されます。リソースリミットの導出順序については、64ページの第4.4.1節「デフォルトリソースの適用」をご参照ください。

### 4.5.3 ジョブワイドのリソースリミット

ジョブワイドのリソースリミットは、ジョブごとのリソース使用量に制限を設定します。ジョブのリソースリミットは、ジョブワイドのリソースから、およびチャンクごとの消費可能リソース合計から、導出されます。リミットは、明示的に要求されたリソースおよびデフォルトリソースから導出されます。

チャンク全体の合計から導出されるジョブワイドのリソースリミットは、ジョブワイドのリソースから導出されるリソースリミットよりも優先されます。

例4-11： ジョブワイドのリミットは、チャンクの合計から導出されます。以下のチャンクが要求されたとします。

```
qsub -lselect=2:ncpus=3:mem=4gb:arch=linux
```

以下のジョブワイドのリミットが導出されます。

ncpus=6

mem=8gb

### 4.5.4 チャンクごとのリソースリミット

各チャンクのチャンク別制限により、そのホストで使用可能なリソースの量が決定されます。PBSは、各ホストのチャンクリミットを合計し、合計値をそのリソースのリミットとして使用します。チャンクあたりのリソース使用リミットとは、明示的な要求およびデフォルトの両方からジョブに割り当てられるチャンクあたりのリソース量です。

#### 4.5.4.1 リミットの影響

実行中のジョブがwalltimeの制限を超過すると、ジョブは終了します。

ジョブのいずれかのプロセスがpcput、pmem、またはpvmemの制限を超過すると、ジョブは終了します。

いずれかのホストのmem、ncpus、cput、またはvmemの制限を超過すると、ジョブは終了します。これらはホストレベルのリミットなので、たとえば、ジョブが１つのホストで２つのチャンクを使用し、一方のチャンクのプロセスがこれらのリミットのいずれかを超過し、もう一方のチャンクのプロセスはチャンクリミットを下回っている場合、ジョブは両方のチャンクでの合計使用量がホストリミットを下回っている限り、実行を継続できます。

### 4.5.5 メモリリミットの例

管理者がメモリリミットを強制することを選択する場合があります。この場合、ジョブ全体のメモリ使用量がResource_List.memで指定されている量を超過することはできません。また、いずれかのホストでのメモリ使用量がそのホストのチャンクの合計量を超過することはできません。この後に示す例では、以下を仮定します。

キューの設定は以下のとおりです。

```
resources_default.mem=200mb
default_chunk.mem=100mb
```

例4-12： `-l select=2:ncpus=1:mem=345mb`を要求するジョブは、２つの各vnodeから345mbを使用し、ジョブワイドの制限が690mb（2 * 345）となります。ジョブのResource_List.memには*690mb*が表示されます。

例4-13： `-l select=2:ncpus=2`を要求するジョブは、各vnodeの default_chunkから100mbを取得し、ジョブワイドの制限が200mb（2 * 100mb）となります。ジョブのResource_List.memには*200mb*が表示されます。

例4-14： `-l ncpus=2`を要求するジョブは、１つのvnodeから200mb（resources_defaultから継承されてselect指定の作成に使用される）を取得し、ジョブワイドの制限は200mbです。ジョブのResource_List.memには*200mb*が表示されます。

例4-15： `-lnodes=2`を要求するジョブはresources_default.memから200mbを継承し、これがジョブワイドの制限になります。メモリは、２つのvnodeからvnodeあたり半分（100mb）ずつ取得されます。生成されるselect指定は`2:ncpus=1:mem=100mb`です。ジョブのResource_List.memには*200mb*が表示されます。

## 4.6 リソースの表示

サーバー、キュー、およびvnode上のリソースを表示できます。また、ジョブに割り当てられ、使用しているリソースを確認することもできます。

### 4.6.1 サーバー、キュー、およびvnodeのリソースの表示

サーバーリソースを確認するには、次の手順を実行します。

```
qstat - Bf
```

キューリソースを確認するには、以下を実行します。

```
qstat -Qf
```

vnodeリソースを確認するには、以下のいずれかを実行します。

```
qmgr -c "list node <vnode名> <属性名>"
pbsnodes -av
pbsnodes [<ホストリスト>]
```

以下の属性を探します。

resources_available.<リソース名>

（サーバー、キュー、vnode）サーバー、キュー、またはvnodeで使用可能なリソースの合計量。使用中のリソースの量は考慮されません。

resources_default.<リソース名>

（サーバー、キュー）ジョブワイドのリソースのデフォルト値。ジョブがこのリソースを要求しない場合、この量が割り当てられます。キュー設定は、サーバー設定より優先されます。

resources_max.<リソース名>

（サーバー、キュー）１つのジョブが要求できる最大量。キュー設定は、サーバー設定より優先されます。

resources_min.<リソース名>

（キュー）１つのジョブが要求できる最小量。

resources_assigned.<リソース名>

（サーバー、キュー、vnode）サーバー、キュー、またはvnodeで実行中および終了中のジョブおよび予約に割り当てられたリソースの合計量。

## 4.6.2 ジョブリソースの表示

ジョブに割り当てられるか、ジョブが使用しているリソースを確認するには、以下を実行します。

```
qstat -f
```

以下のジョブ属性を探します。

Resource_List.<リソース名>

デフォルトも含めてジョブに割り当てられているリソースの量。

resources_used.<リソース名>

ジョブが使用しているリソースの量。

### 4.6.2.1 Resource_List ジョブ属性のリソース

ジョブがジョブワイドのリソースまたは特定の組み込みホストレベルリソースを要求すると、要求した値はジョブのResource_List属性にResource_List.<リソース名>=<値>として格納されます。複数のチャンク内の組み込みホストレベルリソースを要求すると、Resource_Listの値は、すべてのチャンクでのそのリソースの合計になります。Resource_Listで表示可能なリソースのリストについては、『PBS Professional Administrator's Guide』の第5.9.2節「Resources Requested by Job」（241ページ）をご参照ください。

これらのリソースのいずれかに管理者がデフォルト値を定義していて、ジョブがデフォルトを継承した場合、これらのデフォルトが、Resource_List属性に表示される値を制御します。

# 4.7 ジョブ配置の指定

ジョブをvnodeに配置する方法を指定できます。各チャンクを異なるホストに、または異なるvnodeに、配置することを選択できます。または、ジョブが１つのホストにある複数のチャンクを使用することができます。ジョブのすべてのチャンクで、特定のリソースが値を共有する必要があることを指定できます。

ジョブでは、各vnodeの排他的な使用、または他のジョブとの共有使用を要求できます。また、ホストの排他的な使用を要求できます。

ジョブ配置の指定の基本については、この後の各節をご参照ください。MPIジョブ向けのチャンクの配置の詳細については、「マルチプロセッサジョブの投入」をご参照ください。

## 4.7.1 placeステートメントの使用

*place*ステートメントを使用して、ジョブのチャンクの配置方法を指定できます。

placeステートメントには、以下の要素を任意の順序で指定できます。

*-l place=[<arrangement>][: <sharing>][: <grouping>]*

パラメータの意味

*arrangementarrangement*

このチャンクがこのvnodeまたはホストを同じジョブの他のチャンクと共有するかどうかを示します。*free* | *pack* | *scatter* | *vscatter*のいずれかとなります。

*sharing*

このチャンクがこのvnodeまたはホストを他のジョブと共有するかどうかを示します。*excl* | *shared* | *exclhost*のいずれかとなります。

*groupinggrouping*

このジョブのチャンクが、リソースに対して同一の値を持つvnodeに配置される必要があるかどうかを示します。*group=<リソース名>*のインスタンスを１つのみ持つことができます。

修飾子の意味

**表4-3：placement修飾子**

| 修飾子 | 意味 |
|---|---|
| *free* | packやscatterを考慮せず、利用可能なvnodeにジョブを配置します。 |
| *pack* | すべてのチャンクが１台のホストから適用されます。 |
| *scatter* | ホストからは１つのチャンクのみが適用されます。 |
| *vscatter* | vnodeからは１つのチャンクのみが適用されます。<br>各チャンクはvnodeに収まる必要があります。 |
| *excl* | このジョブのみが選択済みvnodeを使用します。 |
| *exclhost* | ホスト全体がこのジョブに割り当てられます。 |
| *shared* | このジョブは選択済みvnodeを共有できます。 |
| *group=<リソース>* | これらのvnodeで共有されているリソースに従って、チャンクがvnodeに配置されます。このリソースは、ストリングまたはストリングアレイである必要があります。グループ内のすべてのvnodeがリソース用の共通の値を含んでいる必要があります。 |

placeステートメントは-l selectなしでは使用されません。

placeステートメントはコロンで始まらない場合があります。

#### 4.7.1.1 チャンクの配置の指定

ジョブのチャンクをどこでも収まるところに配置するには、以下を指定します。

```
-l place=free
```

ジョブのチャンクをすべて1つのホストに配置するには、以下を指定します。

```
-l place=pack
```

ホスト１つにチャンク１つを配置するには、以下を指定します。

```
-l place=scatter
```

vnode１つにチャンク１つを配置するには、以下を指定します。

```
-l place=vscatter
```

##### 4.7.1.1.i 配置の注意事項と制限事項

- *vscatter*を除くすべての配置では、チャンクはホスト間では分割できませんが、同じホスト上のvnode間では分割できます。ジョブがarrangementで*vscatter*を要求しない場合、チャンクは複数のvnodeに分割できます。これは、１つのチャンクが複数のvnodeから取得される可能性があることを意味します。
- ジョブがarrangementで*vscatter*を要求する場合、チャンクはvnodeよりも大きくできません。つまり、vnode間でチャンクを分割できません。この動作は、arrangementの他の値とは異なります。それらの値では、チャンクをvnode間で分割できます。

#### 4.7.1.2 vnodeの使用方法（共有または排他的）の指定

各vnodeを１つのジョブに排他的に割り当てるか、そのリソースをジョブ間で共有することが可能です。また、ホストも１つのジョブに排他的に割り当てるか、ジョブ間で共有できます。

vnodeをジョブに割り当てる方法は、vnodeのsharing属性とジョブのリソース要求の組み合わせによって決定されます。vnodeのsharing属性に設定できる値と、その値がジョブの配置要求とどのように相互作用するかについては、『PBS Professional Reference Guide』の「sharing」（324ページ）をご参照ください。次の表で詳しく説明しています。

**表4-4：vnodeのsharing属性がvnodeの割り当てに与える影響**

| vnodeのsharing属性の値 | 割り当てに与える影響 |
|---|---|
| 未設定 | vnodeをジョブに割り当てる方法は、ジョブのarrangement要求によって決定します。特に指定がない場合は、vnodeは共有されます。 |
| *default_share* | ジョブがvnodeの排他的な使用を明示的に要求しない限り、vnodeは共有されます。 |
| *default_excl* | ジョブが共有割り当てを明示的に要求しない限り、vnodeは排他的にジョブに割り当てられます。 |
| *default_exclhost* | ジョブが共有割り当てを明示的に要求しない限り、このホストからのvnodeはすべて排他的にジョブに割り当てられます。 |

**表4-4：vnodeのsharing属性がvnodeの割り当てに与える影響**

| vnodeのsharing属性の値 | 割り当てに与える影響 |
|---|---|
| *ignore_excl* | ジョブの要求にかかわらず、vnodeは共有されます。 |
| *force_excl* | ジョブの要求にかかわらず、vnodeは排他的にジョブに割り当てられます。 |
| *force_exclhost* | ジョブの要求にかかわらず、このホストのvnodeはすべて排他的にジョブに割り当てられます。 |

vnodeが排他的にジョブに割り当てられる場合、vnodeのリソースすべてがジョブに割り当てられます。vnodeの状態は、*job-exclusive*になります。他のジョブはこのvnodeを使用できません。

ホストが１つのジョブに排他的に割り当てられる場合は、ホスト全体が使用される必要があります。ホストの任意のvnodeで、共有属性が*default_exclhost*または*force_exclhost*のいずれかに設定されている場合、そのホスト上にあるすべてのvnodeでは、共有属性に同じ値が設定されている必要があります。

ジョブで排他的な配置を要求し、それが予約内にある場合、その予約でも`-l place=excl`により排他的な配置を要求する必要があります。

vnodeのsharing属性の値は、以下のどちらかの方法で確認できます。

- `qmgr`の使用：

  **`Qmgr: list node <vnode名> sharing`**

- `pbsnodes`の使用：

  **`pbsnodes -av`**

## 4.7.1.3 リソースのグループ化

ジョブのすべてのチャンクが、選択したリソースに同一の値が含まれるvnodeで動作する必要があることを指定できます。

この方法でジョブのチャンクをグループ化するには、以下の形式を使用します。

*-l place=group=<リソース名>*

ここで、*リソース名*は、文字列または文字列アレイです。

リソースの値にはvnodeごとに１つまたは複数の文字列を使用できますが、すべてのvnodeに共通のストリングが１つ必要です。たとえば、リソースが*router*で、１つのvnodeでは値が“*r1i0,r1*”、別のvnodeでは“*r1i1,r1*”であるとします。この２つのvnodeは文字列“*r1*”を共有しているため、グループ化できます。

リソースでグループ化する方法を使用する場合、リソースにどの値が含まれる必要があるかを指定することはできません。すべてのvnodeで同じ値が含まれるということのみ指定できます。リソースに特定の値が含まれることを指定する必要がある場合は、チャンクの説明でその値を指定します。

### 4.7.1.3.i グループ化とplacement set

管理者がサイトでplacement setを定義する場合があります。placement setは、特定のリソースの値を共有するvnodeグループです。デフォルトで、placement setは“隣接する”vnodeをグループ分けしようとします。placement setが定義されていて、ジョブが特定の配置を必要としない場合、ジョブは自動的にplacement set内で実行されます。『PBS Professional Administrator's Guide』の「Placement Sets」（166ページ）をご参照ください。

リソース別のグループ分けがジョブで要求された場合（*place=group=resource*を使用する場合）、placement setは無視され、要求どおりチャンクが配置されます。

ジョブがグループ分けを要求しても、必要なvnode数が含まれたグループがない場合は、グループ分けが無視されます。

## 4.7.2 ジョブのplaceステートメントの決定方法

管理者がarrangement、sharing、およびgroupingのデフォルト値を定義している場合、１つ以上を明示的に要求している場合を除いて、各ジョブはこのデフォルト値を継承します。つまり、ジョブがarrangementを要求し、sharingやgroupingは要求していない場合、そのジョブは sharing や grouping に関しては値を継承しません。たとえば、管理者が `place=pack:exclhost:group=host`というデフォルト値を設定します。ジョブAが`place=free`を要求し、sharingやgroupingについては指定していない場合、ジョブAはsharingやgroupingの値を継承しません。配置要求を行っていないジョブBは、３つすべてを継承します。

placeステートメントは、以下を指定できます（優先順位の高いものから）。

1. `qalter`における明示的な配置要求
2. `qsub`における明示的な配置要求
3. PBSジョブスクリプト指定における明示的な配置要求
4. デフォルトの`qsub place`ステートメント
5. キューのデフォルト配置ルール
6. サーバーのデフォルト配置ルール
7. 組み込みのデフォルト変換および配置ルール

## 4.7.3 配置の指定の注意事項と制限事項

- place指定はselect指定なしでは使用できません。言い換えれば、配置を指定できるのは、チャンクを指定したときのみです。
- select指定はnodes指定と組み合わせて使用できません。
- select指定は-lncpus、-lmem、-lvmem、-larch、-lhostなどの古いスタイルのリソース要求では使用できません。
- *place=group=<リソース>*を使用する場合、リソースは、文字列または文字列アレイである必要があります。
- 配置を要求するときに、新しい構文と古い構文を混在させないでください。古い構文の説明については、75 ページの第4.8節「下位互換性」をご参照ください。
- ジョブで排他的な配置を要求し、それが予約内にある場合、その予約でも`-l place=excl`により排他的な配置を要求する必要があります。

## 4.7.4 配置の指定例

ジョブに割り当てられたvnodeは、特に指定のない限りvnodeの共有属性に従って共有または排他的に割り当てられます。以下のそれぞれの説明では、-l select= および-l place= の使用方法を示します。

1. 単一ホストには収まるがいずれのvnodeにも収まらないジョブを、可能な限り少ないvnodeに詰め込みます。

   ```
   -l select=1:ncpus=10:mem=20gb
   -l place=pack
   ```

   従来のバージョンでは、これは以下のように表しました。

   ```
   -lncpus=10,mem=20gb
   ```

2. それぞれが１個のCPU、および可能なところから取得される4GBのメモリを持つチャンクを４個要求します。

   ```
   -l select=4:ncpus=1:mem=4GB
   -l place=free
   ```

3. それぞれが１個のCPU、2GBのメモリを持つ４個のチャンクを、archが“linux”の１～４個のvnodeに割り当てます。

```
-l select=4:ncpus=1:mem=2GB:arch=linux -l place=free
```

4. １～４個のvnodeに４個のチャンクを割り当てます。各vnodeは１個のCPU、3GBのメモリを必要とし、チャンクごとに１個のノードロック動的ライセンスを使用します。

```
-l select=4:dyna=1:ncpus=1:mem=3GB -l place=free
```

5. １～４個のvnodeに４個のチャンクおよび４個のフローティング動的ライセンスを割り当てます。これは、“dyna”がサーバーの動的リソースとして指定されていることを前提とします。

```
-l dyna=4 -l select=4:ncpus=1:mem=3GB -l place=free
```

6. archがlinuxであるvnodeを４個だけ選択し、各vnodeが別個のホスト上にあるようにします。各vnodeでは１個のCPUおよび２GBのメモリをジョブに割り当てます。

```
-lselect=4:mem=2GB:ncpus=1:arch=linux -lplace=scatter
```

7. １個のCPU、10GBのメモリをそれぞれが持つチャンクを３個割り当てます。また、100MBのスクラッチスペースが予約されます。スクラッチは、すべてのホストが共有するファイルシステム上にあるのが前提です。“place”の値は、“place=free”というデフォルトの値になります。

```
-l scratch=100mb -l select=3:ncpus=1:mem=10GB
```

8. zolandというホストに２個のCPU、50GBのメモリを割り当てます。“place”の値は、“place=free”というデフォルトの値になります。

```
-l select=1:ncpus=2:mem=50gb:host=zooland
```

9. ２個のホストから１個のCPU、6GBのメモリ、およびホストロックのソフトウェアライセンスを１個割り当てます。

```
-l select=2:ncpus=1:mem=6gb:swlicense=1
-lplace=scatter
```

10. ホスト間に10個のCPUを自由に配置します。

```
-l select=10:ncpus=1
-l place=free
```

11. 単一のHPEシステムには収まるけれども、1個のノードボードに収まりきれない半端なサイズのジョブがあるとします。共有されていないCPUを奇数個要求して、“まとめて”使用できるようにします。

```
-l select=1:ncpus=3:mem=6gb
-l place=pack:excl
```

12. 単一のHPEシステムには収まるけれども、1個のノードボードに収まりきれない半端なサイズのジョブがあるとします。CPUの数は少ないけれども、メモリを大量に要求します。

```
-l select=1:ncpus=1:mem=25gb
-l place=pack:excl
```

13. 空き状況により、複数または単一のHPEシステムジョブを実行します。ジョブは、各HPEシステム上で可能な限り小さなplacement setで実行されます。

```
-l select=2:ncpus=10:mem=12gb
-l place=free
```

14. 複数のHPEシステムを用いてジョブを実行します。ジョブは、各HPEシステム上で可能な限り小さなplacement setで実行されます。

```
-l select=2:ncpus=10:mem=12gb
```

```
-l place=scatter
```

15. 単一ホスト内部でノードボード間に自由に配置します。

```
-l select=1:ncpus=10:mem=10gb
-l place=group=host
```

16. 複数のHPEシステム上のvnode間で自由に配置します。

```
-l select=10:ncpus=1:mem=1gb
-l place=free
```

17. 共有cpusetを使用するスモールジョブです。

```
-l select=1:ncpus=1:mem=512kb
-l place=pack:shared
```

18. グラフィックカードのような、ノードボードの限られたセット上で使用可能な特殊リソースを要求します。

```
-l select= 1:ncpus=2:mem=2gb:graphics=True + 1:ncpus=20:mem=20gb:graphics=False
-l place=pack:excl
```

19. SMPジョブをc-brick内で実行します。

```
-l select=1:ncpus=4:mem=6gb
-l place=pack:group=cbrick
```

20. ジョブがルーターに収まる場合、そのルーターに大きなジョブを配置します。

```
-l select=1:ncpus=100:mem=200gb
-l place=pack:group=router
```

21. １個のルーターに収まりきれない大きなジョブを、できる限り少ない数のルーターに収めます。ここで RES とは、ノードのグループ化に使用されるリソースです。

```
-l select=1:ncpus=300:mem=300gb
-l place=pack:group=<RES>
```

22. MPIジョブを投入するには、MPIタスクごとに１つのチャンクを指定します。MPIタスクごとに2GBのメモリを要求する10ウェイMPIジョブの場合：

```
-l select=10:ncpus=1:mem=2gb
```

23. MPI 以外のジョブ（1CPU のジョブ、OpenMP、共有メモリなど）を投入するには、単一のチャンクを使用します。10GBのメモリが必要な2 CPUのジョブの場合、

```
-l  select=1:ncpus=2:mem=10gb
```

# 4.8 下位互換性

## 4.8.1 古いスタイルのリソース指定

PBSの旧バージョンでは、ジョブ投入者は、“`-lresource=value`”を使用して、-l select外部のリソースを要求できました。現在は、これらのリソースは、-l select内部でチャンク内で要求する必要があります。この古いスタイルのリソース要求は、“リソース指定”と呼ばれていました。リソース指定構文は、**非推奨**です。

下位互換性の目的で、リソース指定はselectおよびplaceステートメントに変換され、デフォルトが適用されます。

## 4.8.2 古いスタイルのノード指定

PBS の初期バージョンでは、ジョブ投入者は、ノード指定で“-l nodes=...”を使用して、ジョブをどこで実行する必要があるかを指定していました。“ノード指定”の構文は、**非推奨**です。

下位互換性の目的で、有効ノード指定またはリソース指定は select 指示文および place 指示文に変換されます。その方法については、この後の各節をご参照ください。

## 4.8.3 古いスタイルから新しいスタイルへの変換

### 4.8.3.1 リソース指定の変換

ジョブのリソース指定のスタイルが古い場合、PBSは、ジョブで指定しているリソースおよびサーバーまたはキュー、あるいはその両方のデフォルトリソースを含むチャンクを１つ要求するselect指定を作成します。リソース指定フォーマット：

*-l<リソース>=<値>[:<リソース>=<値> ...]*

リソース指定は次のように変換されます。

-lselect=1[:<*リソース*>=<値> ...]

-lplace=pack

リソース要求の中で、次の vnode レベルリソースごとに *resource=value* のインスタンスが１つ使用されます。

組み込みリソース：ncpus | mem | vmem | arch | host

サイト定義の vnode レベルリソース

たとえば、ジョブの投入において

**qsub -l ncpus=4:mem=123mb:arch=linux**

を指定した場合、以下の -l select が得られます。

**select=1:ncpus=4:mem=123mb:arch=linux**

### 4.8.3.2 ノード指定の変換

ジョブがノード指定を要求する場合、PBSは、以下の規則に従って、select指定とplace指定を作成します。

古いノード指定フォーマット：

-lnodes=[N:<*指定リスト*> | <*指定リスト*>]

[[+N:<*指定リスト*> | +<*指定リスト*>] ...]

[#<*suffix*> ...][-lncpus=Z]

パラメータの意味：

*指定リスト*の構文：<*spec*>[:<*spec*> ...]

*spec* は hostname | property | ncpus=X | cpp=X | ppn=P のいずれか

*suffix* は property | excl | shared のいずれか

NおよびPは正の整数

XおよびZは負ではない整数

ノード指定は、以下のようにselectおよびplaceステートメントに変換されます。

それぞれの*指定リスト*は1個のチャンクに変換されるため、N:<*指定リスト*>はN個のチャンクに変換されます。

*spec* が hostname の場合：

チャンクには *host=hostname* が含まれる

*spec* が vnode の resources_available.<hostname> 値に一致する場合：

チャンクには *host=hostname* が含まれる

*spec* が property の場合：

チャンクには *<property>=true* が含まれる

property はサイト定義の vnode レベルのブーリアンリソースである必要があります。

*spec* が ncpus=X または cpp=X の場合：

チャンクには ncpus=X が含まれる

*spec* が ncpus=X ではなく cpp=X でない場合：

チャンクには ncpus=P が含まれる

*spec* が ppn=P の場合：

チャンクには mpiprocs=P が含まれる

*ノード指定* が次のような場合

```
-lnodes=N:ppn=P
```

次のように変換されます。

```
-lselect=N:ncpus=P:mpiprocs=P
```

例：

```
-lnodes=4:ppn=2
```

は次のように変換されます。

```
-lselect=4:ncpus=2:mpiprocs=2
```

-lncpus=Z が指定され、spec が ncpus=X を含まず cpp=X も指定がない場合：

すべてのチャンクに ncpus=W が含まれる。ここで W は、チャンクの総数で Z を割った数です （註：W は整数、Z はチャンクの数で均一に割り切れる数でなければなりません）。

property が suffix の場合：

すべてのチャンクに property=true が含まれる

excl が suffix の場合：

place 指示文は -lplace=scatter:excll になる

shared が suffix の場合：

place 指示文は -lplace=scatter:shared になる

excl も shared も suffix でない場合：

place 指示文は -lplace=scatter になる

例：

```
-lnodes=3:green:ncpus=2:ppn=2+2:red
```

次のように変換されます。

```
-l select=3:green=true:ncpus=4:mpiprocs=2+ 2:red=true:ncpus=1
-l place=scatter
```

### 4.8.3.3 古い構文から新しい構文への変換の例

1. 古い構文を使用して、単一ホスト上でCPUおよびメモリを要求します。

   ```
   -l ncpus=5,mem=10gb
   ```

   以下のような同じ意味の構文に変換されます。

   ```
   -l select=1:ncpus=5:mem=10gb
   -l place=pack
   ```

2. 古い構文を使用して、指定されたホスト上でフローティングライセンスを含むカスタムリソースとともに CPU およびメモリを要求します。

   ```
   -l ncpus=1,mem=5mb,host=sunny,opti=1,arch=arch1
   ```

   以下のような同じ意味の構文に変換されます。

   ```
   -l select=1:ncpus=1:mem=5gb:host=sunny:arch=arch1
   -l place=pack
   -l opti=1
   ```

3. 古い構文を使用して、特定のプロパティを持つホストを要求します。

   ```
   -lnodes=1:property
   ```

   以下のような同じ意味の構文に変換されます。

   ```
   -l select=1:ncpus=1:property=True
   -l place=scatter
   ```

4. 古い構文を使用して、特定のプロパティを持つ４個のホストのそれぞれで２個のCPUを要求します。

   ```
   -lnodes=4:property:ncpus=2
   ```

   以下のような同じ意味の構文に変換されます。

   ```
   -l select=4: ncpus=2:property=True -l place=scatter
   ```

5. 古い構文を使用して、特定のソフトウェア、ライセンス、およびジョブ制限メモリ量を必要とする 14 個のホストのそれぞれで１個のCPUを要求します。

   ```
   -lnodes=14:mpi-fluent:ncpus=1 -lfluent=1,fluent-all=1, fluent-par=13
   -l mem=280mb
   ```

   以下のような同じ意味の構文に変換されます。

   ```
   -l select=14:ncpus=1:mem=20mb:mpi_fluent=True
   -l place=scatter
   -l fluent=1,fluent-all=1,fluent-par=13
   ```

6. 古い構文を使用して、ライセンスを要求します。

   ```
   -lnodes=3:dyna-mpi-Linux:ncpus=2 -ldyna=6,mem=100mb, software=dyna
   ```

   以下のような同じ意味の構文に変換されます。

   ```
   -l select=3:ncpus=2:mem=33mb: dyna-mpi-Linux=True
   -l place=scatter
   -l software=dyna
   -l dyna=6
   ```

7. 古い構文を使用して、ライセンスを要求します。

   ```
   -l ncpus=2,app_lic=6,mem=200mb -l software=app
   ```

以下のような同じ意味の構文に変換されます。

```
-l select=1:ncpus=2:mem=200mb
-l place=pack
-l software=app
-l app_lic=6
```

8. 古い構文を使用したその他の例です。

```
-lnodes=1:fserver+15:noserver
```

以下のような同じ意味の構文に変換されます。

```
-l select=1:ncpus=1:fserver=True + 15:ncpus=1:noserver=True
-l place=scatter
```

ただし、以下のようにさらに簡単に指定することもできます。

```
-l select=1:ncpus=1:fserver=True + 15:ncpus=1:fserver=False
-l place=scatter
```

9. ６個のCPUおよび３個のMPIプロセスをそれぞれに持つvnodeを４個割り当てます。各vnodeは別個のホスト上にあるようにします。割り当てられたメモリは、キューまたはサーバーのデフォルトが存在する場合そこで指定されたメモリの４分の１です。これにより、バージョン5.4とは異なるジョブの配置結果となります。

```
-lnodes=4:ppn=3:ncpus=2
```

次のように変換されます。

```
-l select=4:ncpus=6:mpiprocs=3 -l place=scatter
```

10. プロパティblueを指定して４個の異なるホストから４個のvnodeを割り当てます。各vnodeから割り当てられるメモリの量は2560MB（ = 10GB / 4）です。各vnodeから10GBではありません。

```
-lnodes=4:blue:ncpus=2 -l mem=10GB
```

次のように変換されます。

```
-l select=4:blue=True:ncpus=2:mem=2560mb -lplace=scatter
```

## 4.8.4 古い構文を使用する場合の注意事項

### 4.8.4.1 動作の変更点

"`-lnodes`"を使用して投入されるジョブの多くは、通常どおり機能します。これらのジョブは、新しい構文に自動的に変換されます。ただし、ジョブタスクは予期しない順番で実行されます。これは、vnodeが異なる順番で割り当てられるためです。PBS Professional 8.0より前のバージョンでは古い構文で投入しても正常に動作していたジョブが、失敗する可能性があります。これは、チャンクごとの制限がジョブワイドになったためです。

例4-16： `-lnodes=X -lmem=M`を使用して投入されたジョブは、memの制限がジョブワイドになったため、失敗します。次の条件が満たされている場合、

- PBS Professional 9.0以降で標準MPICHを使用
- ジョブが`qsub -lnodes=5 -lmem=10GB`を使用して投入される
- このジョブのマスタープロセスが2GBより多くを使用

ジョブは強制終了されます。バージョン7.0以前では、マスタープロセスは強制終了されるまでに10GB使用できました。この10GBの制限はジョブワイドの制限となり、チャンクごとには分割された2GBの制限が適用されます。

### 4.8.4.2 古いスタイルと新しいスタイルの混在の防止

古いスタイルのリソースまたはノード指定（“-l<リソース>=<値>”または“-lnodes”）をselectおよびplaceステートメント（“-lselect=”または“-lplace=”）と一緒に使用しないでください。コマンドラインで両方を使用しないでください。ジョブスクリプトで両方を使用しないでください。ジョブスクリプトに新旧どちらかのリソース要求を１つ使用し、コマンドラインに他の１つを使用するようなことはしないでください。エラーの原因となります。

### 4.8.4.3 リソースの定義場所に依存するリソース要求の変換

ジョブのリソース要求は、さまざまな規則に従って古いスタイルから新しいスタイルに変換されます。このような規則の1つに、リソースの定義場所に依存した変換があります。以下に例を示します。ブーリアンリソース“Red”はサーバー上に定義されており、ブーリアンリソース“Blue”はホストレベルで定義されています。ジョブで“qsub -l Blue=true”が要求されます。これは古いスタイルのリソース要求の形式であるため、PBSではBlueがどこに定義されているのかを確認します。Blueはホストレベルで定義されているため、要求は“-l select=1:Blue=true”に変換されます。ただし、ジョブで“qsub -l Red=true”が要求されると、古いスタイルのリソース要求の形式であっても、Redはサーバーに定義されているためPBSではチャンク要求に変換しません。

### 4.8.4.4 プロパティの非推奨

プロパティ要求の構文は、**非推奨**です。管理者はプロパティをブーリアンで置き換えています。

### 4.8.4.5 cppのncpusによる置き換え

“cpp”の指定は古い構文に含まれるため、これを“ncpus”で置き換える必要があります。

### 4.8.4.6 変換時に設定される環境変数

ノード指定を-l selectに変換する際、ジョブでは環境変数のNCPUSおよびOMP_NUM_THREADSが古いノード指定の先頭部分のncpusの古い値に設定されます。その結果、それぞれの部分で異なるncpus値およびppn値を使用した複雑なノード指定が変換されると、以前のバージョンとの互換性が失われる場合があります。

### 4.8.4.7 cgroupに適合していない旧-l nodes構文

cgroup hookは、旧来の”-lnodes”構文を新しいselectおよびplace指示文に変換しません。cgroup hookによって管理されているホスト上で旧構文を使用する必要がある場合は、サイトでqueuejob hookを使用してそれを行うか、ジョブ用のmem、vmem、cgswapに明示的に指定します。

# 5 マルチプロセッサジョブ

## 5.1 マルチプロセッサジョブの投入

この章を読む前に、53ページの第4章「ジョブへのリソースの 割り当てとジョブの配置」をお読みください。

### 5.1.1 必要なチャンクの割り当て

PBSはチャンクがselectステートメントに出現する順番でチャンクをジョブプロセスに割り当てます。PBSはプライマリ実行ホストから最初のチャンクを取得します。ここで、ジョブの最優先タスクが実行されます。

例5-1： ３つのチャンクが必要です。最初のチャンクは２つのCPUと20GBのメモリ、２つ目は４つのCPUと100GBのメモリ、３つ目は１つのCPUと5GBのメモリをそれぞれ備えています。

```
-lselect=1:ncpus=2:mem=20gb+ncpus=4:mem=100gb+mem=5gb
```

#### 5.1.1.1 プライマリ実行ホストの指定

ジョブのプライマリ実行ホストとは、ジョブによって要求された最初のチャンクを満たすvnodeを提供するホストのことです。

#### 5.1.1.2 最も固有性の高いチャンクを最優先する

チャンク要求は左から右に解釈されます。具体的なチャンクほど早い順番にする必要があります。たとえば、チャンクAに固有のホストが必要で、チャンクBはホスト固有ではない場合、チャンクAを先に要求します。

### 5.1.2 ジョブのノードファイル

各ジョブに対して、PBSはジョブ固有の“ホストファイル”（または“ノードファイル”）を作成します。このファイルは、ジョブに割り当てられているvnodeを含んだホストの数が記載されているテキストファイルです。ジョブに割り当てられたすべての実行ホストに対して、このファイルが設定されます。

#### 5.1.2.1 ノードファイルの形式および内容

ノードファイルには、ホスト名が１つずつ１行ごとにリストされます。ホストの名前は、割り当てられたvnodeのresources_available.host内の値です。PBSノードファイル内でホストが表示されている順序は、チャンクが-l selectで指定されている順序です。

ノードファイルの各行には、個々のMPIプロセスが実行されるホスト名が記されます。ジョブのMPIプロセスの数とノードファイルの内容は、リソースmpiprocsの値によって制御されます。mpiprocsは、チャンクごとのMPIプロセスの数であり、チャンクにCPUが含まれる場合はデフォルトで*1*、含まれない場合はデフォルトで*0*になります。

mpiprocs=*M*を要求するチャンクごとに、そのチャンクが割り当てられているホストの名前が、ノードファイルに*M*回書き込まれます。したがって、ノードファイルの行数は、ジョブによって要求されたすべてのチャンクに要求されたmpiprocsの合計となります。

例5-2：HostAでは2つのMPIプロセスが実行され、HostBでは１つのMPIプロセスが実行されています。その場合のノードファイルは次のようになります。

```
HostA
HostA
HostB
```

### 5.1.2.2 ノードファイルの名前および場所

このファイルは、ジョブに割り当てられた実行ホストごとに`PBS_HOME/aux/JOB_ID`に作成されます。*JOB_ID*は、そのジョブのジョブ識別子です。

ノードファイルのフルパスと名前は、環境変数PBS_NODEFILEの値としてそのジョブの環境で設定されます。

### 5.1.2.3 古いスタイルの要求のためのノードファイル

古い*-lnodes=nodespec*形式を使用してリソースを要求するジョブについては、ジョブに割り当てられた各vnodeのホストが*N*回リスト表示されます。ここで、*N*はvnode上のMPIランクの数です。MPIランクの数は、ppnリソースによって指定されます。

例5-3：それぞれ２つのMPIプロセスを持つ４つのvnodeを要求します。ここで、それぞれのプロセスには３つのスレッドがあり、各スレッドにCPUがあります。

```
qsub -lnodes=4:ncpus=3:ppn=2
```

これによって、vnodeがジョブに割り当てられた順序で４つのホストそれぞれが２回書き込まれます。

### 5.1.2.4 ノードファイルの使用と変更

ジョブスクリプトで$PBS_NODEFILEを使用できます。

ノードファイルは変更可能です。エントリを削除したり、ソートできます。

### 5.1.2.5 ノードファイルの注意事項

新しいホストにエントリを追加しないでください。PBSはこれらのホスト上でプロセスが実行されると予想していないため、これらのホスト上でのプロセスを停止する場合があります。要求された数よりも多くのCPUを使用しているため、同じホストにエントリを追加するとジョブが停止する可能性があります。

### 5.1.2.6 実行ホストの表示

どのホストがプライマリ実行ホストであるかは、次の方法で確認できます。プライマリ実行ホストは、ジョブのノードファイルにリスト表示されている最初のホストです。

## 5.1.3 チャンクごとのMPIプロセス数の指定

チャンクの要求方法は重要です。まず、mpiprocsリソースを使用して指定しない限り、デフォルトでのチャンクごとのMPIプロセス数は、CPUのあるチャンクの場合は*1*、CPUのないチャンクの場合は*0*です。次に、MPIプロセスがCPUを共有するかどうかを指定できます。たとえば、４つのCPUと４つのMPIプロセスを備えた１つのチャンクの要求は、それぞれ１つのCPUと１つのMPIプロセスを備えた４つのチャンクの要求と同じではありません。前者の場合、４つのMPIプロセスすべてが４つのCPUをすべて共有しています。後者の場合、各プロセスが独自のCPUを持ちます。

mpiprocsリソースを使用して、各チャンクに必要な数のMPIプロセスを要求します。たとえば、各チャンクが２つのCPUを持つ４つのチャンクそれぞれに２つのMPIプロセスを要求する場合、

```
-lselect=4:ncpus=2:mpiprocs=2
```

mpiprocsリソースの値を明示的に要求しないと、デフォルトでCPUを要求する各チャンクの場合は*1*、CPUを要求しないチャンクの場合は*0*となります。

例5-4： どちらも２つのCPUを持つ、２つのMPIプロセスを備えたチャンクを１つと１つのMPIプロセスを備えたチャンクを１つ要求するには次のコマンドを使用します。

```
-lselect=ncpus=2:mpiprocs=2+ncpus=2
```

例5-5： それぞれ１つのMPIプロセスを備えた３つのvnodeの要求の場合、

```
qsub -l select=3:ncpus=2
```

これにより、次のノードファイルが生成されます。

<１つ目のvnodeのホスト名>

<２つ目のvnodeのホスト名>

<３つ目のvnodeのホスト名>

例5-6： ３つのホストそれぞれで２つのMPIプロセスを実行し、各ホスト上の１個のプロセッサをMPIプロセス間で共有する場合は、次のように要求します。

```
-lselect=3:ncpus=1:mpiprocs=2
```

ノードファイルには、次のリストが含まれます。

VnodeAのホスト名

VnodeAのホスト名

VnodeBのホスト名

VnodeBのホスト名

VnodeCのホスト名

VnodeCのホスト名

例5-7： それぞれが２個のCPUを持ち２つのMPIプロセスを実行する３つのチャンクを要求する場合は、次のコマンドを使用します。

```
-l select=3:ncpus=2:mpiprocs=2...
```

ノードファイルには、次のリストが含まれます。

VnodeAのホスト名

VnodeAのホスト名

VnodeBのホスト名

VnodeBのホスト名

VnodeCのホスト名

VnodeCのホスト名

使用するCPUの数が異なっても、ノードファイルは前の例と同じになることにご注意ください。

例5-8： 各プロセスに独自のCPUを持つ４つのMPIプロセスを要求する場合

```
-lselect=4:ncpus=1
```

mpiprocsリソースの定義については、『PBS Professional Reference Guide』の「Resources Built Into PBS」（265ページ）をご参照ください。

### 5.1.3.1 MPIプロセスのないチャンク

MPIプロセスのないチャンクを要求すると、PBSはすでに別のチャンクを提供したvnodeからそのチャンクを取得する場合があります。次のいずれかを使用して、MPIプロセスのないチャンクを要求します。

```
-lselect=1:ncpus=0
-lselect=1:ncpus=2:mpiprocs=0
```

## 5.1.4 マルチプロセッサジョブに関する注意事項とアドバイス

### 5.1.4.1 ユニフォームプロセッサの要求

いくつかのMPIジョブは、次のステージに移行する前にすべてのvnode上での作業が同じステージになることを必要とします。これらのアプリケーションの場合、速いvnodeは遅いvnodeが追い着くまで待たなくてはならないため、作業は最も遅いvnodeのペースでしか進めません。この場合、ジョブのvnodeが同種であると便利です。

vnodeのアーキテクチャ、タイプ、速度を識別するリソースがある場合、それを使用してすべてのチャンクが同じ値のvnodeから取得されるようにできます。すべてのチャンクに対してこのリソースに特定の値を要求することも、リソースの値に従ってvnodeをグループ分けすることもできます。72ページの第4.7.1.3節「リソースのグループ化」をご参照ください。

例5-9： 速度を識別するリソースは*speed*という名前が付けられ、ジョブはそれぞれ２つのCPU、２つのMPIプロセスを持ち、すべてspeedが*fast*の16のチャンクを要求します。

```
-lselect=16:ncpus=2:mpiprocs=2:speed=fast
```

例5-10： グループ分けを使用して、すべてのチャンクが同じ速度を共有するようにして、各チャンクが２つのCPUを持つ16のチャンクを要求します。速度を特定するリソースの名前は*speed*です。

```
-lselect=16:ncpus=2:mpiprocs=2:place=group=speed
```

### 5.1.4.2 NFSサーバー上でのストレージ要求

コンプレックス内のvnodeの１つが残りのvnodeに対してNFSサーバーとして機能する場合があります。このため、vnodeはすべてNFSサーバー上のストレージに対するアクセスを持ちます。

例5-11： scratchリソースはコンプレックス内のすべてのvnodeで共有され、“*nfs_server*” vnodeと呼ばれる中央の場所から要求されます。計算を行うために２つのCPUをそれぞれ備えた２つのvnodeと10GBのメモリを備えMPIプロセスのないvnodeを１つを要求するには次のコマンドを使用します。

```
-l select=2:ncpus=2+1:host=nfs_server:scratch=10gb:ncpus=0
```

この要求では、ジョブはCPUを含む各チャンクにMPIプロセスを持ち、メモリのみのチャンクにはMPIプロセスを持ちません。ジョブは“nfs_server”ホストにチャンクを持っていると表示されます。

## 5.1.5 マルチプロセッサジョブのためのファイルステージング

PBSでは、プライマリ実行ホストのみでファイルがステージインおよびステージアウトされます。

## 5.1.6 prologueとepilogue

prologueはプライマリホスト上でrootとして実行されます。この際、PBS_HOME/mom_privをカレントディレクトリとし、PBS_JOBDIRおよびTMPDIRは環境変数として設定されます。

PBSはプライマリホスト上でepilogueをrootとして実行します。epilogueはジョブのステージングおよび実行ディレクトリを現在の作業ディレクトリとして実行され、PBS_JOBDIRおよびTMPDIRが環境変数として設定されます。

## 5.1.7 MPI環境変数

NCPUS

PBSはプライマリ実行ホスト上のジョブの環境でNCPUS環境変数を設定します。PBSは、最初のチャンクに対して要求されたncpusの値をNCPUSに設定します。

OMP_NUM_THREADS

PBSはプライマリ実行ホスト上のジョブの環境でOMP_NUM_THREADS環境変数を設定します。PBSはこの変数を最初のチャンクに対して要求されたompthreadの値に設定します。デフォルトでは最初のチャンクに対して要求されたncpusの値になります。

## 5.1.8 マルチプロセッサジョブの例

例5-12： MPIタスクごとに2GBのメモリを要求する10ウェイMPIジョブの場合：

```
qsub -l select=10:ncpus=1:mem=2gb
```

例5-13： それぞれが２個のCPUを備えているような小規模システムのクラスタ環境で、４台の異なるホスト上で実行するMPIジョブを投入するには、以下のように実行します。

```
qsub -l select=4:ncpus=1 -l place=scatter
```

この例で、ノードファイルには、ジョブに割り当てられたホストごとに１つのエントリ、つまり４つのエントリが作成されます。

NCPUS変数およびOMP_NUM_THREADS変数は、1に設定されます。

例5-14： ４つのMPIプロセスがどこで実行されてもよい場合は、以下のように投入します。

```
qsub -l select=4:ncpus=1 -l place=free
```

ここで、ジョブは、使用できる台数に応じて、２台、３台、または４台のホストで実行されます。

この例では、ノードファイルには４つのエントリが含まれます。４台の異なるホスト、３台のホスト（その内１台は２回記載）、または２台のホストで構成されます。

NCPUSおよびOMP_NUM_THREADSは、最初のチャンクから割り当てられたCPUの数である*1*に設定されます。

### 5.1.9 SMPジョブの投入

SMPジョブを投入するには、必要なCPUとメモリのすべてを含むチャンクを１つ要求し、必要に応じてホスト名を指定するだけです。以下に例を示します。

```
qsub -l select=ncpus=8:mem=20gb:host=host1
```

ジョブの実行時に、ノードファイルに１つのエントリ（選択した実行ホストの名前）が記載されます。

ジョブには、NCPUSおよびOMP_NUM_THREADSの２つの環境変数があり、CPU割り当て数に設定されます。

## 5.2 PBSでのMPIの使用

### 5.2.1 統合されたMPIの使用

PBSに統合されたMPIは多数あります。PBSは、それらのほとんどを統合するツールを提供します。いくつかのMPIは統合が可能です。ジョブが統合されたMPIで実行されると、PBSはジョブのすべてのプロセスに対してリソース使用量を追跡し、ジョブプロセスにシグナルを送信し、アカウンティングを実行できます。

ジョブがPBSに統合されていないMPIで実行されると、PBSはプライマリvnode上のジョブの管理のみに制限されるため、リソース追跡、ジョブへのシグナルの送信、アカウンティングはプライマリvnode上のプロセスにのみ行われます。

次の指示は統合されたMPI向けです。ご自身のサイトでどのMPIが統合されているかについては、管理者にご確認ください。MPIがPBSと統合されていない場合、PBS外部の場合と同じように使用します。

一部の統合されたMPIではコマンドラインが若干異なります。各MPIの手順をご参照ください。

次の表には、サポートされているMPIと各MPIを使用するための手順へのリンクを示します。

**表5-1：サポートされているMPIのリスト**

| MPI名 | バージョン | 使用方法 |
|---|---|---|
| HP MPI | 1.08.03<br>2.0.0 | 89ページの第5.2.4節「PBSでのHP MPI」をご参照ください。 |
| Intel MPI | 2.0.022<br>3<br>4 | 90ページの第5.2.7節「Intel MPI 2.0.022、3、および4とPBS」をご参照ください。 |
| Intel MPI | 4.0.3（Linuxの場合） | 89ページの第5.2.5節「Linux上のIntel MPI 4.0.3とPBS」をご参照ください。 |
| Intel MPI | 4.0.3（Windowsの場合） | 89ページの第5.2.6節「Windows上のIntel MPI 4.0.3とPBS」をご参照ください。 |
| MPICH-P4<br>**非推奨**。 | 1.2.5<br>1.2.6<br>1.2.7 | 93ページの第5.2.8節「PBSでのMPICH-P4」をご参照ください。 |
| MPICH-GM<br>**非推奨**。 | | 94ページの第5.2.9節「PBSでのMPICH-GM」をご参照ください。 |
| MPICH-MX<br>**非推奨**。 | | 97ページの第5.2.10節「PBSでのMPICH-MX」をご参照ください。 |

表5-1：サポートされているMPIのリスト

| MPI名 | バージョン | 使用方法 |
|---|---|---|
| MPICH2<br>**非推奨**。 | 1.0.3<br>1.0.5<br>1.0.7<br>（Linuxの場合） | 100ページの第5.2.11節「Linux上のMPICH2とPBS」をご参照ください。 |
| MPICH2 | 1.4.1p1（Windowsの場合） | 102ページの第5.2.12節「Windows上のMPICH2 1.4.1p1とPBS」をご参照ください。 |
| MVAPICH<br>**非推奨**。 | 1.2 | 102ページの第5.2.13節「PBSでのMVAPICH」をご参照ください。 |
| MVAPICH2 | 1.8 | 104ページの第5.2.14節「PBSでのMVAPICH2」をご参照ください。 |
| Open MPI | 1.4.x | 106ページの第5.2.15節「PBSでのOpen MPI」をご参照ください。 |
| Platform MPI | 8.0 | 106ページの第5.2.16節「PBSでのPlatform MPI」をご参照ください。 |
| HPE MPI | 任意 | 106ページの第5.2.17節「PBSでのHPE MPI」をご参照ください。 |

### 5.2.1.1 統合の注意事項

- 一部のMPIのコマンドラインは若干異なります。それぞれの相違点については説明してあります。

### 5.2.1.2 オンザフライによるMPIの統合

PBS管理者はこの手順を実行して、サポートされているMPIを統合できます。統合されていないMPIについては、オンザフライで統合できます。Intel MPI 4.0.3は、環境変数を使用して統合します。89ページの第5.2.5節「Linux上のIntel MPI 4.0.3とPBS」をご参照ください。それ以外については、`pbs_tmrsh`コマンドを使用して統合します。

#### 5.2.1.2.i `pbs_tmrsh`コマンドを使用したオンザフライによるMPIの統合

統合されたMPIまたはIntel MPI 4.0.3には`pbs_tmrsh`を使用しないでください。

このコマンドは`rsh`をエミュレートしますが、PBS TMインターフェースを使用して、シスターvnodes上の`pbs_mom`と直接通信します。`pbs_tmrsh`コマンドは、シスターvnodes上のジョブプロセスについてプライマリMoMとシスターMoMに通知します。ジョブが`pbs_tmrsh`を使用すると、PBSはすべてのジョブプロセスのリソース使用量を追跡できます。

`rsh`または`ssh`コマンドとして、`pbs_tmrsh`を使用します。`pbs_tmrsh`を使用するには、適切な環境変数を*pbs_tmrsh*に設定します。たとえば、MPICHを統合するにはP4_RSHCOMMAND環境変数を*pbs_tmrsh*に設定し、HP MPIを統合するにはMPI_REMSHを*pbs_tmrsh*に設定します。

次の図は、オンザフライでMPIを統合するためのpbs_tmrshコマンドの使用方法を示しています。

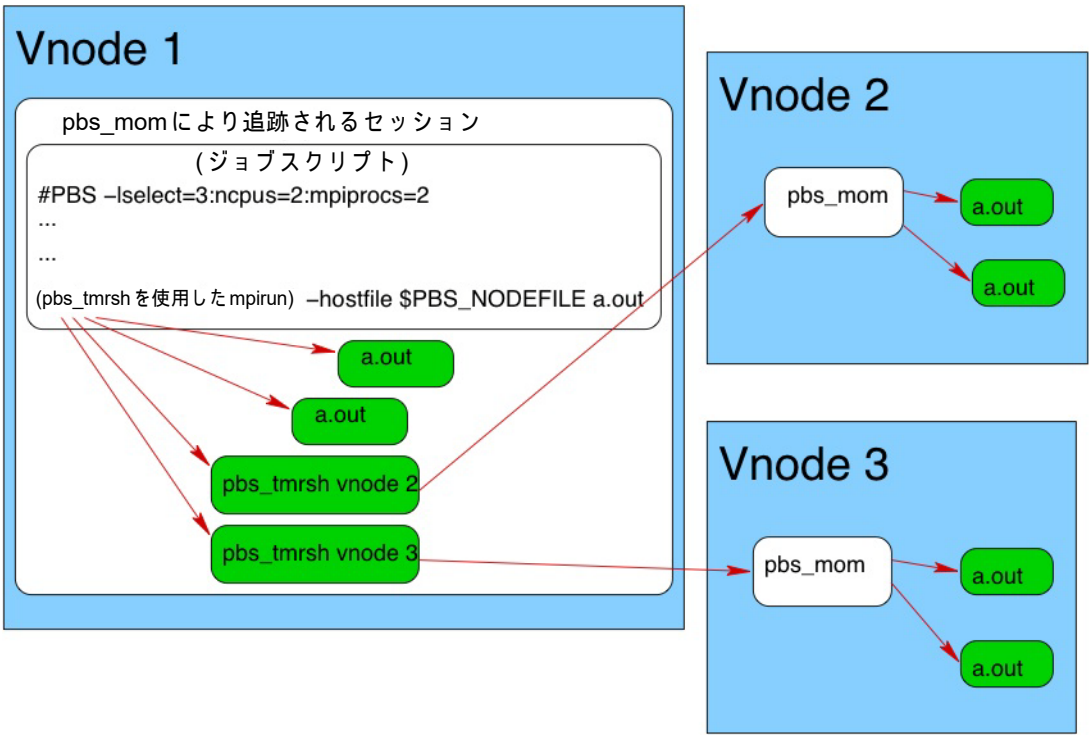


図5-1：pbs_tmrshは直接pbs_momと通信するため、PBSはvnode2およびvnode3上のプロセスを認識し、pbs_momはvnode2およびvnode3上でプロセスを開始します。

#### 5.2.1.2.ii pbs_tmrshコマンドの注意事項

- このコマンドはPBSジョブの外部で使用できません。PBSジョブの外部で使用すると、このコマンドは失敗します。
- pbs_tmrshコマンドはrshとまったく同じには機能しません。たとえば、pbs_tmrshからの出力をパイピングできません。これは失敗します。

## 5.2.2 PBSでのMPI使用の前提条件

PBSと使用するMPIは、PBSと使用する前に機能している必要があります。PBS外部でMPIジョブを実行できる必要があります。

## 5.2.3 MPI使用の注意事項

一部のアプリケーションは、一時的な場所にスクラッチファイルを書き込みます。PBSはこのために使用できる一時ディレクトリを作成し、そのパスをTMPDIR環境変数に設定します。一時ディレクトリの場所はホストによって異なります。Open MPI以外のMPIを使用しており、アプリケーションにスクラッチスペースが必要な場合、ジョブの一時ディレクトリは実行ホスト全体で共通している必要があります。PBS管理者は、$tmpdir MoMパラメータを使用して、各ホスト上の一時ディレクトリのルートを指定できます。この場合、TMPDIR環境変数は、生成される一時ディレクトリのフルパスに設定されます。TMPDIRを設定しようとしないでください。

## 5.2.4 PBSでのHP MPI

HP MPIはLinux上でPBSと統合できるため、PBSはすべてのジョブプロセスのリソース使用量を追跡し、プロセスにシグナルを送信し、アカウンティングを実行できます。PBS管理者はHP MPIとPBSを統合できます。

### 5.2.4.1 HP MPIのための環境の設定

デフォルトの`rsh`を指定変更するには、ジョブスクリプト内でPBS_RSHCOMMANDを設定します。

```
export PBS_RSHCOMMAND=<選択したrsh>
```

### 5.2.4.2 PBSでのHP MPIの使用

MPIコマンドラインに変更を加えずに、HP MPIを使用してPBSでジョブを実行できます。

### 5.2.4.3 オプション

PBS HP MPIジョブを実行する場合、PBS外部の`mpirun`コマンドにも同じ引数を使用できます。次のオプションはPBS下で異なる扱いをされます。

-h <ホスト>
: 無視

-l <ユーザー>
: 無視

-np <数>
: 使用できるリソースに合わせて修正されます

### 5.2.4.4 PBSでのHP MPIの注意事項

統合されたHP MPIでは、ジョブの作業ディレクトリがユーザーのホームディレクトリに変更されます。

## 5.2.5 Linux上のIntel MPI 4.0.3とPBS

PBS管理者が、Linux上のIntel MPI 4.0.3をPBSと統合している場合、その`mpirun`は、PBSジョブの内部と外部でまったく同じ方法で使用できます。

Linux上のIntel MPI 4.0.3に対するデフォルトのプロセスマネージャは、Hydraです。

## 5.2.6 Windows上のIntel MPI 4.0.3とPBS

Windowsでは、PBSには、`pbs_intelmpi_mpirun.bat`と呼ばれるIntel MPI用のラッパースクリプトが`$PBS_EXEC¥bin`に用意されています。Intel `mpirun`の代わりにこのスクリプトを呼び出します。オプションはすべて、このスクリプトから`mpirun`に渡されます。

#### 5.2.6.1 オンザフライによるIntel MPI 4.0.3の統合

Intel MPI 4.0.3をPBSと統合せずに使用している場合、以下の手順を実行して、オンザフライでPBSを統合できます。

1. rshを指定します。

```
I_MPI_HYDRA_BOOTSTRAP=rsh
```

2. pbs_tmrshを指定します。

   a. デフォルトのPATHにPBS_EXEC/binが含まれているホストでジョブ全体を実行している場合は、以下を設定します。

```
I_MPI_HYDRA_BOOTSTRAP_EXEC=pbs_tmrsh
```

   b. デフォルトのPATHにPBS_EXEC/binが含まれていないホストでジョブ全体を実行している場合は、フルパスを環境変数に追加します。以下に例を示します。

```
I_MPI_HYDRA_BOOTSTRAP_EXEC=/opt/pbs/bin/pbs_tmrsh
```

### 5.2.7 Intel MPI 2.0.022、3、および4とPBS

PBSは、これらのバージョンのIntel MPIのmpirunへのインターフェースを提供します。PBSジョブの内部で実行されると、すべてのIntel MPIのプロセスがPBSによって追跡されるため、PBSはアカウンティングを実行してジョブを完全に制御できます。PBSジョブの外部で実行される場合、標準のIntel MPIのmpirunが使用された場合と同様に動作します。

#### 5.2.7.1 PBSに統合されたIntel MPI 2.0.022、3、または4の使用

PBSの外部で使用するmpirunコマンドと同様のものを使用します。

Intel MPIのmpirunにPBS提供インターフェースを呼び出すPBSジョブを投入する際、ランクまたはMPIタスクの実際の数をqsubのselect指定の中で明示的に指定するようにしてください。そうしないと、“マシンファイル内のエントリが少なすぎます”というメッセージが出力されジョブが失敗します。

この問題の例として、次のような指定があります。

```
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

これにより、次のノードファイルが生成されます。

```
hostA
hostB
```

これでは2個のMPDデーモンしか起動されないため、mpirunの中の“-np 3”指定と競合します。

正しい方法は、次のいずれかを指定することです。

```
#PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2
```

これによって、ノードファイルに次が含まれます。

```
hostA
hostB
hostB
```

さらに“mpirun -np 3”と一致しています。

### 5.2.7.2 統合されたIntel MPI 2.0.022、3、または4のオプション

PBSジョブスクリプト内で実行された場合、PBSインターフェースへのすべてのオプションは次のものを除いてIntel MPIのmpirunと同様です。

-host, -ghost

実行されるホストを指定するのに使用します。無視されます。

-machinefile <ファイル>

file引数の内容は無視され$PBS_NODEFILEの内容と置き換えられます。

mpdboot option --totalnum=*

無視され、$PBS_NODEFILE内の一意のエントリ数によって置き換えられます。

mpdboot option --file=*

無視され、$PBS_NODEFILE内の一意の名前によって置き換えられます。このオプションの引数は$PBS_NODEFILEによって置き換えられます。

mpdbootオプション-f <mpd_hosts_file>の引数は$PBS_NODEFILEによって置き換えられます。

-s

Intel MPIのmpirunへのPBSインターフェースがPBSジョブ内で呼び出されると、mpirunのオプション"-s <spec>"が厳密に比較されるため、Intel MPIのmpdbootの引数mpirun -sはサポートされません。mpirunを呼び出す前に別のmpdboot -sを実行できます。-sオプションが見つかった場合、サポートされる形式をユーザーに伝える警告メッセージがPBSインターフェースによって出力されます。

-np

ユーザーが-npオプションを指定しない場合、PBSインターフェースはデフォルト値を提供しません。標準のmpirunによって適切なデフォルト値(通常は*1*)が決定されます。起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

### 5.2.7.3 MPDの起動と停止

Intel MPIのmpirunによりMPDデーモンの起動および停止が行われます。Intel MPIのmpirunのPBSインターフェースは、$PBS_NODEFILE内の一意のエントリからの入力を使用して、引数-totalnum=<開始するMPDの数>および-file=<mpd_hosts_file>を実際のmpirunに常に渡します。

### 5.2.7.4 例

例5-15: $PBS_NODEFILEにリストされているPBS割当ホスト間に展開されている6個のプロセスを使用して実行可能なIntel MPIシングルジョブを実行します。

ノードファイル:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

ジョブスクリプト：

```
# mpirunにより$PBS_NODEFILEにリスト
# されている一意のホスト上でMPDデーモンの起動が
# 行われ、$PBS_NODEFILEにリストされている
# ６つのホスト上で、６つのプロセスも
# 実行され、mpirun により
# MPDの停止が行われます
mpirun /path/myprog.x 1200
```

ジョブスクリプトの実行：

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<ジョブID>
```

例5-16： mpirunの中で$PBS_NODEFILEおよびmpiexec引数を使用して、複数のホスト上で複数の実行可能ファイルとIntel MPIジョブを実行します。

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

ジョブスクリプト：

```
# mpirunが$PBS_NODEFILEにリストされているホスト上でMPDデーモンを実行します
# mpirunがhostA上でmpitest1の２つのインスタンス、
# hostB上でmpitest2の２つのインスタンス、
# hostB; 2 instances of mpitest3 on hostC.
# MPIジョブの実行の終了時にmpirunにより
# MPDの停止が行われます
mpirun -np 2 /tmp/mpitest1 : -np 2 /tmp/mpitest2 : -np 2 /tmp/mpitest3
```

ジョブスクリプトの実行：

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
 <ジョブID>
```

例5-17： -configfileオプションおよび$PBS_NODEFILEを使用して、複数のホスト上で複数の実行可能ファイルとIntel MPIジョブを実行します。

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

ジョブスクリプト：

```
echo "-np 2 /tmp/mpitest1" >> my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file

# mpirunによりMPD デーモンの起動が行われます
# configファイルがhostA上でmpitest1の２つのインスタンス、
# hostB上でmpitest2の２つのインスタンス、
# hostB; 2 instances of mpitest3 on hostC.
# mpirunによりMPDデーモンの停止が行われます
mpirun -configfile my_config_file

# クリーンアップ
rm -f my_config_file
```

ジョブスクリプトの実行：

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<ジョブID>
```

#### 5.2.7.5 制約

統合されたIntel MPIで起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

### 5.2.8 PBSでのMPICH-P4

ラッパーは**非推奨**です。MPICH-P4はLinux上でPBSと統合できるため、PBSはすべてのジョブプロセスのリソース使用量を追跡し、プロセスにシグナルを送信し、アカウンティングを実行できます。PBS管理者はMPICH-P4とPBSを統合できます。

#### 5.2.8.1 PBSでのMPICH-P4のオプション

PBSで、Linux上のMPICH-P4 `mpirun`コマンドの構文と引数は次の１つのオプション以外は同じで、これを設定するべきではありません。

-machinefile <ファイル>

PBSにはマシンファイルが用意されています。これを指定しようとすると、PBSは、マシンファイルが置き換えられようとしているという警告を表示します。

### 5.2.8.2 PBSでのMPICH-P4の使用例

mpirunの使用例

```
#PBS -l select=arch=linux
#
mpirun a.out
```

### 5.2.8.3 WindowsでのMPICH

Windows環境では、ジョブのプロセスをより効率的に実行できるようにするため、また“failed to communicate with the barrier command”エラーを避けるために、MPICHのmpirunコマンドの-localrootオプションが必要となる場合があります。ジョブスクリプトの例を以下に示します。

```
C:¥DOCUME~1¥user1>type job.scr
echo begin
type %PBS_NODEFILE%
"¥Program Files¥MPICH¥mpd¥bin¥mpirun" -localroot -np 2 -machinefile %PBS_NODEFILE%
    ¥winnt¥temp¥netpipe -reps 3
echo done
```

チャンクの指定ごとに“arch=windows”を指定することも必要です。

#### 5.2.8.3.i WindowsでのMPICHの注意事項

WindowsでMPICHはPBSに統合されていません。このため、PBSはプライマリvnode上のジョブプロセスのプロセスの追跡と制御、アカウンティングの実行のみに限られています。

## 5.2.9 PBSでのMPICH-GM

### 5.2.9.1 PBSでのMPICH-GMとMPDの使用

ラッパーは**非推奨**です。PBSはMPDを使用してMPICH-GMのmpirunへのインターフェースを提供します。PBSジョブの内部で実行されると、MPDデーモン経由で開始されたすべてのMPICH-GMプロセスがPBSによって追跡されるため、PBSはアカウンティングを実行してジョブを完全に制御できます。PBSジョブの外部で実行される場合、標準のmpirunがMPDで使用された場合と同様に動作します。

PBSの外部で使用するmpirunコマンドと同様のものを使用します。MPDデーモンがまだ実行されていない場合、PBSインターフェースがデーモンを開始します。

#### 5.2.9.1.i オプション

PBSジョブスクリプト内では、PBSインターフェースへのすべてのオプションは次のものを除いてmpirunと同様です。

-m <ファイル>

fileの引数の内容は無視され$PBS_NODEFILEの内容と置き換えられます。

-np

指定しない場合、$PBS_NODEFILEの中のエントリ数が使用されます。起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

-pg

複数のホスト上の複数の実行可能ファイルのために-pgオプションを使用することはできますが、プロセスグループファイルでPBSホストのみが指定されていることを確認する必要があります。これは、非PBSホスト上で生成されたMPIプロセスはPBSによって制御される保証がないからです。

### 5.2.9.1.ii MPDの起動と停止

スクリプトは$PBS_NODEFILEにリストされている一意の各ホスト上で、環境変数RSHCOMMANDの値に基づいてrshまたはsshメソッドを使用してMPDデーモンを開始します。デフォルトはrshです。スクリプトは実行が終了するとMPDデーモンを停止します。

MPDデーモンが起動していない場合、mpirunへのPBSインターフェースにより、割り当てられたPBSホスト上でGMのMPDデーモンがユーザーとして開始されます。MPDデーモンは管理者またはユーザーによってすでに開始されている可能性があります。MPDバイナリへのパスを決定する、ユーザーが実行したmpirunのパス（GMまたはMX）をMPDデーモンは持たないため、このデーモンはPBS prologueスクリプト内では開始されません。

### 5.2.9.1.iii 例

例5-18： $PBS_NODEFILEにリストされているPBS割当ホスト間に展開されている3個のプロセスを使用して実行可能なMPICH-GMシングルジョブを実行します。

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

```
qsub -l select=3:ncpus=1
[MPICH-GM-HOME]/bin/mpirun -np 3 /path/myprog.x 1200
^D
<ジョブID>
```

GM MPDデーモンが起動していない場合、mpirunへのPBSインターフェースにより、割り当てられたPBSホスト上でGM MPDデーモンがユーザーとして開始されます。デーモンは管理者またはユーザーによって以前に開始されている可能性があります。

例5-19： プロセスグループファイルprocgrpにリストされている複数のホスト上で複数の実行可能ファイルとMPICH-GMジョブを実行します。

ジョブスクリプト：

```
qsub -l select=2:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp

[MPICH-GM-HOME]/bin/mpirun -pg procgrp /path/mypro.x 1200
rm -f procgrp
^D
<ジョブID>
```

ジョブが実行されるとmpirunは次の警告メッセージを出力します。

```
Warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI
    processes spawned are not guaranteed to be under PBS-control.
```

この警告は、procgrpにリストされている任意のホストがPBSによって制御されていないため、そのホスト上のプロセスがPBSによって制御されていないことが原因で発行されます。

## 5.2.9.2 PBSでのMPICH-GMとrsh/sshの使用

PBSはrsh/sshを使用してMPICH-GMのmpirunへのインターフェースを提供します。PBSジョブの内部で実行されると、rsh/ssh経由で開始されたすべてのMPICH-GMプロセスがPBSによって追跡されるため、PBSはアカウンティングを実行してジョブを完全に制御できます。PBSジョブの外部で実行される場合、標準のmpirunが使用された場合と同様に動作します。

PBSの外部で使用するmpirunコマンドと同様のものを使用します。

### 5.2.9.2.i オプション

PBSジョブスクリプト内では、PBSインターフェースへのすべてのオプションは次のものを除いてmpirunと同様です。

-machinefile <ファイル>

file引数の内容は無視され$PBS_NODEFILEの内容と置き換えられます。

-np

指定しない場合、$PBS_NODEFILEの中のエントリ数が使用されます。起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

-pg

複数のホスト上の複数の実行可能ファイルのために-pgオプションを使用することはできますが、プロセスグループファイルでPBSホストのみが指定されていることを確認する必要があります。これは、非PBSホスト上で生成されたMPIプロセスはPBSによって制御される保証がないからです。

### 5.2.9.2.ii 例

例5-20： $PBS_NODEFILEにリストされているPBS割当ホスト間に展開されている64個のプロセスを使用して実行可能なMPICH-GMシングルジョブを実行します。

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
...
pbs-host64
```

```
qsub -l select=64:ncpus=1 -l place=scatter
mpirun -np 64 /path/myprog.x 1200
^D
<ジョブID>
```

例5-21： プロセスグループファイルprocgrpにリストされている複数のホスト上で複数の実行可能ファイルとMPICH-GMジョブを実行します。

```
qsub -l select=2:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp
mpirun -pg procgrp /path/mypro.x
rm -f procgrp
^D
<ジョブID>
```

ジョブが実行されるとmpirunは次の警告メッセージを出力します。

```
Warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI
    processes spawned are not guaranteed to be under the control of PBS.
```

この警告は、procgrpにリストされている任意のホストがPBSによって制御されていないため、そのホスト上のプロセスがPBSによって制御されていないことが原因で発行されます。

#### 5.2.9.3 制約

統合されたMPICH-GMで起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

## 5.2.10 PBSでのMPICH-MX

### 5.2.10.1 PBSでのMPICH-MXとMPDの使用

ラッパーは**非推奨**です。PBSはMPDを使用してMPICH-MXのmpirunへのインターフェースを提供します。PBSジョブの内部で実行されると、MPDデーモン経由で開始されたすべてのMPICH-MXプロセスがPBSによって追跡されるため、PBSはアカウンティングを実行してジョブを完全に制御できます。PBSジョブの外部で実行される場合、標準のMPDのMPICH-MX mpirunが使用された場合と同様に動作します。

PBSの外部で使用するmpirunコマンドと同様のものを使用します。MPDデーモンがまだ実行されていない場合、PBSインターフェースがデーモンを開始します。

#### 5.2.10.1.i オプション

PBSジョブスクリプト内では、PBSインターフェースへのすべてのオプションは次のものを除いてmpirunと同様です。

-m <ファイル>

file引数の内容は無視され$PBS_NODEFILEの内容と置き換えられます。

-np

指定しない場合、$PBS_NODEFILEの中のエントリ数が使用されます。起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

-pg

複数のホスト上の複数の実行可能ファイルのために-pgオプションを使用することはできますが、プロセスグループファイルでPBSホストのみが指定されていることを確認する必要があります。これは、非PBSホスト上で生成されたMPIプロセスはPBSによって制御される保証がないからです。

#### 5.2.10.1.ii MPDの起動と停止

PBS mpirunインターフェースは$PBS_NODEFILEにリストされている一意の各ホスト上で、環境変数RSHCOMMANDの値に基づいてrshまたはsshメソッドを使用してMPDデーモンを開始します。デフォルトはrshです。インターフェースは実行が終了するとMPDデーモンを停止します。

MPDデーモンが起動していない場合、mpirunへのPBSインターフェースにより、割り当てられたPBSホスト上でMXのMPDデーモンがユーザーとして開始されます。MPDデーモンは管理者またはユーザーによってすでに開始されている可能性があります。MPDバイナリへのパスを決定する、ユーザーが実行したmpirunのパス（GMまたはMX）をMPDデーモンは持たないため、このデーモンはPBS prologueスクリプト内では開始されません。

#### 5.2.10.1.iii 例

例5-22：$PBS_NODEFILEにリストされているPBS割当ホスト間に展開されている６４個のプロセスを使用して実行可能なMPICH-MXシングルジョブを実行します。

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
...
pbs-host64
```

```
qsub -l select=64:ncpus=1 -lplace=scatter
[MPICH-MX-HOME]/bin/mpirun -np 64 /path/myprog.x 1200
^D
<ジョブID>
```

MPDデーモンが起動していない場合、mpirunへのPBSインターフェースにより、割り当てられたPBSホスト上でMXのMPDデーモンがユーザーとして開始されます。MPDデーモンは管理者またはユーザーによってすでに開始されている可能性があります。

例5-23： プロセスグループファイルprocgrpにリストされている複数のホスト上で複数の実行可能ファイルとMPICH-MXジョブを実行します。

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe arg1 arg2" > procgrp
echo  "pbs-host2 1 username /x/x/b.exe arg1 arg2" >> procgrp
[MPICH-MX-HOME]/bin/mpirun -pg procgrp /path/myprog.x 1200
rm -f procgrp
^D
<ジョブID>
```

mpirunは次の警告メッセージを出力します。

```
Warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI
    processes spawned are not guaranteed to be under PBS-control
```

この警告は、procgrpにリストされている任意のホストがPBSによって制御されていないため、そのホスト上のプロセスがPBSによって制御されていないことが原因で発行されます。

### 5.2.10.2 PBSでのMPICH-MXとrsh/sshの使用

**非推奨**。PBSはrsh/sshを使用してMPICH-MXのmpirunへのインターフェースを提供します。PBSジョブの内部で実行されると、rsh/ssh経由で開始されたすべてのMPICH-MXプロセスがPBSによって追跡されるため、PBSはアカウンティングを実行してジョブを完全に制御できます。PBSジョブの外部で実行される場合、標準のmpirunが使用された場合と同様に動作します。

PBSの外部で使用するmpirunコマンドと同様のものを使用します。

### 5.2.10.2.i オプション

PBSジョブスクリプト内では、PBSインターフェースへのすべてのオプションは次のものを除いて標準のmpirunと同様です。

-machinefile <ファイル>

file引数の内容は無視され$PBS_NODEFILEの内容と置き換えられます。

-np

指定しない場合、$PBS_NODEFILEの中のエントリ数が使用されます。起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

-pg

複数のホスト上の複数の実行可能ファイルのために-pgオプションを使用することはできますが、プロセスグループファイルでPBSホストのみが指定されていることを確認する必要があります。これは、非PBSホスト上で生成されたMPIプロセスはPBSによって制御される保証がないからです。

### 5.2.10.2.ii 例

例5-24：$PBS_NODEFILEにリストされているPBS割当ホスト間に展開されている６４個のプロセスを使用して実行可能なMPICH-MXシングルジョブを実行します。

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
...
pbs-host64
```

```
qsub -l select=64:ncpus=1
mpirun -np 64 /path/myprog.x 1200
^D
<ジョブID>
```

例5-25：プロセスグループファイルprocgrpにリストされている複数のホスト上で複数の実行可能ファイルとMPICH-MXジョブを実行します。

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe arg1 arg2" > procgrp
echo  "pbs-host2 1 username /x/x/b.exe arg1 arg2" >> procgrp
mpirun -pg procgrp /path/myprog.x
rm -f procgrp
^D
<ジョブID>
```

mpirunは次の警告メッセージを出力します。

```
Warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI
    processes spawned are not guaranteed to be under PBS-control
```

この警告は、procgrpにリストされている任意のホストがPBSによって制御されていないため、そのホスト上のプロセスがPBSによって制御されていないことが原因で発行されます。

### 5.2.10.3 制約

統合されたMPICH-MXで起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

## 5.2.11 Linux上のMPICH2とPBS

Linux上では、PBSはMPICH2の`mpirun`へのインターフェースを提供します。PBSジョブの内部で実行されると、すべてのMPICH2プロセスがPBSによって追跡されるため、PBSはアカウンティングを実行してジョブを完全に制御できます。PBSジョブの外部で実行される場合、標準のMPICH2の`mpirun`が使用された場合と同様に動作します。

PBSの外部で使用する`mpirun`コマンドと同様のものを使用します。

PBSインターフェースでMPICH2の`mpirun`にPBSジョブを投入する際、ランクまたはMPIタスクの実際の数を`qsub`のselect指定の中で明示的に指定するようにしてください。そうしないと、“マシンファイル内のエントリが少なすぎます”というメッセージが出力されジョブが失敗します。

たとえば、次のような誤った指定の場合があります。

```
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

$PBS_NODEFILEで次のようなリストを作成します。

```
hostA
hostB
```

これでは２個のMPDデーモンしか起動されないため、`mpirun`の中の“`-np 3`”指定と競合します。

正しい方法は、次のいずれかを指定することです。

```
#PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2
```

これによって、$PBS_NODEFILE中で次のようなリストが作成されます。

```
hostA
hostB
hostB
```

さらに“`mpirun -np 3`”と一致しています。

### 5.2.11.1 オプション

PBSジョブスクリプト内で実行された場合､PBSインターフェースへのすべてのオプションは次のものを除いてMPICH2の`mpirun`と同様です。

-host, -ghost

実行されるホストを指定するのに使用します。無視されます。

-machinefile <ファイル>

file引数の内容は無視され$PBS_NODEFILEの内容と置き換えられます。

-localonly <プロセスの数>

ローカルで実行される*プロセスの数*を指定するのに使用します。サポートされていません。次のような同等の引数を代わりに使用してください。

"-np <x> -localonly".

-np

ユーザーが-npオプションを指定しない場合、PBSインターフェースはMPICH2にデフォルト値を提供しません。標準のmpirunによって適切なデフォルト値（通常は1）が決定されます。起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

## 5.2.11.2 MPDの起動と停止

インターフェースによって$PBS_NODEFILEにリストされている各ホスト上でMPDデーモンが起動されたことが確認されます。さらに、MPIジョブ実行の最後でMPDデーモンの停止も確認されます。

## 5.2.11.3 例

例5-26：$PBS_NODEFILEにリストされているPBS割当ホスト間に展開されている６個のプロセスを使用して実行可能なMPICH2シングルジョブを実行します。３つのホストしか使用できません。

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

ジョブスクリプト：

```
# mpirunが３つのホストに分散された６つのプロセスを実行します
# $PBS_NODEFILEにリストされます
mpirun -np 6 /path/myprog.x 1200
```

ジョブスクリプトの実行：

```
qsub -l select=6:ncpus=1 -lplace = scatter job.script
<ジョブID>
```

例5-27： mpirunの中で$PBS_NODEFILEおよびmpiexec引数を使用して、複数のホスト上で複数の実行可能ファイルとMPICH2ジョブを実行します。

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

ジョブスクリプト：

```
#PBS -l select=3:ncpus=2:mpiprocs=2
mpirun -np 2 /tmp/mpitest1 : -np 2 /tmp/mpitest2 : -np 2 /tmp/mpitest3
```

ジョブの実行：

```
qsub job.script
```

例5-28：mpirun -configfileオプションおよび$PBS_NODEFILEを使用して、複数のホスト上で複数の実行可能ファイルとMPICH2ジョブを実行します。

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

ジョブスクリプト：

```
#PBS -l select=3:ncpus=2:mpiprocs=2
echo "-np 2 /tmp/mpitest1" > my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file
mpirun -configfile my_config_file
rm -f my_config_file
```

ジョブの実行：

```
qsub job.script
```

### 5.2.11.4 制約

統合されたMPICH2で起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

## 5.2.12 Windows上のMPICH2 1.4.1p1とPBS

Windowsでは、PBSには、pbs_mpich2_mpirun.batと呼ばれるMPICH2 1.4.1p1用のラッパースクリプトが$PBS_EXEC¥binに用意されています。MPICH2 mpirunの代わりにこのスクリプトを呼び出します。オプションはすべて、このスクリプトからmpirunに渡されます。

## 5.2.13 PBSでのMVAPICH

ラッパーは**非推奨**です。PBSはMVAPICH mpirunへのmpirunインターフェースを提供します。PBS提供のmpirunを使用する場合、PBSはMVAPICHプロセスをすべて追跡し、アカウンティングを実行して、ジョブを完全に制御できます。PBS管理者はMVAPICHをPBSと統合できるため、ジョブスクリプトのMVAPICH mpirunの場所にPBS提供のmpirunを使用できます。

MVAPICHはジョブがInfiniBandを使用できるようにします。

### 5.2.13.1 MVAPICH mpirunコマンドへのインターフェース

PBSジョブの外部で実行される場合、mpirunへのPBS提供インターフェースは標準のMVAPICH mpirunが使用された場合と同様に動作します。

PBSジョブスクリプト内で実行された場合、PBSインターフェースへのすべてのオプションは次のものを除いてMVAPICHのmpirunと同様です。

-map

mapオプションは無視されます。

-machinefile <ファイル>

machinefileオプションは無視されます。

-exclude

excludeオプションは無視されます。

-np

-npオプションを指定しない場合、PBSでは$PBS_NODEFILEの中のエントリ数を使用します。起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

### 5.2.13.2 例

例5-29：$PBS_NODEFILEにリストされているPBS割当ホスト間に展開されている６個のランクを使用して実行可能なMVAPICHシングルジョブを実行します。

$PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

job.scriptの内容：

```
#mpirunは$PBS_NODEFILEで各行にマッピングされた６つのプロセスを実行します
mpirun -np 6 /path/myprog
```

ジョブスクリプトの実行：

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<ジョブID>
```

### 5.2.13.3 制約

統合されたMVAPICHで起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

## 5.2.14 PBSでのMVAPICH2

PBSはMVAPICH2のmpiexecへのmpiexecインターフェースを提供します。PBS提供のmpiexecを使用する場合、PBSはMVAPICH2プロセスをすべて追跡し、アカウンティングを実行して、ジョブを完全に制御できます。PBS管理者はMVAPICH2をPBSと統合できるため、ジョブスクリプトのMVAPICH2 mpirunの場所にPBS提供のmpirunを使用できます。

MVAPICH2はジョブがInfiniBandを使用できるようにします。

### 5.2.14.1 MVAPICH2 mpiexecコマンドへのインターフェース

PBSジョブの外部で実行される場合、標準のMVAPICH2のmpiexecが使用された場合と同様に動作します。

PBSジョブスクリプト内で実行した場合、PBSインターフェースへのすべてのオプションは次のものを除いてMVAPICH2の mpiexecと同様です。

-host

hostオプションは無視されます。

-machinefile <ファイル>

fileオプションは無視されます。

-mpdboot

mpdbootがmpiexecの前に呼び出されない場合、PBSによって割り当てられた各ホストでMPDデーモンが起動されるように、mpdbootはmpiexecの実行の前に自動的に呼び出されます。

### 5.2.14.2 MPDの起動と停止

インターフェースによって$PBS_NODEFILEにリストされている各ホスト上でMPDデーモンが起動されたことが確認されます。さらに、MPIジョブ実行の最後でMPDデーモンの停止も確認されます。

### 5.2.14.3 例

例5-30：$PBS_NODEFILEにリストされているホストで６つのランクを使用して実行可能なMVAPICH2シングルジョブを実行します。

$PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

ジョブスクリプト：

```
mpiexec -np 6 /path/mpiprog
```

ジョブスクリプトの実行：

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<ジョブID>
```

例5-31： デフォルトファイル“mpd.hosts”にリストされている複数のホスト上で複数の実行可能ファイルとMVAPICH2 MPIジョブを起動します。ここで、host1で２つのランクのprog1、host2で２つのランクのprog2、およびhost3で２つのランクのprog2（すべてコマンドラインで指定）を使用して、実行可能プログラムprog1とprog2を実行します。

$PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

ジョブスクリプト：

```
mpiexec -n 2 prog1 : -n 2  prog2 : -n 2  prog2
```

ジョブスクリプトの実行：

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<ジョブID>
```

例5-32： デフォルトファイル“mpd.hosts”にリストされている複数のホスト上で複数の実行可能ファイルとMVAPICH2 MPIジョブを起動します。host1で２つのランクのprog1、host2で２つのランクのprog2、およびhost3で２つのランクのprog2（すべて-configfileオプションで指定）を使用して、実行可能プログラムprog1とprog2を実行します。

$PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

ジョブスクリプト：

```
echo "-n 2 -host host1 prog1" > /tmp/jobconf
echo "-n 2 -host host2 prog2" >> /tmp/jobconf
echo "-n 2 -host host3 prog2" >> /tmp/jobconf
mpiexec -configfile /tmp/jobconf
rm /tmp/jobconf
```

ジョブスクリプトの実行：

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<ジョブID>
```

#### 5.2.14.4 制約

MVAPICH2で起動可能なランクの最大数は$PBS_NODEFILEのエントリ数になります。

## 5.2.15 PBSでのOpen MPI

Open MPIはLinux上でPBSと統合できるため、PBSはすべてのジョブプロセスのリソース使用量を追跡し、プロセスにシグナルを送信し、アカウンティングを実行できます。PBS管理者はOpen MPIとPBSを統合できます。

### 5.2.15.1 PBSでのOpen MPIの使用

MPIコマンドラインに変更を加えずに、Open MPIを使用してPBSでジョブを実行できます。

## 5.2.16 PBSでのPlatform MPI

Platform MPIはLinux上でPBSと統合できるため、PBSはすべてのジョブプロセスのリソース使用量を追跡し、プロセスにシグナルを送信し、アカウンティングを実行できます。PBS管理者はPlatform MPIとPBSを統合できます。

### 5.2.16.1 PBSでのPlatform MPIの使用

MPIコマンドラインに変更を加えずに、Platform MPIを使用してPBSでジョブを実行できます。

### 5.2.16.2 環境の設定

デフォルトの`rsh`を指定変更するには、ジョブスクリプト内でPBS_RSHCOMMANDを設定します。

```
export PBS_RSHCOMMAND=<rshコマンド>
```

## 5.2.17 PBSでのHPE MPI

PBSは、サポートされているバージョンのHPE MPIを実行しているマルチvnodeマシン上でHPE MPIと共に使用するために独自の`mpiexec`を提供します。PBS提供の`mpiexec`を使用する場合、PBSはすべてのジョブプロセスのリソース使用量を追跡し、プロセスにシグナルを送信し、アカウンティングを実行できます。PBS `mpiexec`には標準の`mpiexec`インターフェースがあります。

システムがPBS `mpiexec`向けに設定されているか否かをPBS管理者にお問い合わせください。

### 5.2.17.1 PBSでのHPE MPIの使用

MPIジョブは単一のHPEシステムまたは複数のHPEシステムで起動できます。複数のHPEシステム上で実行されるMPIジョブに関しては、PBSがマルチホストジョブを管理します。たとえば、host1およびhost2という名前の２台のHPEシステムがあり、それらのマシン上で`mympi1`および`mympi2`と呼ばれる２つのアプリケーションを実行する場合、次のようなジョブスクリプトを作成できます。

```
mpiexec -host host1 -n 4 mympi1 : -host host2 -n 8 mympi2
```

PBSがジョブのプロセスの管理および追跡を行います。ジョブが完了すると、PBSが引き続きクリーンアップを行います。

PBSにより選択されたplacement setの中でMPIジョブを実行できます。

#### 5.2.17.2 前提条件

PBS ジョブ内で HPE MPI と共に MPI を使用するには、MPI を呼び出す前に次をジョブスクリプトに追加する必要がある場合があります。

```
module load mpt
```

#### 5.2.17.3 ノードボードへのジョブの適合

PBSは単一ノードボードに収まるジョブを１つのノードボードのみに挿入しようとします。ただし、唯一利用可能なCPUが別のノードボード上にあり、vnodeが既存のジョブに排他的に割り当てられておらず、ジョブがvnodeを共有できる場合は、ジョブは別のノードボード上で実行されます。

#### 5.2.17.4 ジョブのチェックポイントと中断

HPEシステムでジョブを中断する場合は、PBSの中断機能を使用します。

HPE システムでは、アプリケーションレベルのチェックポイントを使用してジョブにチェックポイントを設定します。OSレベルのチェックポイントはありません。

サスペンドまたはチェックポイントが行われたジョブは、元のノードボードで再開されます。

#### 5.2.17.5 CSAの使用

HPEシステムのCSAに対するPBSのサポートは利用できません。HPEシステムのCSA機能はPBSから**削除**されています。

## 5.3 PBSでのPVMの使用

Parallel Virtual Machine（PVM）プログラムを実行するには、`pvmexec`コマンドを使用します。PVMはPBSに統合されていません。PBSはプライマリvnode上のジョブプロセスのモニタリング、制御、アカウンティングのみに限られています。

### 5.3.1 pvmexecコマンドの引数

`pvmexec`コマンドでは、並列ジョブを起動するホストのリストを渡す引数`hostfile`を取ります。

### 5.3.2 PVMデーモンの使用

$PBS_NODEFILEにリストされたホスト上でPVMデーモンを起動するには

1. このリストの最初のホスト上でPVMコンソールを起動します。
2. `jobname.o<PBSジョブID>`という名前の標準出力ファイルにホストを出力します。

```
echo conf | pvm $PBS_NODEFILE
```

PVMコンソールを終了し、PVMデーモンは稼働したままにするには、次のコマンドを使用します。

```
quit
```

PVMデーモンを停止して、PVMコンソールを再起動して終了するには、次のコマンドを使用します。

```
echo halt | pvm
```

### 5.3.3 PVMジョブの投入

PVMジョブをPBSに投入するには、次のコマンドを使用します。

*qsub <ジョブスクリプト>*

### 5.3.4 例

例5-33： PVMジョブをPBSに投入するには、次のコマンドを使用します。

```
qsub  your_pvm_job
```

your_pvm_jobのスクリプト例を以下に示します。

```
#PBS -N pvmjob
#PBS -V
cd $PBS_O_WORKDIR
echo conf   | pvm $PBS_NODEFILE
echo quit | pvm
./my_pvm_program
echo halt | pvm
```

例5-34： PVMジョブに対するPBSスクリプトの例：

```
#PBS -N pvmjob
#
pvmexec a.out -inputfile data_in
```

## 5.4 PBSでのOpenMPの使用

PBS Professionalでは、ジョブのリソース要求に基づき、ジョブの環境でOMP_NUM_THREADS変数を設定することにより、OpenMPアプリケーションをサポートします。OpenMPランタイムでは、OMP_NUM_THREADSの値を取得して、スレッドを適切に生成します。

MoMは`select`ステートメントの最初のチャンクに基づいてOMP_NUM_THREADSの値を設定します。最初のチャンクでompthreadsを要求すると、MoMは環境変数をompthreadsの値に設定します。最初のチャンクでompthreadsを要求しない場合、OMP_NUM_THREADSは該当するチャンクのncpusリソースの値に設定されます。selectステートメントの最初のチャンクのncpusまたはompthreadsを要求しない場合、OMP_NUM_THREADSは*1*に設定されます。

OMP_NUM_THREADS環境変数の値を直接設定することはできません。MoMがこの設定を指定変更します。

ompthreadsリソースの定義については、『PBS Professional Reference Guide』の「Resources Built Into PBS」（265ページ）をご参照ください。

例5-35： 10GBのメモリを必要とする２個のCPU、２個のスレッドのジョブの場合、単一のチャンクとしてOpenMPジョブを投入するには、次のコマンドを使用します。

```
qsub -l select=1:ncpus=2:mem=10gb
```

例5-36： プロセスごとに１個のスレッドを持ち、６４個のMPIプロセスを備えたMPIアプリケーションを実行するには、次のコマンドを使用します。

```
#PBS -l select=64:ncpus=1
mpiexec -n 64 ./a.out
```

例5-37： プロセスごとに４個のOpenMPスレッドを持ち、６４個のMPIプロセスを備えたMPIアプリケーションを実行するには、次のコマンドを使用します。

```
#PBS -l select=64:ncpus=4
mpiexec -n 64 omplace -nt 4 ./a.out
```

または

```
#PBS -l select=64:ncpus=4:ompthreads=4
mpiexec -n 64 omplace -nt 4 ./a.out
```

### 5.4.1 CPUより少ないスレッドの実行

ホスト上でOpenMPアプリケーションを実行していて、要求されたCPU数よりも少ないスレッドを実行する必要が生じる場合があります。これは、スレッドがマルチコアプロセッサシステムの共有リソース（コア間で共有されたキャッシュまたはメモリなど）に排他的にアクセスする必要があるためである場合があります。

例5-38： １６個のCPUと８個のスレッドを持つ１つのチャンクが必要な場合、次のコマンドを使用します。

```
qsub -l select=1:ncpus=16:ompthreads=8
```

### 5.4.2 CPUより多いスレッドの実行

ホスト上でOpenMPアプリケーションを実行していて、各スレッドがI/Oバインドであるため要求されたCPU数よりも多くのスレッドを実行する必要が生じる場合があります。

例5-39： ８個のCPUと１６個のスレッドを持つ１つのチャンクが必要な場合、次のコマンドを使用します。

```
qsub -l select=1:ncpus=8:ompthreads=16
```

### 5.4.3 PBSでのOpenMPの使用の注意事項

PBSノードファイルには正しい数のエントリが含まれているように、ジョブに対して正しい数のMPIランクを要求していることを確認します。83ページの第5.1.3節「チャンクごとのMPIプロセス数の指定」をご参照ください。

## 5.5 MPI-OpenMPハイブリッドジョブ

MPIのマルチスレッドのジョブでは、すべてのチャンクについて、チャンクごとのスレッド数は、最初のチャンクで（明示的または黙示的に）要求されたスレッド数に設定されます。ただし、PBS TM APIと統合されたMPIは除きます。

PBS TMインターフェースに統合されたMPI（Open MPI）に対して、各チャンクにompthreadsリソースを個別に指定して、各チャンクのスレッド数を個別に指定できます。

大部分のMPI、OMP_NUM_THREADS、およびNCPUS環境変数は、デフォルトで、最初のチャンクで要求されたncpus数に設定されます。

MPIとマルチスレッドのジョブがある場合は、MPIプロセスごとに１つのチャンクを要求するか、チャンクごとに必要なMPIプロセス数にmpiprocsを設定できます。83ページの第5.1.3節「チャンクごとのMPIプロセス数の指定」をご参照ください。

## 5.5.1 例

例5-40： それぞれが１個のMPIプロセス、２個のCPU、および２個のスレッドを持つ４つのチャンクを要求するには、次のコマンドを使用します。

```
qsub -l select=4:ncpus=2
```

または

```
qsub -l select=4:ncpus=2:ompthreads=2
```

例5-41： それぞれが２個のCPUと４個のスレッドを持つ４つのチャンクを要求するには、次のコマンドを使用します。

```
qsub -l select=4:ncpus=2:ompthreads=4
```

例5-42： ２個のプロセッサを備えた複数のマシン上で、それぞれが２個のスレッドを持つ１６個のMPIプロセスを要求するには、次のコマンドを使用します。

```
qsub -l select=16:ncpus=2
```

例5-43： それぞれが８個のCPU、８個のMPIタスク、および４個のスレッドを持つ２つのチャンクを要求するには、次のコマンドを使用します。

```
qsub -l select=2:ncpus=8:mpiprocs=8:ompthreads=4
```

例5-44： 次の場合、

```
qsub -l select=4:ncpus=2
```

VnodeA の CPU ４個、VnodeB の CPU ２個、および VnodeC の CPU ２個によってこの要求が満たされるため、$PBS_NODEFILEには次のように書き込まれます。

```
VnodeA
VnodeA
VnodeB
VnodeC
```

４個のMPIプロセスに対応する４個のPBSタスクに対してOpenMP環境変数が次のように設定されます。

- VnodeAのPBSタスク#1用： OMP_NUM_THREADS=2 NCPUS=2
- VnodeAのPBSタスク#2用： OMP_NUM_THREADS=2 NCPUS=2
- VnodeBのPBSタスク#3用： OMP_NUM_THREADS=2 NCPUS=2
- VnodeCのPBSタスク#4用： OMP_NUM_THREADS=2 NCPUS=2

例5-45： 次の場合、

```
qsub -l select=3:ncpus=2:mpiprocs=2:ompthreads=1
```

３つのノード（VnodeA、VnodeB、および VnodeC）から CPU を２個ずつ取得してこの要求が満たされるため、$PBS_NODEFILEには次のように書き込まれます。

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

６個のMPIプロセスに対応する６個のPBSタスクに対してOpenMP環境変数が次のように設定されます。

- VnodeAのPBSタスク#1用： OMP_NUM_THREADS=1 NCPUS=1
- VnodeAのPBSタスク#2用： OMP_NUM_THREADS=1 NCPUS=1
- VnodeBのPBSタスク#3用： OMP_NUM_THREADS=1 NCPUS=1
- VnodeBのPBSタスク#4用： OMP_NUM_THREADS=1 NCPUS=1
- VnodeCのPBSタスク#5用： OMP_NUM_THREADS=1 NCPUS=1
- VnodeCのPBSタスク#6用： OMP_NUM_THREADS=1 NCPUS=1

例5-46： すべて同じHPEシステムに配置され、それぞれが1つのプロセスを実行しているN個の各チャンク上で、2個のスレッドを実行するには、次のコマンドを使用します。

```
qsub -l select=N:ncpus=2 -l place=pack
```

このコマンドでは、OMP_NUM_THREADS=2であるため、単一ホスト上で、プロセスごとに２つのOpenMPスレッドを持つ*N*個のプロセスが開始されます。

# 6 ジョブの動作方法の制御

## 6.1 ジョブ終了ステータスの使用

PBSは、ジョブ終了ステータスを、epilogueへの入力として使用したり、依存するジョブを実行するかどうかの判定に使用したりできます。Linuxで実行する場合、ジョブ終了ステータスを正しく取得していることを確認してください。6ページの第1.4.2.4節「正しいジョブ終了ステータスの取得」をご参照ください。

ジョブ終了コードは『PBS Professional Administrator's Guide』の第10.9節「Job Exit Status Codes」（469ページ）にリストされています。

ジョブアレイの終了ステータスは、完了された各サブジョブのステータスによって決定され、すべての有効なサブジョブが完了した際にのみ使用できます。完了したサブジョブの個々の終了ステータスはepilogueに渡され、そのサブジョブの'E'アカウンティングログ記録内で参照できます。167ページの「ジョブアレイの終了ステータス」をご参照ください。

### 6.1.1 終了ステータスの注意事項

- qsubは、通常は、ブロッキングジョブの終了ステータスで終了しますが、ブロッキングとインタラクティブの両方の特性を持つジョブを投入した場合、PBSはそのジョブの終了ステータスを返しません。128ページの第6.10節「ジョブの終了までqsubを待機させる」をご参照ください。
- ブロッキングジョブの場合、終了ステータスは、ステージングが完了する前に返されます。128ページの第6.10.2節「ジョブのブロッキングの注意事項」をご参照ください。
- インタラクティブジョブの終了ステータスは、実際の終了ステータスに関係なく、常に0（ゼロ）として記録されます。

## 6.2 ジョブの依存関係の使用

PBSでは、2つ以上のジョブの間の依存関係を指定することができます。依存関係は、次のようなさまざまなタスクで役立ちます。

- セット内のジョブを実行する順序の指定
- 別のジョブでエラーが発生した場合にのみジョブを実行するように要求
- 特定のジョブが実行を開始または終了するまでジョブを保留

ジョブごとの依存関係の数に制限はありません。

ジョブj1に依存しており、j1の実行後でなければ実行できない1つまたは複数のジョブj2... jNがある場合、j1を削除すると、PBSによりジョブj2... jNも削除されます。j1の実行に失敗した後でのみ実行可能なジョブj2... jNがある場合、j1を削除すると、PBSによりジョブj2... jNの依存関係は解除され、これらが実行できるようになります。

## 6.2.1 ジョブの依存関係の構文

qsubで“-W depend=<依存関係リスト>”オプションを使用することで、ジョブ間の依存関係が定義されます。*依存関係リスト*の構文は、以下のとおりです。

*<タイプ>:<引数リスト>[,<タイプ>:<引数リスト> ...]*

ここで、onタイプの場合を除き、*引数リスト*は１つまたは複数のPBSジョブIDになります。形式は以下のとおりです。

*<ジョブID>[:<ジョブID> ...]*

使用可能な依存関係タイプを以下に示します。

after:<引数リスト>

このジョブは、*引数リスト*内のすべてのジョブが実行を開始した後にのみ、開始できます。

afterok:<引数リスト>

このジョブは、*引数リスト*内のすべてのジョブがエラーなく終了した後にのみ、開始できます。

afternotok:<引数リスト>

このジョブは、*引数リスト*内のすべてのジョブがエラーで終了した後にのみ、開始できます。

afterany:<引数リスト>

このジョブは、*引数リスト*内のすべてのジョブがエラーの有無に関係なく実行を終了した後に、開始できます。
このジョブは、*引数リスト*内のジョブが実行されることなく削除された場合は実行されません。

before:<引数リスト>

*引数リスト*内のジョブは、指定したジョブが実行を開始した後にのみ、開始できます。onタイプのジョブを、他のジョブの前に投入する必要があります。

beforeok:<引数リスト>

*引数リスト*内のジョブは、このジョブがエラーなしで実行を終了した後にのみ、開始できます。

beforenotok:<引数リスト>

このジョブがエラーありで実行を終了すると、*引数リスト*内のジョブが開始します。

beforeany:<引数リスト>

*引数リスト*内のジョブは、指定したジョブがエラーの有無に関係なく実行を終了した後にのみ、開始できます。
他のジョブを投入する前に、実行するジョブのon依存関係を使用する必要があります。

on:count

このジョブは、他のジョブで*count*依存関係が満たされた後にのみ、開始できます。このタイプは、上記のbeforeタイプのいずれかと組み合わせて使用されます。*count*は*0*より大きい整数を示します。

runone:<ジョブID>

現在のジョブおよび*ジョブID*のジョブを、ジョブのセットに配置します。PBSは最終的にこのセットの中から1つのジョブのみを実行します。ジョブをセットに追加するには、すでにセット内にある別のジョブのジョブIDを指定します。

ジョブ依存関係は、dependジョブ属性で制御します。これは、コマンドラインでqsubを使用して、またはPBS指示文で設定できます。

```
qsub -W depend=...
#PBS depend=...
```

### 6.2.1.1 使用可能な最初のリソースでのジョブの実行

ジョブが別のリソースセットで実行可能であっても、最初に使用可能になったリソースでジョブを実行することができます。大きなリソースセットを長い間待つのではなく、小さなリソースセットでできるだけ早く柔軟なジョブを開始することができます。または、特定のリソースを使用したい場合に、他のリソースを使用することもできます（たとえば、特定のプロセッサを使用したいが、使用できるのがもう1つのプロセッサだけの場合、そのプロセッサで実行することができます）。

各ジョブがその他に対して“runone”の依存関係を持つジョブのセットを投入すると、PBSは“runoneのセット”内のジョブのうち1つだけを実行します。PBSは自動的にそれらのジョブをrunoneセットにグループ化します。runoneセット内のジョブは、別々のスクリプトを実行できます。

セット内のいずれかのジョブが開始されると、PBSは他のジョブに対してシステム保留を適用します。他のジョブの保留は、実行中のジョブが以下により再キューイングされると解除されます：

- qrerunを使用
- ノード障害時の再キューイングが生じた場合

セット内の別のジョブの削除：

- 終了ステータスによらず、ジョブが終了した場合
- 実行中のジョブが削除された場合

ジョブをセットのメンバーとして識別するには、このジョブに、以前投入したセットメンバーに対する“runone”依存関係を付与します。たとえば、3つのジョブがあり、それぞれが別のリソースで実行されるものとします。この3つのジョブをrunoneセットとして投入するには、次のように指定します。

```
qsub -lselect=200:ncpus=16 -lwalltime=1:00:00 myscript.sh
10.myserver
qsub -lselect=100:ncpus=16 -lwalltime=2:00:00 -Wdepend=runone:10 myscript.sh
11.myserver
qsub -lselect=50:ncpus=16 -lwalltime=4:00:00 -Wdepend=runone:10 myscript.sh
12.myserver
```

## 6.2.2 ジョブの依存関係の例

例6-1： job1、job2、およびjob3の３つのジョブがあり、job3は、job1とjob2が*終了した後*で開始するようにします。

```
qsub job1
16394.jupiter
qsub job2
16395.jupiter
qsub -W depend=afterany:16394:16395 job3
16396.jupiter
```

例6-2： job2は、job1がエラーなしで終了した*場合にのみ*開始するようにします。

```
qsub job1
16397.jupiter
qsub -W depend=afterok:16397 job2
16396.jupiter
```

例6-3： job1は、job2とjob3の前に実行する必要があります。beforeany依存関係を使用するには、on依存関係を使用する必要があります。

```
qsub -W depend=on:2 job1
16397.jupiter
qsub -W depend=beforeany:16397 job2
16398.jupiter
qsub -W depend=beforeany:16397 job3
16399.jupiter
```

## 6.2.3 ジョブアレイの依存関係

ジョブの依存関係は次のような間でサポートされています。

- ジョブとジョブの間
- ジョブアレイとジョブアレイの間
- ジョブアレイとジョブの間
- ジョブとジョブアレイの間

ジョブの依存関係は、サブジョブまたはサブジョブの範囲についてはサポートされていません。

## 6.2.4 ジョブの依存関係に関する注意事項とアドバイス

### 6.2.4.1 正しい終了ステータスが必要

Linuxでは、ジョブ終了ステータスを正しく取得していることを確認してください。113ページの第6.1節「ジョブ終了ステータスの使用」をご参照ください。

### 6.2.4.2 依存関係に必要な権限

beforeタイプを使用するには、*引数リスト*内のジョブを変更する権限が必要です。権限を保有していない場合、依存関係が拒否され、新規ジョブは中止されます。

#### 6.2.4.3 ジョブの履歴に関する警告

ジョブ履歴を有効にすると、依存するジョブの動作が変更されます。終了したジョブj2にジョブj1が依存し、j2の履歴がPBSで保持されている場合、j1の依存関係は解除され、適切なアクションが取られます。終了したジョブj3にジョブj1が依存し、j3がジョブ履歴から削除された場合、j1は拒否されます。これは、PBSの以前のバージョンでジョブがシステム内に存在しなくなる動作と同じです。

#### 6.2.4.4 エラーのレポーティング

PBSは、ジョブを受け入れた後、その存在、状態、または条件のエラーをチェックします。エラーが存在する場合、そのエラーに関するメールを送信して、ジョブを削除します。

## 6.3 ジョブの実行時間の調整

この機能は、PBS Professional 12.0で追加されました。

### 6.3.1 Shrink-to-fitジョブ

PBSでは、実行時間を使用可能なスケジューリングスロットに合わせるように調整できるジョブを投入できます。ジョブの最小および最大実行時間は、min_walltimeおよびmax_walltimeリソースで指定されます。PBSは実際のwalltimeを選択します。min_walltimeを要求するジョブは、**shrink-to-fit**ジョブです。

#### 6.3.1.1 Shrink-to-fitジョブの要件

ジョブがshrink-to-fitであるためには、min_walltimeの値を持つ必要があります。Shrink-to-fitジョブはmax_walltimeを要求する必要はありませんが、max_walltimeを要求してmin_walltimeを要求しないとエラーになります。

min_walltimeの値を持たないジョブは、shrink-to-fitジョブではなく、walltimeを指定できます。

#### 6.3.1.2 Shrink-to-fitジョブとNon-shrink-to-fitジョブの比較

shrink-to-fitジョブとnon-shrink-to-fitジョブの唯一の相違点は、ジョブのwalltimeの処理方法です。PBSはジョブの実行時にwalltimeを設定します。ジョブの実行前に存在したwalltime値は無視されます。

### 6.3.2 Shrink-to-fitジョブの使用

必要と予想されたより短い時間で実行でき、有効な進展のあるジョブがある場合、最大限に利用するためにこれらをshrink-to-fitジョブにできます。

次の場合にshrink-to-fitジョブを使用できます。

- 内部的にチェックポイント処理されたジョブ。これは、より大きな仕事の一部であるジョブを含みます。ここで、ジョブは強制終了される前に可能な限り多くの作業を行い、その仕事内の次のジョブは以前のジョブを終了したところから再開されます。
- 定期的なPBSチェックポインティングを使用するジョブ
- 実際の実行時間が予想された時間よりもかなり短い可能性のあるジョブ
- システム保守の専用時間があり、シャットダウンまでの時間スロットを利用したい場合に、終了前にジョブが強制終了されるというリスクがある場合は、投機的なshrink-to-fitジョブを実行できます。同様に投機的なジョブは予約を開始する直前の時間を活用できます。

- いくつかの作業を終了するための投機的な試みとして実行しても構わないジョブ

## 6.3.3 Shrink-to-fit ジョブの実行時間

### 6.3.3.1 Shrink-to-fit ジョブの実行時間範囲の設定

ジョブが min_walltime を要求することのみが shrink-to-fit ジョブにするために必要となります。min_walltime を要求せずに max_walltime を要求するとエラーになります。

min_walltime および max_walltime を要求すると、ジョブの実行時間範囲を設定できます。たとえば、次のようになります。

*qsub -l min_walltime=<最小walltime>, max_walltime=<最大walltime> <ジョブスクリプト>*

### 6.3.3.2 Shrink-to-fit ジョブのための walltime の設定

shrink-to-fit ジョブに対して、walltime がジョブに対して指定されているかどうかに関係なく、PBS は min_walltime と max_walltime の値に基づいて walltime リソースを設定します。

PBS は初めに各 shrink-to-fit ジョブを調べ、長さがジョブの min_walltime と max_walltime の間である時間スロットを探します。ジョブがどこかに一致する場合、PBS はジョブの walltime を時間スロットに一致する期間に設定し、ジョブを実行します。walltime に選択した値は、ジョブの Resource_List.walltime 属性で表示できます。以前の実行など、どこから来たかに関係なく既存の walltime の値は、新しく計算された実行時間にリセットされます。

shrink-to-fit ジョブが複数回実行されると、PBS はジョブの実行時間を min_walltime と max_walltime の間の使用可能な時間スロットに合うように再計算し、ジョブが実行されるたびにジョブの walltime をリセットします。

マルチ vnode ジョブの場合、PBS はジョブに必要なすべてのチャンクで機能する walltime を選択し、配置指定に基づいてジョブチャンクを配置します。

## 6.3.4 Shrink-to-fit ジョブと Non-shrink-to-fit ジョブの変更

### 6.3.4.1 min_walltime と max_walltime の変更

`qalter` コマンドを使用して、shrink-to-fit ジョブの min_walltime および / または max_walltime を変更できます。変更は、現在のスケジューリングサイクルの終了後に有効になります。変更はキューイングされているジョブにのみ影響します。実行中のジョブは再実行されない限り影響されません。

#### 6.3.4.1.i non-shrink-to-fit ジョブから shrink-to-fit ジョブへの変換

`qalter` コマンドを使用して min_walltime と max_walltime の値を設定することによって、通常の non-shrink-to-fit ジョブを shrink-to-fit ジョブに変換できます。

変更は、現在のスケジューリングサイクルの終了後に有効になります。変更はキューイングされているジョブにのみ影響します。実行中のジョブは再実行されない限り影響されません。

#### 6.3.4.1.ii shrink-to-fit ジョブから non-shrink-to-fit ジョブへの変換

shrink-to-fit ジョブを通常の non-shrink-to-fit ジョブに変換するには、`qalter` コマンドを使用して以下を実行します。

- ジョブの walltime を max_walltime の値に設定します。
- min_walltime の値を無設定にします。
- max_walltime の値を無設定にします。

## 6.3.5 ジョブの実行時間の表示

### 6.3.5.1 min_walltime と max_walltime の表示

qstat -f を使用して、min_walltime と max_walltime の値を表示できます。以下に例を示します。

```
% qsub -lmin_walltime=01:00:15, max_walltime=03:30:00 job.sh
<ジョブID>
% qstat -f <ジョブID>
...
Resource_List.min_walltime=01:00:15
Resource_List.max_walltime=03:30:00
```

tracejob を使用して、ジョブのリソースリストとして max_walltime と min_walltime を表示できます。以下に例を示します。

```
12/16/2011 14:28:55  A    user=pbsadmin group=Users project=_pbs_project_default
…
    Resource_List.max_walltime=10:00:00
    Resource_List.min_walltime=00:00:10
```

### 6.3.5.2 Shrink-to-fit ジョブのための walltime の表示

PBS はジョブの実行時にのみ、そのジョブの walltime を設定します。ジョブの実行中は、qstat -f により walltime を表示できます。ジョブが実行されていない間は、実際の walltime を表示できません。walltime 用に設定された値があっても、この値は無視されます。

ジョブ履歴を保持している場合、終了した shrink-to-fit ジョブの walltime 値を表示できます。479 ページの第 10.15 節「Managing Job History」をご参照ください。

## 6.3.6 Shrink-to-fit ジョブのライフサイクル

### 6.3.6.1 shrink-to-fit ジョブの実行

Shrink-to-fit ジョブは non-shrink-to-fit ジョブと同じように開始されます。

### 6.3.6.2 Shrink-to-fit ジョブの終了

shrink-to-fit ジョブが PBS の設定した walltime を超えると、non-shrink-to-fit ジョブが walltime を超えた場合に強制終了されるのと同様に、PBS によって強制終了されます。

### 6.3.7 min_walltime リソースと max_walltime リソース

max_walltime

Shrink-to-fit ジョブに対して可能な最大 walltime。ジョブの実際の walltime は、max_walltime と min_walltime の間です。PBS は shrink-to-fit ジョブに対して walltime を設定します。このリソースが指定された場合は、min_walltime も指定する必要があります。min_walltime より大きいか同じである必要があります。resources_min または resources_max に使用することはできません。ジョブアレイや予約に関しては設定できません。指定していない場合、PBS は最長の時間スロットとして 5 年を使用します。-l select の外部に限り要求できます。消費不可能。デフォルト：なし。タイプ：duration。Python タイプ：pbs.duration

min_walltime

Shrink-to-fit ジョブに対して可能な最小 walltime。このリソースを指定した場合、ジョブは shrink-to-fit ジョブです。この属性が設定されると、PBS はジョブの walltime を設定します。ジョブの実際の walltime は、max_walltime と min_walltime の間です。max_walltime より小さいか同じである必要があります。resources_min または resources_max に使用することはできません。ジョブアレイや予約に関しては設定できません。-l select の外部に限り要求できます。消費不可能。デフォルト：なし。タイプ：duration。Python タイプ：pbs.duration

### 6.3.8 Shrink-to-fit ジョブの注意事項と制限事項

min_walltime を指定せずにジョブの max_walltime を指定するとエラーになります。`qsub` または `qalter` を使用した場合、次のエラーが出力されます。

```
'Can not have "max_walltime" without "min_walltime"'
```

max_walltime より大きい min_walltime を指定するとエラーになります。`qsub` または `qalter` を使用した場合、次のエラーが出力されます。

```
'"min_walltime" can not be greater than "max_walltime"'
```

ジョブアレイは shrink-to-fit にできません。Shrink-to-fit ジョブアレイは作成できません。ジョブアレイの min_walltime または max_walltime を指定するとエラーになります。`qsub` または `qalter` を使用した場合、次のエラーが出力されます。

```
'"min_walltime" and "max_walltime" are not valid resources for a job array'
```

予約は shrink-to-fit にできません。Shrink-to-fit 予約は作成できません。予約の min_walltime または max_walltime を設定するとエラーになります。`pbs_rsub` を使用した場合、次のエラーが出力されます。

```
'"min_walltime" and "max_walltime" are not valid resources for reservation.'
```

min_walltime および max_walltime の resources_max または resources_min を設定するとエラーになります。設定しようとすると、次のどちらかのエラーメッセージが表示されます。

```
"Resource limits can not be set for min_walltime"
"Resource limits can not be set for max_walltime"
```

## 6.4 チェックポイント処理の使用

### 6.4.1 チェックポイント処理の前提条件

ジョブは、チェックポイント不可としてマークされておらず、次のいずれかに該当する場合、チェックポイント可能です。

- アプリケーションでチェックポイントをサポートし、管理者がチェックポイントスクリプトを設定している
- サードパーティ製のチェックポイントアプリケーションが利用できる
- OS でチェックポイントをサポートしている

## 6.4.2 最小チェックポイント間隔

ジョブが格納されている実行キューでは、ジョブにチェックポイントを設定できる最小間隔を制御します。この間隔は、CPU時間（分単位）またはwalltime（分単位）で指定します。両方に同じ値が使用されるため、たとえば、最小間隔に12を指定すると、キューの間隔にCPU時間を使用しているジョブでは、CPU時間の12分ごとにチェックポイントが設定され、キューの間隔にwalltimeを使用しているジョブでは、walltime時間の12分ごとにチェックポイントが設定されます。

## 6.4.3 チェックポイント間隔を指定する構文

qsubの"-c checkpoint-spec"オプションを使用して、ジョブにチェックポイントを設定する間隔をCPU時間（分単位）またはwalltime（分単位）で指定します。

*checkpoint-spec*引数には以下の値を指定できます。

c

ジョブは、CPU時間で測定された間隔に従ってチェックポイント処理され、ジョブが格納されている実行キュー上で設定されます。

c=<CPU時間（分単位）>

ジョブは、ジョブで使用するCPU時間（分単位）の指定した間隔でチェックポイント処理されます。この値はゼロより大きくなければなりません。指定した間隔が、ジョブが格納されている実行キューの設定値よりも小さい場合は、キューの間隔が使用されます。

フォーマット：*整数*

w

ジョブは、walltimeで測定された間隔に従ってチェックポイント処理され、ジョブが格納されている実行キュー上で設定されます。

w=<walltime（分単位）>

チェックポイント処理は、ジョブで使用するwalltime（分単位）の指定した間隔で実行されます。この値はゼロより大きくなければなりません。指定した間隔が、ジョブが格納されている実行キューの設定値よりも小さい場合は、キューの間隔が使用されます。

フォーマット：*整数*

n

ジョブはチェックポイント処理されません。

s

ジョブはPBSサーバーのシャットダウン時にのみチェックポイントされます。

u

チェックポイント処理を指定しません。デフォルトでは、"s"と同じ動作になります。

ジョブのチェックポイント間隔は、Checkpointジョブ属性で制御します。これは、コマンドラインでqsubを使用して、またはPBS指示文で設定できます。

qsubを使用して、ジョブが実行キューのチェックポイント間隔を使用する必要があることを指定します。

```
qsub -c c my_job
```

指示文を使用して、CPU時間の１０分ごとにジョブをチェックポイント処理します。

```
#PBS -c c=10
```

### 6.4.4 チェックポイント処理によるジョブのプリエンプションまたは保留

サイトで実行中のジョブをプリエンプトする必要がある場合、または実行中のジョブを保留する必要がある場合があります。どちらかを許可するには、ジョブをチェックポイント可能にします。これは、ジョブをチェックポイント不可としてマークしてはいけない（`qsub -c n`を使用しない）こと、ユーザーのアプリケーションがチェックポイント可能またはサードパーティ製チェックポイント処理アプリケーションが存在すること、およびジョブが実行されるMoMが実行するチェックポイント処理スクリプトを管理者が用意する必要があることを意味します。

ジョブをプリエンプトまたは保留して結果の一部を保存する場合、アプリケーションレベルのチェックポイント処理を使用できます。チェックポイント処理されたジョブが再起動された場合、ジョブスクリプトでジョブがチェックポイント処理されたことを検出でき、最初から始めるのではなくチェックポイント処理されたファイルから開始できます。

チェックポイント可能ではない（チェックポイント不可としてマークされているか、スクリプトが存在しないか、異常を返す場合）実行中のジョブを保留しようとすると、ジョブは、Hold_Types属性が*h*に設定されて、実行を継続します。122ページの第6.5節「ジョブの保留/保留解除」をご参照ください。

### 6.4.5 チェックポイント処理の注意事項と制限事項

- チェックポイント処理は、ジョブアレイについてはサポートされていません。
- `qsub -c checkpoint-spec`を指定しない場合、チェックポイント処理は未指定であり、デフォルトで“s”と同じ動作になります。
- PBSは、ジョブの実行試行回数を21回に制限して、ジョブのrun_count属性でこの回数を追跡します。ジョブがチェックポイント処理および再キューイングされた回数が制限を超えると、ジョブは保留されます。

## 6.5 ジョブの保留/保留解除

ジョブを保留して、以下を実行できます。

- キューイングされているジョブは、ユーザーが保留を解除するまで、キューイングされたままです。123ページの第6.5.3節「ジョブを実行する前の保留」をご参照ください。
- 実行中のジョブは実行を停止しますが、停止したところから再開できます。123ページの第6.5.4.1節「ジョブのチェックポイント処理および再キューイング」をご参照ください。
- 実行中のジョブは実行を継続しますが、再キューイングされる場合は保留されます。124ページの第6.5.4.2節「実行中のジョブの保留タイプの設定」をご参照ください。

ジョブはqholdコマンドを使用して保留します。『PBS Professional Reference Guide』の「qhold」（150ページ）をご参照ください。

ユーザーは、キューイングされているジョブの保留を解除して、実行をスケジュール可能にすることができます。また、実行中のジョブの保留を解除できます。ジョブの保留はqrlsコマンドを使用して解除します。『PBS Professional Reference Guide』の「qrls」（183ページ）をご参照ください。

qholdコマンドの構文は、以下のとおりです。

*qhold [ -h <保留リスト> ] <ジョブID> [<ジョブID> ...]*

*qrlsコマンドの構文は、以下のとおりです。*

*qrls [ -h <保留リスト> ] <ジョブID> [<ジョブID> ...]*

ジョブアレイでは、*ジョブID*を二重引用符で囲む必要があります。

### 6.5.1 保留のタイプ

*保留リスト*は、ジョブにかける保留のタイプを指定します。*保留リスト*引数は、u、p、o、sの１つまたは複数を組み合わせた文字列、あるいは文字nです。以下の表は、各文字に関連付けられている保留のタイプを示します。

**表6-1：保留のタイプ**

| 保留のタイプ | 意味 | 設定または解除が可能なロール |
|---|---|---|
| *u* | *ユーザー* | *ジョブ所有者、オペレータ、マネージャ、管理者、ルート* |
| *o* | *その他* | *オペレータ、マネージャ、管理者、ルート* |
| *s* | *システム* | *マネージャ、管理者、ルート、PBS（依存関係）* |
| *n* | *保留なし* | *ジョブ所有者、オペレータ、マネージャ、管理者、ルート* |
| *p* | *不正パスワード* | *管理者、ルート* |

-hオプションが指定されていない場合は、PBSは、*ジョブID*リストで示されたジョブに対して*ユーザー*保留をかけます。

*ジョブID*リストで指定されたジョブがキュー待機、保留、または待機状態にある場合は、ジョブの他の保留への保留のタイプの追加のみが行われます。ジョブが実行キューでキューイングされているか、または待機している場合は、保留状態にもなります。

### 6.5.2 ジョブの保留または保留解除の要件

`qhold`コマンドまたは`qrls`コマンドを実行するユーザーは、保留を適用または解除するために必要な権限を持っている必要があります。保留の解除と保留の設定には、同じ規則が適用されます。

### 6.5.3 ジョブを実行する前の保留

通常、PBSは、適切なスロットがオープンするとすぐにジョブを実行します。ただし、ユーザーがPBSに、ジョブは実行不可であり、キューイングしたままにする必要があることを通知することができます。`qsub`の"`-h`"オプションを使用して、ジョブを投入するときに、ジョブに*ユーザー保留*をかけます。PBSは、ジョブを受け入れて*保留*状態にします。保留を解除しない限り、そのジョブは保留されたままであり、実行できません。

Hold_Typesジョブ属性は、ジョブの保留の動作を制御し、以下のように`qsub`または指示文で設定します。

```
qsub -h my_job
#PBS -h
```

### 6.5.4 実行中のジョブの保留

#### 6.5.4.1 ジョブのチェックポイント処理および再キューイング

ジョブがチェックポイント可能な場合、ジョブを保留することによってその実行を停止できます。この場合、以下の処理が行われます：

- ジョブがチェックポイント処理されます。
- このジョブに割り当てられているリソースが解放されます。
- ジョブが保留状態で実行キューに戻されます。

120ページの第6.4.1節「チェックポイント処理の前提条件」をご参照ください。

ジョブを保留するには、qholdコマンドを使用します。

```
qsub -h my_job
```

### 6.5.4.2 実行中のジョブの保留タイプの設定

ジョブにチェックポイントを設定できない場合、qholdは単にジョブのHold_Types属性の設定のみ行います。これは、ジョブをqrerunコマンドで再キューイングしていなければ効果ありません。その場合、ユーザーが保留を解除しない限り、そのジョブはキューイングされたままであり、実行できません。

## 6.5.5 ジョブの保留解除

qrlsコマンドを使用して、ジョブの１つまたは複数の保留を解除できます。

qrlsコマンドの構文は、以下のとおりです。

*qrls [ -h <保留リスト> ] <ジョブID> ...*

ジョブアレイでは、*ジョブID*を二重引用符で囲む必要があります。

保留状態でないジョブに対して保留を解除しようとした場合、qrlsコマンドは無視されます。qrlsコマンドを使用して、すでに実行されていて、チェックポイントが設定されたジョブの保留を解除する場合、保留は解除され、ジョブはキュー待機（Q）状態に戻ります。ジョブは、リソースが使用可能になると、実行するようスケジュールされる状態になります。

qrlsコマンドではジョブは実行されません。ジョブの保留状態を解除して、スケジューラによって次回選択された際に実行可能な状態にするだけです。

## 6.5.6 ジョブの保留と保留解除の注意事項と制限事項

- qholdコマンドはジョブアレイに使用することができますが、サブジョブまたはサブジョブの範囲には使用できません。ジョブアレイについては、qholdコマンドは‘Q’、‘B’、または‘W’状態の場合にのみ適用できます。これにより、ジョブアレイは‘H’（保留）状態に置かれます。サブジョブが実行中である場合、それらは完了するまで実行を継続します。実行中のサブジョブが存在する場合、ジョブアレイを‘H’状態に移行することはできません。
- チェックポイント処理は、ジョブアレイについてはサポートされていません。チェックポイント処理をサポートしているシステム上であっても、サブジョブにはチェックポイントが設定されず、完了するまで実行が継続されます。
- 実行中のジョブを保留してその実行を停止するには、ジョブがチェックポイント可能である必要があります。120ページの第6.4.1節「チェックポイント処理の前提条件」をご参照ください。
- qrlsコマンドはジョブアレイオブジェクトのみに使用でき、サブジョブまたは範囲には使用できません。ジョブアレイは、‘Q’、‘B’、または‘W’のいずれかである保留前の状態に戻されます。
- qholdコマンドはジョブアレイオブジェクトのみに使用でき、サブジョブまたは範囲には使用できません。保留は‘Q’、‘B’、または‘W’状態からのみ、ジョブアレイに適用することが可能です。これにより、ジョブアレイは‘H’（保留）状態に置かれます。サブジョブが実行中である場合、それらは完了するまで実行を継続します。‘H’（保留）状態にあるキュー内のサブジョブは開始されません。
- PBSは、ジョブの実行試行回数を21回に制限して、ジョブのrun_count属性でこの回数を追跡します。ジョブがチェックポイント処理および再キューイングされた回数が制限を超えると、ジョブは保留されます。

## 6.5.7 ジョブが保留される理由

ジョブは、以下のいずれかの理由で保留される可能性があります。

- 無効なプロビジョニング要求または内部システムエラーが原因でプロビジョニングが失敗する（“s”）
- プロビジョニング後、vnodeによってレポートされたAOEが、ジョブの要求したAOEと一致しない（“s”）
- ジョブがPBSマネージャまたはオペレータにより保留された（“o”）
- ジョブがチェックポイントされ、再キューイングされた（“s”）
- ジョブが、PBSが履歴を保持している終了したジョブに依存している（“s”）
- ジョブのパスワードが無効である（“p”）
- ジョブのrun_count属性の値が20より大きい

## 6.5.8 ジョブの保留および保留解除の例

次の例は、qholdコマンドおよびqrlsコマンドの両方を使用する方法を示しています。状態（“S”）の列は、これらの2つのコマンドによって、ジョブの状態がどのように変化するかを示しています。

```
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --

qhold 54
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 H  --




qrls -h u 54
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --
```

# 6.6 ジョブの再実行の許可

ジョブが完了前に何らかの理由で終了している場合にジョブが再実行可能かどうかを指定できます。qsubの“-r”オプションを使用して、ジョブが再実行可能かどうかを指定します。このオプションの引数は、“*y*”（再実行可能）または“*n*”（再実行不可能）です。ジョブが再実行可能かどうかを指定しない場合は、再実行可能になります。

ジョブを複数回実行すると問題が発生する場合、ジョブを再実行不可能としてマークします。それ以外の場合は、再実行可能にします。ジョブを再実行不可能に設定する目的は、ジョブを複数回開始するのを回避することにあります。

再実行不可能に設定されたジョブで、実行が開始される前の起動時にエラーが発生した場合、ジョブの再試行が再キューイングされます。

Rerunnableジョブ属性は、ジョブが再実行可能かどうかを制御し、以下のようにqsubまたは指示文で設定できます。

```
qsub -r n my_job
#PBS -r n
```

以下の表に、ジョブのRerunnable属性によって違いが生じる状況と生じない状況を示します。

**表6-2：Rerunnable属性の影響を受ける状況**

| 状況 | Rerunnable | 再実行不可能 |
|---|---|---|
| ジョブが実行前の起動時に失敗した | ジョブは再キューイングされる | ジョブは再キューイングされる |
| ジョブが複数のホストで実行中に、1つのホストが停止した | ジョブは再キューイングされる | ジョブは削除される |
| ジョブが複数のホストで実行をスケジュールされているが、少なくとも1つのホストで開始しなかった | ジョブは再キューイングされる | ジョブは再キューイングされる |
| サーバーが少し時間をおいてシャットダウンされた | ジョブは再キューイングされる | ジョブは終了する |
| サーバーが時間をおかずにシャットダウンされた | ジョブは再キューイングされる | ジョブは削除される |
| ジョブがプロビジョニングを要求したが、プロビジョニングスクリプトが失敗した | ジョブは再キューイングされる | ジョブは再キューイングされる |
| ジョブが複数のホストで実行中に、1つのホストがコンソールのアクティビティによりビジーになった | ジョブは再キューイングされる | ジョブは削除される |
| より優先度の高いジョブがリソースを要求している | ジョブは再キューイングされる可能性がある | ジョブは削除される可能性がある |

## 6.6.1 ジョブを再実行可能とマークする場合の注意事項と制限事項

- インタラクティブジョブは再実行されません。
- ジョブアレイは再実行可能にする必要があります。PBSでは、再実行不可能とマーキングされているジョブアレイは受け付けません。再実行可能かどうかを指定せずにジョブアレイを投入することはできますが、その場合は、PBSによって自動的に再実行可能とマーキングされます。
- 自身のジョブを再実行不可能としてマークするのは、複数回実行することによって問題が発生する場合のみです。自身のジョブが再実行不可能としてマークされ、優先度の高いジョブがリソースを要求している場合、自身のジョブは削除される場合があります。

# 6.7 ジョブの再実行回数の制御

PBSでは、ジョブの実行を試行する回数に対する組み込みのリミットが、21に設定されています。試行回数は、ジョブのrun_count属性で追跡されます。デフォルトでは、ジョブ投入時のrun_countの値は0です。run_countの値が20を超えると、ジョブは保留されます。

ユーザーは、PBSによるジョブの試行回数を減らすことができます。ジョブ投入の際、run_countに対して負でない値を指定することができます。ジョブの実行中は、`qalter`を使用してrun_countの値を増やすことができます。このリミットを超えてジョブを再試行することはできません。また、ジョブの実行中にrun_countの値を減らすことはできません。

## 6.7.1 run_count属性の値を増やす場合の注意事項

ジョブがチェックポイント処理および再キューイングされた回数が制限を超えると、ジョブは保留されます。

# 6.8 実行の延期

通常、PBSは、適切なスロットがオープンするとすぐにジョブを実行します。そうしないで、ジョブが実行対象となる時間を指定することができます。投入されてから実行可能になるまで、ジョブは待機（W）状態になります。

## 6.8.1 実行の延期の構文

`qsub`の"`-a` <日時>"オプションを使用して、ジョブの実行に適した開始時刻を指定します。*日時*引数の形式は以下のとおりです。

*[[[[CC]YY]MM]DD]hhmm[.SS]*

パラメータの意味

*CC*は年の最初の２桁（世紀）：オプション

*YY*は年の最後の２桁：オプション

*MM*は月を表す２桁の数値：オプション

*DD*は日付：オプション

*hh*は時間

*mm*は分

*SS*は秒：オプション

日（*DD*）が未来であり、月（*MM*）を指定しない場合、月はデフォルトで現在の月が適用されます。日（*DD*）が過去であり、月（*MM*）を指定しない場合、月は次の月に設定されます。たとえば、今日が10日で、月を指定しないで12日を指定した場合、ジョブは今日から２日後の12日に実行可能になります。

同様に、時刻（*hhmm*）が未来であり、日（*DD*）を指定しない場合、日はデフォルトで当日が適用されます。時刻（*hhmm*）が過去であり、日（*DD*）を指定しない場合、日は明日に設定されます。たとえば、"`1110`"という時間を指定して午前11時15分にジョブを投入した場合、そのジョブは、翌日の午前11時10分に実行可能になります。

実行の延期は、ジョブのExecution _Time属性で制御します。これは、以下のどちらかを使用して設定できます。

```
qsub -a 0700 my_job
#PBS -a 10220700
```

# 6.9 ジョブの優先度の設定

PBSでは、ジョブごとにその優先度を指定できる場所があります。管理者がジョブをスケジューリングする際にこの優先度を使用するかどうかはわかりません。“-p <priority>”を使用して、ジョブの優先度を指定します。*priority*引数は、-1024（最下位優先）から+1023（最上位優先）までの整数でなければなりません。デフォルトは未設定で、これは0と同じです。

指定する値は、Priorityジョブ属性に格納されます。以下のように、qsubまたは指示文で設定します。

```
qsub -p 120 my_job
#PBS -p -300
```

ユーザーがそのジョブの絶対的な順序を指定する必要がある場合は、113ページの第6.2節「ジョブの依存関係の使用」をご参照ください。

# 6.10 ジョブの終了までqsubを待機させる

通常、ジョブを投入する際、qsubコマンドは新しいジョブのIDを返した後に終了します。qsubの“-W block=true”オプションを使用すると、qsubに“ブロック”するように、すなわちジョブが完了し、ジョブの終了値がレポートされるまで待機するように、指定できます。

ジョブが正常に投入された場合は、qsubは、ジョブが終了するかエラーが発生するまでブロックします。ジョブの投入に失敗した場合は、特別な処理は行われません。

ジョブが最後まで実行された場合は、qsubはジョブの終了ステータスで終了します。ジョブアレイについては、qsubのブロックはジョブアレイ全体が完了するまで待機したうえで、ジョブアレイの終了ステータスを戻します。

ブロッキングは、blockジョブ属性で制御します。以下のように、qsubまたはPBS指示文のどちらかで設定します。

```
qsub -W block=true
#PBS -W block=true
```

## 6.10.1 ジョブのブロッキングのシグナル処理とエラー処理

シグナルのSIGQUITとSIGKILLはトラップされないので、ただちにqsubプロセスを終了します。関連付けられていたジョブは実行中またはキューイングされたままになります。

qsubは、SIGHUP、SIGINT、またはSIGTERMのいずれかのシグナルを受け取ると、メッセージを出力し、終了ステータス*2*で終了します。

実行して終了する前にジョブが削除された場合、または、内部PBSエラーが発生した場合は、qsubは状況を説明するエラーメッセージをこのエラーストリームに出力し、終了ステータス*3*で終了します。

## 6.10.2 ジョブのブロッキングの注意事項

- ブロッキングとインタラクティブの両方の特性を持つジョブを投入した場合、ジョブの終了時にジョブの終了ステータスは返されません。
- ブロッキングジョブの場合、そのステージングが完了する前に、PBSがジョブの終了ステータスを返します。ジョブがまだステージング中かどうかを確認するには、qstat -fを使用して、ジョブのsubstate属性を確認します。ファイルがステージアウトされているときは、この属性の値は*51*です。

# 6.11 ジョブのインタラクティブな実行

PBSでは、*インタラクティブバッチジョブ*または*インタラクティブジョブ*と呼ばれる特殊なバッチジョブが使用できます。インタラクティブジョブは、通常のバッチジョブと同様に処理されます。つまり、キューに格納され、そのジョブを実行するためのリソースが使用可能になるまで待機する必要があります。ただし、ジョブを開始した後は、ユーザー端末の入力と出力が、ログインセッションと同じようにジョブに接続されます。ユーザーが使用可能ないずれかの実行マシンにログインすると、そのジョブに必要なリソースが予約されます。これは、アプリケーションのデバッグや処理の制御に役立ちます。

リモートホストのインタラクティブジョブでGUIアプリケーションを使用できます。PBSインターフェースは、LinuxとWindowsで若干異なります。Linuxの場合は131ページの第6.11.9節「インタラクティブLinuxジョブからのX出力の受け取り」をご参照ください。Windowsの場合は133ページの第6.11.10節「Windowsでのインタラクティブ GUIジョブの投入」をご参照ください。

インタラクティブジョブではプロビジョニングを使用できます。

## 6.11.1 インタラクティブジョブの入出力

インタラクティブジョブには、端末制御情報の設定コマンドを実行するのに適した擬似環境が完備しています。インタラクティブジョブの実行を開始した後は、ジョブへの入力とジョブからの出力は`qsub`を経由します。インタラクティブジョブへの入力はすべて、ジョブを実行している端末セッションから行います。

インタラクティブジョブの場合、ジョブスクリプト内でPBS指示文を指定できます。ジョブスクリプトを使用してジョブにコマンドを渡すことはできません。インタラクティブジョブの場合、PBSはジョブスクリプト内の実行可能コマンドを無視します。

## 6.11.2 インタラクティブジョブの実行

インタラクティブにジョブを実行するには、以下のどちらの方法も使用できます。

- コマンドラインで`qsub -I`を使用
- PBS指示文で`#PBS interactive=true`（**非推奨**）を使用

インタラクティブジョブの実行中に、ユーザーはコマンド、実行可能ファイル、シェルスクリプト、DOSコマンドなどを実行できます。これらのコマンドは普通に動作します。たとえば、コマンドへのパスがユーザーの環境のPATH環境変数に存在しない場合は、フルパスを指定する必要があります。

## 6.11.3 インタラクティブジョブのライフサイクル

1. インタラクティブジョブは、`qsub #PBS interactive=true`（**非推奨**）または`-I`を使用して開始します。
2. スクリプトが存在する場合、PBSはスクリプト内に指示文があればそれを処理します。
3. スケジューラがジョブを実行します。
4. 出力が投入ウィンドウに接続されます。
5. コマンド、実行可能ファイル、シェルスクリプトなどをインタラクティブに実行します。
6. ジョブは終了します。

### 6.11.3.1 インタラクティブジョブの終了

インタラクティブジョブを実行する場合、qsubコマンドはジョブを投入しても終了しません。以下のいずれかが実行されるまで、qsubは実行中のままです：

- ユーザーがジョブに対してqdelを実行する
- ユーザー以外の誰かがジョブを削除する
- ユーザーがシェルを終了する
- ジョブが中止される
- スケジューラがジョブを開始する前に、ユーザーがSIGINT(Ctrlキーを押しながらCを押す)によってqsubを中断する

スケジューラがジョブを開始した後は、SIGINTは無視されます。

Linuxでは、ジョブの開始前にユーザーがqsubを中断した場合、qsubが終了するかどうかをユーザーに確認します。ユーザーが"*yes*"と応答すると、qsubが終了し、ジョブが中止されます。Windowsでは、ジョブの開始前にジョブを中断すると、ジョブが削除され、以下のメッセージが出力されます。

```
qsub: wait for job <ジョブID> interrupted by signal 2
<ジョブID> is being deleted
```

## 6.11.4 インタラクティブジョブと終了コード

Windowsでは、ユーザーがインタラクティブセッションを終了する際に"exit <終了コード>"を使用して終了コードを指定した場合、その終了コードがジョブの終了コードとして使用されます。この終了コードは、tracejobコマンドの出力で表示されます。

Linuxでは、ユーザーはインタラクティブセッションの終了コードを指定できません。

## 6.11.5 インタラクティブジョブの進行状況の追跡

ユーザーがインタラクティブジョブを投入した後、そのウィンドウに以下のメッセージが出力されます。

```
qsub: waiting for job <ジョブID> to start
```

スケジューラによってジョブが開始されると、投入ウィンドウに以下のメッセージが出力されます。

```
qsub: job <ジョブID> ready
```

インタラクティブジョブが終了すると、投入ウィンドウに以下のメッセージが出力されます。

```
qsub: job <ジョブID> completed
```

## 6.11.6 インタラクティブジョブの特殊シーケンス

キーボードからの割り込みはジョブに渡されます。チルダ記号（~）で始まり、特殊文字を含む行は、qsub自体によって解釈されます。認識される特殊シーケンスは次のとおりです。

~.

qsubの実行を終了します。バッチジョブも終了します。

~susp

qsubプログラムを一時停止します。"*susp*"は一時停止文字で、通常はCtrl + Zキーです。

~asusp

qsubの入力（ジョブの端末）が抑制されます。ただし、出力は引き続き表示されます。“*asusp*”は補助の一時停止文字で、通常はCtrl + Yキーです。

## 6.11.7 インタラクティブジョブの注意事項と制限事項

- ログインファイルがバックグラウンドでプロセスを実行しないようにしてください。6ページの第1.4.2.5節「ジョブ内のバックグラウンドプロセスの回避」をご参照ください。
- アレイジョブはインタラクティブに実行することはできません。
- インタラクティブジョブは再実行されません。
- インタラクティブジョブではCLSコマンドを使用できません。これでは画面はクリアされません。
- スケジューラがインタラクティブジョブを開始した後は、SIGINT（Ctrl + Cキー）は無視されます。
- Linuxでは、ユーザーはインタラクティブセッションの終了コードを指定できません。
- インタラクティブジョブの終了時に、ステージングされたファイルおよび*stdout*や*stderr*がまだコピーされていない可能性があります。
- 投入ホストは、着信エフェメラルポートを受け入れる必要があります。

## 6.11.8 エラーとロギング

- PBS がリモートインタラクティブシェルを開いてインタラクティブジョブを実行することができない場合、PBS では次のエラーメッセージが出力されます。
  “qsub: failed to run remote interactive shell”
- リモートホストのIPC$に接続できない場合、PBSでは次のエラーメッセージが出力されます。
  “Couldn't connect to host <ホスト名>”
- PBSが実行ホストでIPC$への接続には成功したが、リモートシェルの実行には失敗した場合、PBSでは次のエラーメッセージが出力されます。
  “Couldn't execute remote shell at host <ホスト名>”

## 6.11.9 インタラクティブLinuxジョブからのX出力の受け取り

Linuxでは、qsub -Xオプションにより、インタラクティブジョブからX出力を受け取ることができます。

### 6.11.9.1 LinuxでのX出力の受け取り方法

X出力を受け取るには、qsub -X -Iを使用します。以下に例を示します。

```
qsub -I -X <return>
xterm <return>
```

Xプロセスが終了すると、制御はここに戻ります。必要に応じて、ここからプロセスをバックグラウンドで実行できます。

#### 6.11.9.1.i 非投入ホストでのX出力の受け取り

ジョブ投入ホスト以外のホストでX出力を表示できます。たとえば、SubHostからジョブを投入して、ViewHostで出力を確認するとします。ViewHostなど、投入ホスト以外のホストでX出力を受け取る場合、以下を実行します。

- ViewHost上でXサーバーを実行する
- ViewHost上で`ssh -X`を使用してSubHostにログインする
- SubHostにログインしたウィンドウで`qsub -I -X`を実行する

### 6.11.9.2 X出力を受け取るための要件

- Linuxを実行している必要があります。
- ジョブはインタラクティブでなければならず、`-I`を指定する必要もあります。
- Xサーバーは、X出力を表示したいシステム上で実行している必要があります。
- ジョブ投入環境のDISPLAY変数には、X出力の表示が求められているディスプレイを設定する必要があります。
- 管理者は、MoMのPATHを設定して、`xauth`ユーティリティを含める必要があります。

### 6.11.9.3 X出力のジョブ属性の表示

各ジョブには、X転送情報を含む読み込み専用の属性が２つあります。それらの属性は次のとおりです。

forward_x11_cookie
: この属性には、X権限のクッキーが含まれています。

forward_x11_port
: この属性には、投入ホスト上でポートフォワーダーが受信待機するポートの番号が含まれています。

これらの属性は、`qstat -f <ジョブID>`で表示できます。

### 6.11.9.4 X出力の受け取りに関する注意事項とアドバイス

- このオプションはWindows環境では使用できません。
- `qsub -V`オプションを使用する場合、PBSがDISPLAY変数を適切に処理します。
- `qsub -v DISPLAY`オプションを使用すると、エラーが発生します。
- 同一のジョブセッションを使用して同時に実行できるXアプリケーションは最大で25個です。
- `qsub -X -I`を使用してXで問題が発生した場合、以下を使用して、`qsub`がXセッションを確立する際に使用する適切な`~/.Xauthority`ファイルを作成します。
  `ssh -X <ホスト名> server <-> <実行ホスト>`

### 6.11.9.5 X転送のエラー

- DISPLAY 環境変数が、適切にフォーマットされているものの正しくないディスプレイ番号を指している場合、インタラクティブなX転送ジョブを投入すると次のようなエラーメッセージが表示されます。
  "cannot read data from 'xauth list <display number>', errno=<errno>"
- DISPLAY 環境変数が適切にフォーマットされていない値を指している場合、インタラクティブな X 転送ジョブを投入すると次のようなエラーメッセージが表示されます。
  "qsub: Failed to get xauth data (check $DISPLAY variable)"
- X権限ユーティリティ（xauth）を投入ホストで検出できない場合、次のエラーメッセージが表示されます。
  "execution of xauth failed: sh: xauth: command not found"
- xauthユーティリティの実行時にエラーが発生する場合、xauthユーティリティによって表示されるエラーメッセージの先頭には次の文言が表示されます。
  "execution of xauth failed: "
- -Iを指定しないでqsub -Xオプションを使用する場合、次のエラーメッセージが表示されます。
  "qsub: X11 forwarding possible only for Interactive Jobs"

## 6.11.10 WindowsでのインタラクティブGUIジョブの投入

GUIアプリケーションを使用するインタラクティブジョブを実行できます。ジョブを投入するホストとは別のホストでジョブを実行する場合、PBSはリモートビューアまたはインタラクティブシェルを使用してGUIアプリケーションをリモートホストに接続します。Windowsでは、Remote ViewerやXを含む任意のGUIアプリケーションがサポートされます。ジョブでGUIアプリケーションまたはインタラクティブシェルが必要な場合は、そのジョブをインタラクティブジョブとして実行する必要があります。

GUIアプリケーションを起動するインタラクティブPBSジョブを実行するには、以下のコマンドを実行します。

**qsub -I -G -- <GUIアプリケーション>**

投入と実行で同じホストが使用される場合、アプリケーションはそのローカルコンソールで起動されます。リモートビューアクライアントは起動されません。

投入ホストと実行ホストが異なる場合は、指定したリモートビューアを使用して、リモートセッションでGUIアプリケーションが起動されます。リモートビューアクライアントが起動されます。

WindowsでXを実行する場合は、-Xオプションを使用しないでください。このオプションはWindows環境では使用できません。-Gを使用します。

PBSジョブでインタラクティブシェルを起動するには、次のコマンドを実行します。

**qsub -I -G**

投入と実行で同じホストが使用される場合、そのローカルコンソールでインタラクティブシェルが起動されます。リモートビューアクライアントは起動されません。

投入ホストと実行ホストが異なる場合は、インタラクティブシェルが起動され、このシェルを介して起動された任意のGUIアプリケーションが、設定されたリモートビューアを使用してリモートセッションで表示されます。リモートビューアクライアントが起動されます。

以下の状況では、インタラクティブGUIジョブは終了され、実行されなくなります。

- qsub -I -G <GUIアプリケーション>によって起動されたGUIアプリケーションが閉じられた場合
- qsub -I -Gによって起動されたインタラクティブシェルが終了した場合
- リモートビューアが終了した、閉じられた、またはログオフされた場合。そのリモートビューアによって起動されたすべてのアプリケーションは閉じられます。
- qdelによってGUIジョブが削除された場合。そのジョブに関連付けられたすべてのアプリケーションおよびタスクは強制終了されます。

『PBS Professional Reference Guide』の「-G [<path to GUI application or script>]」（223ページ）をご参照ください。

# 6.12 環境変数の使用

PBSはジョブに環境変数を提供し、ジョブはそれらに基づいて動作します。環境変数には、ユーザーの投入環境から取得するものと、PBSによって作成されるものがあります。ユーザーはそのジョブに合わせて環境変数を作成できます。PBSによって作成される環境変数の名前は、“*PBS_*”で始まります。PBSがユーザーの投入（発生元）環境から取得する環境変数の名前は、“*PBS_O_*”で始まります。

たとえば、ジョブに付随する環境変数の一部とその一般的な値を以下に示します。

```
PBS_O_HOME=/u/user1
PBS_O_LOGNAME=user1
PBS_O_PATH=/usr/bin:/usr/local/bin:/bin
PBS_O_SHELL=/bin/tcsh
PBS_O_HOST=host1
PBS_O_WORKDIR=/u/user1
PBS_JOBID=16386.server1
```

PBS環境変数の完全なリストについては、『PBS Professional Reference Guide』の「PBS Environment Variables」（397ページ）をご参照ください。

## 6.12.1 すべての環境変数のエクスポート

“-V”オプションでは、qsubコマンドの環境内にあるすべての環境変数をバッチジョブにエクスポートすることを宣言します。

```
qsub -V my_job
#PBS -V
```

## 6.12.2 特定の環境変数のエクスポート

qsubの“-v <変数リスト>”オプションでは、ジョブにエクスポートする追加の環境変数を指定できます。*変数リスト*には、ジョブの実行時に使用可能にするqsubコマンド環境の環境変数を指定します。これらの変数とその値がジョブに渡されます。これらの変数は、すでに自動的にエクスポートされている変数に追加されます。フォーマット：コンマで区切られた文字列のリストで、次の形式で指定します。

*-v <変数>*

または

*-v <変数>=<値>*

<*変数*>=<*値*>の組み合わせにコンマが含まれている場合は、値を一重または二重引用符で囲み、<*変数*>=<*値*>全体は、値を囲むのに使用した引用符とは別の種類の引用符で囲む必要があります。以下に例を示します。

```
qsub -v DISPLAY,myvariable=32 my_job
qsub -v "var1='A,B,C,D'" job.sh
qsub -v a=10, "var2='A,B'", c=20, HOME=/home/zzz job.sh
```

### 6.12.3 環境変数とシェル関数の注意事項

転送しようとするエクスポートされたシェル関数と同じ名前の環境変数が存在しないことを確認します。環境変数と同じ名前のシェル関数は環境内で表示できません。

### 6.12.4 エクスポートされたシェル関数の転送

エクスポートされたシェル関数は、`qsub -V`または`qsub -v` <関数名>のどちらかを使用して転送できます。また、これらの関数を実行ホスト（複数可）の`.profile`または`.login`に追加することもできます。

`-V`または`-V`を使用する場合、転送しようとするエクスポートされたシェル関数と同じ名前の環境変数が存在しないことを確認します。存在する場合、そのようなシェル関数は環境内で表示できません。

## 6.13 プリエンプトするジョブの指定

ユーザーのジョブを実行するためにプリエンプトできるジョブのグループを指定できます。1つまたは複数のキューに存在するすべてのジョブを指定したり、preempt_targetsリソースにリストされている特定のリソースを要求するすべてのジョブを指定したりできます。

構文：

*...-l preempt_targets="queue=<キュー名>[,queue=<キュー名>],*
*Resource_List.<リソース>=<値>[,Resource_List.<リソース>=<値>]"*

たとえば、ユーザーのジョブが、QueueAという名前のキューに存在するジョブとarch=linuxを要求したジョブをプリエンプトできることを指定するには、以下のように指定します。

```
...-l preempt_targets="queue=QueueA,Resource_List.arch=linux"
```

ジョブがコンプレックス内の他のジョブをプリエンプトしないようにするには、そのジョブのpreemption_targetsを“None”（大文字と小文字は区別されません）というキーワードに設定します。

後続のリソース指定には、preempt_targetsリソース指定を持続させるか、別の-l指定を使用します。このようにしないと、後続のリソース指定は、PBSをpreempt_targetsへの追加のように見なします。

# 6.14 ジョブからの不要なvnodeの解放

不要なリソース使用を避けたい場合、(プライマリ実行ホストやそのvnodeではなく)不要なシスターホストまたはvnodeをジョブから解放できます。pbs_release_nodesコマンドまたはrelease_nodes_on_stageoutジョブ属性を使用できます。

- コマンドラインまたはジョブスクリプトでpbs_release_nodesコマンドを使用して、このコマンドの発行時にシスターホストまたはvnodeを解放することができます。このコマンドを使用して、プライマリ実行ホスト上にない特定のvnode、またはプライマリ実行ホスト上にないすべてのvnodeを解放できます。これを使用して、指定したもの(保持するホスト数、または保持するvnodeを表す選択指定のいずれか)を除くすべてのホストまたはvnodeを解放することもできます。プライマリ実行ホスト上のvnodeを解放するのにこのコマンドを使用することはできません。『PBS Professional Reference Guide』の「pbs_release_nodes」(92ページ)をご参照ください。
- ジョブのrelease_nodes_on_stageout属性を*True*に設定することで、ステージアウトの開始時に、PBSはプライマリ実行ホスト上にないそのジョブのvnodeをすべて解放できます。そのジョブのstageout属性も設定する必要があります。『PBS Professional Reference Guide』の「Job Attributes」(327ページ)をご参照ください。

## 6.14.1 vnodeの解放の注意事項と制限事項

- ステージアウト時にvnodeを解放するには、ステージアウトパラメータを指定する必要があります。ステージアウトを指定しない場合は、release_nodes_on_stageoutは無効になります。
- プライマリ実行ホスト上にないvnodeのみを解放できます。プライマリ実行ホスト上のvnodeを解放することはできません。
- ジョブが実行中(*R*状態)である必要があります。
- pbs_release_nodesコマンドは、Cray X*シリーズシステムに関連付けられたvnode(vntypeに“cray_”というプレフィックスがあるvnode)上ではサポートされていません。
- cgroupサポートが有効で、MoMによって管理される一部の(すべてではない)vnodeを解放するためにpbs_release_nodesが呼び出されると、これらのvnode上のリソースが解放されます。
- マルチvnodeホスト上のvnodeがジョブに排他的に割り当てられ、そのvnodeが解放された場合、そのジョブではそのvnodeが解放されたことが示されますが、pbsnodes -avでは、そのホスト上の他のvnodeが解放されるまで、そのvnodeはそのジョブにまだ割り当てられているものとして示されます。マルチvnodeマシン上のvnodeがジョブに排他的に割り当てられておらず、そのvnodeが解放された場合、そのホスト上の他のvnodeが解放されたかどうかにかかわらず、そのvnodeは解放されたものとして示されます。
- ジョブプロセスが実行されているvnodeの解放を指定した場合、そのvnodeが解放されるとそのプロセスは終了されます。

## 6.14.2 vnodeを解放した場合の動作

ジョブのvnodeを解放すると、以下のようになります。

- ジョブの$PBS_NODEFILEに解放したvnodeがリストされなくなります。
- サーバーは、ジョブがvnodeからクリーンアップされたことの確認を受け取るまで、引き続きジョブを保持します。
- vnodeは最後のアクションとしてプライマリ実行ホストMoMにそのジョブのresources_used*値をレポートします。解放されたvnodeはジョブの一部ではなくなるため、そのジョブのresources_used値を更新しません。しかし、プライマリ実行ホストはデータを保持しており、ジョブの終了時のresources_used値の最後の集計でそのデータを追加します。
- pbs_release_nodesが正常に呼び出されるたびに、qstatはジョブのexec_host、exec_vnode、およびResource_List属性の更新値を表示します。

vnodeを解放する際、同じMoMによって管理されるジョブに割り当てられたすべてのvnodeが解放されると、そのジョブはそのMoMのホストから完全に削除されます。これにより、次のようになります。

- execjob_epilogue hookスクリプト（存在する場合）が実行されます。
- そのホストのジョブプロセスは強制終了されます。
- ジョブの一時ディレクトリを含むジョブ固有のファイルは削除されます。

ジョブに割り当てられた実行ホストから、1つまたは複数（ただしすべてでなはい）のvnodeが解放されても、そのジョブはまだそのホストからは削除されません。解放されたこれらのvnodeが共有されるように設定されていた場合、これらは他のジョブに再度割り当てることができます。

## 6.14.3 ジョブからの不要なvnodeの解放例

例6-4： ステージアウト時、プライマリ以外の実行ホストvnodeを解放するジョブを投入：

```
% qsub -W stageout=my_stageout@executionhost2:my_stageout.out -W release_nodes_on_stageout=true
  job.scr
```

例6-5： ジョブから特定のvnodeを解放：

構文：*pbs_release_nodes [-j < ジョブID>] <vnode名> [<vnode名>] ...]*

```
% qsub job.scr
241.myserverhost
% qstat  241 | grep "exec|Resource_List|select"
exec_host = executionhost1[0]/0*0+executionhost2/0*0+executionhost3/0*2
exec_vnode =
    (executionhost1[0]:mem=1048576kb:ncpus=1+executionhost1[1]:mem=1048576kb:ncpus=1+executionhost1
    [2]:ncpus=1)+(executionhost2:mem=104
8576kb:ncpus=1+executionhost2[0]:mem=1048576k:ncpus=1+executionhost2[1]:ncpus=1)+(executionhost3:nc
    pus=2:mem=2097152kb)
Resource_List.mem = 6gb
Resource_List.ncpus = 8
Resource_List.nodect = 3
Resource_List.place = scatter
Resource_List.select = ncpus=3:mem=2gb+ncpus=3:mem=2gb+ncpus=2:mem=2gb
schedselect = 1:ncpus=3:mem=2gb+1:ncpus=3:mem=2gb+1:ncpus=2:mem=2gb
% pbs_release_nodes -j 241 executionhost2[1] executionhost3
% qstat  241 | grep "exec|Resource_List|select"
```

exec_host = executionhost1[0]/0*0+executionhost2/0*0 （executionhost3なし。executionhost3に割り当てられたすべてのvnodeは解放済み）

exec_vnode = (executionhost1[0]:mem=1048576kb:ncpus=1+executionhost1[1]:mem=1048576kb:ncpus=1+executionhost1[2]:ncpus=1)+(executionhost2:mem=1048576kb:ncpus=1+executionhost2[0]:mem=1048576kb:ncpus=1) （executionhost2[1]およびexecutionhost3は表示されない）

Resource_List.mem = 4194304kb （executionhost3から2gb削減）

Resource_List.ncpus = 5 （executionhost2[1]から1つ、executionhost3から2つの計3つのCPUを削減）

Resource_List.nodect = 2 （executionhost3の解放時に1チャンク削減、すべてのチャンク割り当てが消失）

Resource_List.place = scatter

schedselect = 1:mem=2097152kb:ncpus=3+1:mem=2097152kb:ncpus=2

例6-6： プライマリ実行ホストにないすべてのvnodeを解放：

構文：*pbs_release_nodes [-j <ジョブID>] -a*

```
% pbs_release_nodes -j 241 -a
% qstat -f 241
exec_host = executionhost1[0]/0*0
exec_vnode = (executionhost1[0]:mem=1048576kb:ncpus=1)+executionhost1[1]:mem=1048576kb:ncpus=1+executionhost1[2]:ncpus=1)
Resource_List.mem = 2097152kb
Resource_List.ncpus =  3
Resource_List.nodect = 1
Resource_List.place = scatter
schedselect = 1:mem=2097152kb:ncpus=3
```

例6-7： 4つ以外のすべてのシスターホストを解放：

```
% pbs_release_nodes -k 4
```

例6-8： “bigmem”としてマークされた8つを除くすべてのシスターvnodeを解放：

```
% pbs_release_nodes -k select=8:bigmem=true
```

例6-9： シスターvnodeの解放後、シスターvnodeは$PBS_NODEFILEにリストされなくなる：

```
% qsub -l select=2:ncpus=1:mem=1gb -l place=scatter -I
qsub: waiting for job 247.executionhost1.example.com to start
qsub: job 247.executionhost1.example.com ready
% cat $PBS_NODEFILE
executionhost1.example.com
executionhost2.example.com
% pbs_release_nodes -j 247 executionhost2
% cat $PBS_NODEFILE
executionhost1.example.com
```

# 6.15 コンテナーでのジョブの実行

PBSは、DockerおよびSingularityコンテナーでのマルチvnode、マルチホスト、およびインタラクティブジョブの実行をサポートしています。

パブリックレジストリから抽出できるほか、ログインできるプライベートレジストリであれば、そこから抽出することもできます。

“qsub -l container_image=hello-world”などのスクリプトを指定しない場合、qsubによって、スクリプトについてインタラクティブに尋ねられます。

qsubにスクリプトを指定すると、PBSは指定されたコンテナー内でそのスクリプトを実行します。

マルチホストジョブの場合、コンテナーには任意のバージョンのOpenMPIを使用できます。

コンテナーを存続させるため、PBSはこのコンテナー内で無限のsleepコマンドを実行します。

## 6.15.1 コンテナーエンジンの要求

値がコンテナーエンジンに設定されているリソースを要求することによりそのコンテナーエンジンを指定するか、リソースを要求せずにデフォルトのコンテナエンジンを使用できます。要求できるコンテナーエンジンは、ホストレベルで要求する場合でも、ジョブごとに1つのみです。すべてのチャンクで同じコンテナーエンジンを要求する必要があります。使用可能なコンテナーエンジンをリストするリソースの名前を管理者に尋ねるか、pbsnodes（コンテナーエンジン名の検索）を使用して探します。推奨されるリソース名は“container engine”です。

*qsub ...-l select=ncpus=...:<コンテナーエンジンリソース>=<コンテナーエンジン>*

## 6.15.2 コンテナーイメージの要求

-l container_image=<コンテナーイメージ>を使用するか、CONTAINER_IMAGE環境変数をイメージの名前に設定し、ジョブでその環境変数を渡すことによって、そのジョブのコンテナーイメージを要求します。

*qsub ...-l container_image=<コンテナーイメージ> ...*

または

*qsub ...-v CONTAINER_IMAGE=<コンテナーイメージの名前> ...*

### 6.15.2.1 レジストリの指定

レジストリを指定しないと、管理者が設定したデフォルトのレジストリが使用されます。コンテナーイメージにレジストリを指定できます。

例6-10： コンテナーイメージにレジストリ（およびネームスペース）を指定するには次のように入力します。

```
qsub -v CONTAINER_IMAGE=pbsprohub.local/pbsuser/test-image
```

### 6.15.2.2 プライベートレジストリからの抽出

PBSでプライベートレジストリから抽出するには、そのレジストリにログインするためのクレデンシャルファイルを使用します。このファイルはJSONフォーマットです。

#### 6.15.2.2.i レジストリのクレデンシャルファイル名

レジストリのクレデンシャルファイル名のフォーマットは次のとおりです。

*<ジョブ所有者>/.container/tokens.json*

#### 6.15.2.2.ii レジストリのクレデンシャルファイルのフォーマット

このファイルの内容は次のフォーマットで記述されています。

```
{
   "registry1 <URL>/<エンドポイント>": {
      "user_id" : "<ユーザー ID>" , "passwd" : "<生成されたOAUTH トークンとパスワード>"
   },

   "registry2 <URL>/<エンドポイント>": {
      "user_id" : "<ユーザー ID>" , "passwd" : "<生成されたOAUTH トークンとパスワード>"
   }
}
```

#### 6.15.2.2.iii レジストリのクレデンシャルファイルのデフォルト値

*registry* : デフォルトのレジストリ（allowed_registries パラメータの最初の要素）

*user_id* : ジョブオーナー。空の場合は、ジョブオーナーの ID の使用が試されます。

*passwd* : パスワードなし

#### 6.15.2.2.iv レジストリのクレデンシャルファイルの場所

レジストリのクレデンシャルファイルの *ベースパス* は、そのファイルの格納場所へのパスですが、<ジョブ所有者>/.container/tokens.json は含まれません。レジストリのクレデンシャルファイルへのデフォルトのパスは /home です。管理者は、レジストリのクレデンシャルファイルが実際に格納されている場所をこのベースパスとして設定できます。

例 6-11 : このベースパスが “/container/creds/” で、ジョブオーナーが User1 であるとします。JSON ファイルへのフルパスは次のようになります。

```
/container/creds/User1/.container/tokens.json
```

### 6.15.2.3 イメージのネームスペースの指定

別途指定しない限り、PBS ではレジストリのデフォルトのネームスペースがコンテナーイメージ向けに使用されます。必要なイメージがデフォルトではないネームスペースに存在する場合は、イメージ名とともにそのネームスペースを指定します。

例 6-12 : “MyImages” ネームスペースを使用して、Docker コンテナーエンジンとイメージ “centos” を要求するには次のように指定します。

```
qsub -l select=1:ncpus=1:container_engine=docker -lcontainer_image="MyImages/centos"  -- /bin/sleep
    500
```

例 6-13 : デフォルトのネームスペースを使用して、Docker コンテナーエンジンとイメージ “centos” を要求するには次のように指定します。

```
qsub -l select=1:ncpus=1:container_engine=docker -lcontainer_image="centos"  -- /bin/sleep 500
```

## 6.15.3 Dockerコンテナーでのポートの指定

Dockerコンテナーの単一vnodeジョブに対して、アプリケーションのポートを要求することができます。PBSは要求されたポートをホスト上の使用可能なポートにマップし、そのマッピングを返します。ポートを要求するには、container_portsジョブリソースでポート番号をコンマで区切ってリストします。ポート番号のリストは、一重引用符で囲む必要があります。PBSは、ジョブのresources_used.container_ports値をコンマ区切りの*<コンテナーポート>:<ホストポート>*ペアに設定します。たとえば、ジョブは次のようにして特定のポートを要求できます。

```
qsub -l container_ports="'2324,8989'" ...
```

PBSは、ジョブのresources_used.container_portsリソースでポートマッピングを返します。

```
resources_used.container_ports = 2324:8080,8989:32771
```

## 6.15.4 コンテナーエンジンの追加引数の指定

PBS_CONTAINER_ARGS環境変数を介して、コンテナーエンジンの追加引数を指定できます。この変数はセミコロンで区切られたリストとして指定します。たとえば、`--shm-size`を1GBに、`--tmpfs`を“/run:rw,noexec,nosuid,size=65536k”に指定する場合は、次のようになります。

```
export PBS_CONTAINER_ARGS="--shm-size=1GB";"--tmpfs /run:rw,noexec,nosuid,size=65536k"
```

追加引数をジョブで使用するには、PBS管理者がこれらをホワイトリストに追加しておく必要があります。

`docker run`の`--env`および`--entrypoint`引数はサポートされていません。

## 6.15.5 コンテナーへの環境変数の引き渡し

環境変数を直接PBSに渡すには、`qsub -v <環境変数リスト>`を使用します。コンテナーへの環境変数の引き渡しでは`--env`引数を使用できません。

## 6.15.6 Dockerコンテナーのセカンダリグループへのジョブオーナーの追加

管理者は、コンテナー内のセカンダリグループにジョブオーナーを追加するようにPBSを設定できます。このグループは、ジョブオーナーがすでにメンバーになっている実行ホスト上のグループです。Singularityでは自動的にジョブオーナーをすべてのグループに追加するため、この機能はDockerコンテナーに対してのみ適用されます。

## 6.15.7 Singularityコンテナーでの単一vnode単一ホストジョブの実行

PBSを使用したコンテナーの起動に加え、スクリプト、実行可能ファイル、またはコマンドの先頭にSingularityバイナリを付加することにより、いつでも単一のSingularityコンテナーで単一のvnodeジョブを実行できます。

## 6.15.8 コンテナーでのシェルの指定

コンテナー内で追加の手順を実行することなくデフォルトのシェルを実行できます。デフォルトに加えて何かを使用してコンテナーでシェルを実行するには、`qsub`の`-S`オプションを使用してそのシェルを指定する必要があります。選択したシェルがコンテナー内部で使用できることを確認します。

### 6.15.9 注意事項と制限事項

- コンテナーでは-lncpusなどの古いスタイルのリソース要求を使用することはできません。
- コンテナー内のエントリポイントは無効です。エントリポイントコマンド相当のものを実行する場合は、コマンドラインで引数を使用した完全なコマンドを指定する必要があります。
- コンテナー内の一部のディレクトリまたはファイルのマウントが制限される場合があります。詳細は管理者に問い合わせてください。

### 6.15.10 PBSでのクラウドバースティングの制限事項と注意事項

- クラウドバースティングは、Linux環境のみでサポートされます。
- クラウドノードでは、予約はサポートされていません。

## 6.16 ジョブへのvnode障害の許容設定

管理者がvnodeの障害を許容するようにPBSを設定している場合、ユーザーのジョブでvnodeの障害を許容できるようにすることができます。PBSではユーザーが追加のvnodeをジョブに割り当てることができ、一部のvnodeに障害が発生してもジョブを正常に開始し、実行することができます。PBSでは、追加のvnodeを起動用にのみ割り当てることも、ジョブの存続期間全体に対して割り当てることもできます。後で、起動の信頼性確保のためにのみ追加のvnodeを必要としているジョブについては、PBSが割り当てられたvnodeをジョブが実行で使用する分のみに削減し、不要なvnodeを他のジョブのために解放します。

ジョブが起動時のみvnodeの障害を許容できるようにするには、ジョブのtolerate_node_failures属性を“*start*”に設定します。

ジョブがその存続期間全体でvnodeの障害を許容できるようにするには、ジョブのtolerate_node_failures属性を“*all*”に設定します。

この属性の設定例：

- qsubを使用：

  **qsub -W tolerate_node_failures="all" <ジョブスクリプト>**

- qalterを使用：

  **qalter -W tolerate_node_failures="job_start" <ジョブID>**

# 7 リソースの予約

本章では、ジョブの予約（先行予約、継続予約、ジョブ固有予約）についてのみ説明します。メンテナンス予約については、『PBS Professional Administrator's Guide』の「Reservations」（194 ページ）をご参照ください。

## 7.1 用語集

### 先行予約

ある特定の時刻における一連のリソースの予約です。予約は、予約の作成者およびその作成者によって指定されたユーザーまたはグループのみが利用できます。

### degradedとなった予約

関連付けられている1つまたは複数のvnodeが利用できないジョブ固有予約または先行予約です。

最短での予約に関連付けられた 1 つまたは複数の vnode が利用できない継続予約です。

### ジョブ固有予約

特定のジョブに対して作成される、そのジョブが要求したものと同じリソースの予約。

### ジョブ固有ASAP予約

キューイングされている特定のジョブに対して作成される、そのジョブが要求したものと同じリソースの予約。PBSはできるだけ早く実行するようにその予約をスケジュールし、そのジョブを予約に移動します。キューイングされているジョブで`pbs_rsub -Wqmove=<ジョブID>`を使用すると作成されます。

### ジョブ固有即時予約

特定の実行中のジョブに対して作成される予約。PBSは、ジョブが使用しているものと同じリソースで、直ちにジョブ固有即時予約を作成し、ジョブをその予約に移動します。実行中のジョブで`pbs_rsub --job <ジョブID>`を使用すると、この予約が作成され、直ちに実行が開始されます。

### ジョブ固有開始予約

特定のジョブに対して作成される、そのジョブが要求したものと同じリソースの予約。PBSは、スケジューリングポリシーに従ってジョブを開始します。ジョブが開始されると、PBSは予約を作成して開始し、そのジョブを予約に移動します。この予約が作成されるのは、`qsub -Wcreate_resv_from_job=true`を使用してジョブを投入した場合、またはジョブに対して`qalter`を実行してそのジョブのcreate_resv_from_job属性を*True*に設定した場合です。

### 継続予約の発生

継続予約のインスタンスです。

継続予約の発生は先行予約と同様に動作しますが、次の場合は例外です。

- ジョブは特定の先行予約に投入できますが、継続予約では特定の発生ではなく継続予約全体に対してのみ投入できます。ジョブが実行可能になる*時間*のみを指定できます。『PBS Professional Reference Guide』の「qsub」（216ページ）をご参照ください。
- 先行予約の終了時は、その予約および予約の全ジョブが実行中でもキューイング中でも削除されますが、発生の終了時は、その実行中のジョブのみが削除されます。

継続予約の各発生にはリソース要求を満たす予約済みリソースが含まれますが、発生ごとに別のソースからリソースが割り当てられる場合があります。継続予約に割り当てられるリソースを照会すると、`pbs_rstat`で報告されるresv_nodes属性には最短での予約に割り当てられるリソースが返されます。

### 継続予約の最短での予約

現在アクティブな発生か、現在アクティブな発生がない場合は、次の発生を表します。

### 継続予約

特定の時間に繰り返し発生する先行予約です。たとえば、翌月からの３ヶ月間の毎週水曜日および木曜日に、午後5時から8時まで８つのCPUと10GBを予約できます。

# 7.2 ジョブの予約の簡単な説明

後でジョブによって使用されるリソースを予約したり、特定のジョブによって要求されるリソースを使用して予約を作成し、そのジョブを予約に移動することができます。

先行予約または継続予約を作成し、ジョブをその予約に投入します。**先行予約**では、特定のリソースを特定の期間予約します。**継続予約**も同様ですが、この期間が繰り返されます。

PBSは、キュー待機中のジョブが要求しているもの、または実行中のジョブが使用しているものと同じリソースを予約して、そのジョブを予約キューに移動することにより、**ジョブ固有予約**を作成します。

- PBSは、create_resv_from_job属性が*True*のキューイングされている特定のジョブに対してジョブ固有開始予約を作成します。ジョブが実行されると、PBSは予約を作成して開始し、そのジョブを予約に移動します。この予約により、後で、再度ジョブがスケジュールされるのを待つことなく、ジョブの再実行が可能になります。この属性は、投入時に`qsub -Wcreate_resv_from_job=true`を使用して設定できます。
- 特定のキューイングされているジョブで`pbs_rsub -Wqmove=<ジョブID>`を使用すると、PBSはこのジョブに対してジョブ固有ASAP予約を作成します。PBSは予約を作成し、ジョブをその予約に移動して、できるだけ早く実行されるように予約をスケジュールします。
- 特定の実行中のジョブで`pbs_rsub --job <ジョブID>`を使用すると、PBSはこのジョブに対してジョブ固有即時予約を作成します。PBSは、直ちに予約を作成し、これを開始して、ジョブをこの予約に移動します。この予約により、再度ジョブがスケジュールされるのを待つことなく、ジョブの再実行が可能になります。

# 7.3 リソースを予約するための前提条件

予約を要求する時間は、投入ホストのタイムゾーンで設定します。

また、投入ホストのPBS_TZID環境変数を設定する必要があります。PBS_TZIDの形式はタイムゾーンの場所になります。例：America/Los_Angeles, America/Detroit, Europe/Berlin, Asia/Kolkata. 10ページの第1.4.4節「投入ホストのタイムゾーンの設定」をご参照ください。

# 7.4 先行予約と継続予約

## 7.4.1 先行予約と継続予約の作成および使用の概要

先行予約および継続予約とも、作成にはpbs_rsubコマンドを利用できます。PBSは、予約が可能かどうかを確認するか、または要求を拒否します。予約が確認されると、PBSでは予約のジョブに対するキューが作成されます。予約を利用して実行するジョブは、このキューに投入します。

予約の確認は、現在実行中のジョブ、他の確認済みの予約または専用時間と競合せずに、要求されたリソースを利用可能な場合に成立します。このテストでエラーが発生した予約要求は、拒否されます。継続予約の確認には、継続予約の全発生に対し、リソースが利用可能であることが確認される必要があります。

pbs_rsubコマンドは予約名の*予約ID*を返します。先行予約の場合、この予約IDは次のようなフォーマットになります。

*R<シーケンス番号>.<サーバー名>*

継続予約の場合、この予約IDはシリーズ全体を参照し、次のようなフォーマットになります。

*S<シーケンス番号>.<サーバー名>*

ユーザーは、ジョブの場合と同じ構文を使用して、予約のためのリソースを指定します。予約内のジョブは、非予約ジョブがplacement setに配置されるのと同じように配置されます。

予約を要求する時間は、投入ホストのタイムゾーンで設定します。

pbs_rsubコマンドでは、予約ID文字列および予約の現在のステータスを返します。

予約がアイドル状態の場合、指定した時間が経過した後、自動的に削除されるように先行予約または継続予約を作成することができます。継続予約の場合、これは各予約に対して個別に適用されます。継続予約の1つの予約が削除されても、次の予約は指定された時間に開始されます。予約を自動的に削除するには、pbs_rsub -Wdelete_idle_time=<許容されるアイドル時間>を使用し、秒数を整数で指定するか、その期間を*HH:MM:SS*で指定します。

pbs_rsubコマンドのオプションについては、『PBS Professional Reference Guide』の「pbs_rsub」（96ページ）をご参照ください。

## 7.4.2 先行予約の作成

先行予約を作成するには、pbs_rsubコマンドを使用します。PBSでは予約の開始時刻および終了時刻を計算して、以下に示す3つのオプションの2つを指定する必要があります。

D 期間

E 終了時刻

R 開始時刻

### 7.4.2.1 先行予約のタイムゾーンの設定

先行予約のタイムゾーンをUTCにする必要がある場合、予約作成時に以下のように設定します。

```
TZ=UTC pbs_rsub -R...
```

### 7.4.2.2 先行予約の作成例

次の例では、1つのvnode、経過時間30分、開始時刻11:30を要求する先行予約を作成します。終了時刻が指定されていないため、PBSでは予約の開始時刻と期間に基づいて終了時刻を計算します。

```
pbs_rsub -R 1130 -D 00:30:00
```

PBSから予約IDが返されます。

```
R226.south UNCONFIRMED
```

次の例では、２つのCPUを午後8時から午後10時まで要求する先行予約を示しています。

```
pbs_rsub -R 2000.00 -E 2200.00 -l select=1:ncpus=2
```

PBSから予約IDが返されます。

```
R332.south UNCONFIRMED
```

## 7.4.3 継続予約の作成

継続予約を作成するには、pbs_rsubコマンドを使用します。継続予約を作成する場合は、開始時刻と終了時刻を指定する**必要があります**。予約の繰り返し特性は、pbs_rsubに-rオプションを使用して指定します。-rオプションは、継続予約の発生を指定するrecurrence_rule引数を取得します。繰り返し規則では、iCalendar構文を使用し、RFC 2445に記載されているパラメータのサブセットを使用します。

繰り返し規則は２つの形式で指定できます。

*"FREQ=<頻度指定>;COUNT=<回数指定>;<間隔指定>"*

この形式では、発生の頻度、回数、および適用する曜日または時刻（あるいはその両方）を指定します。

*"FREQ=<頻度指定>;UNTIL=<期限指定>;<間隔指定>"*

繰り返し規則には空白を入れないでください。

この形式では、発生の頻度、終了日時、および適用する曜日または時刻（あるいはその両方）を指定します。

*頻度指定*

予約が繰り返される頻度を指定します。有効値は*WEEKLY|DAILY|HOURLY*です。

*頻度指定*に*WEEKLY*を使用している場合は、*間隔指定*の*BYDAY*または*BYHOUR*（あるいはその両方）を使用できます。*頻度指定*に*DAILY*を使用している場合は、*間隔指定*の*BYHOUR*を使用できます。*頻度指定*にHOURLYを使用している場合は、*間隔指定*を使用しないでください。

*回数指定*

正確な発生数です。最大で４桁の数字を指定できます。フォーマット：整数。

*間隔指定*

発生の間隔を指定します。*BYDAY=<曜日>*または*BYHOUR=<時刻>*のいずれか、または両方を設定できます。有効値は*BYDAY = MO|TU|WE|TH|FR|SA|SU*および*BYHOUR = 0|1|2|...|23*です。曜日と時刻の両方を使用する場合は、両者をセミコロンで区切ります。曜日や時刻を複数指定する場合は、それぞれコンマで区切ります。

たとえば、火曜日と水曜日の午前9時と午前11時に繰り返し発生する設定にするには、BYDAY=TU,WE;BYHOUR=9,11と指定します。

*BYDAY*は*FREQ=WEEKLY*と組み合わせて使用する必要があります。*BYHOUR*は*FREQ=DAILY*または*FREQ=WEEKLY*と組み合わせて使用する必要があります。

*期限指定*

発生はこの日時まで開始されますが、それを過ぎると開始しなくなります。つまり、1時間継続し、通常は午前9時に開始する発生の場合、*期限指定*に午前9時5分が指定されていると、その当日には発生が開始します。

フォーマット：*YYYYMMDD[THHMMSS]*

年月日の部分と時分秒の部分は、大文字の*T*で区別されます。

デフォルト：予約作成日時から３年後

### 7.4.3.1 予約の開始時刻と期間の設定

継続予約では、pbs_rsubの-Rおよび-Eオプションの引数を使用して、先行予約よりも多くの情報を提供できます。先行予約の場合、これらの引数が提供する情報は予約の開始時刻と終了時刻です。継続予約では、開始時刻と終了時刻を提供できるだけでなく、これらの引数を使用して、期間および間隔の開始時点からのオフセットを計算することも可能です。

-Rおよび-Eの引数の値によって、予約の期間が異なります。たとえば、次のように指定すると、

```
-R 0930 -E 1145
```

予約の期間は2時間15分になります。次のように指定すると、

```
-R 150800 -E 170830
```

予約の期間は2日と30分になります。

*間隔指定*では、間隔を開始する曜日または時刻を指定できます。次のように指定すると、

```
-R 0915 -E 0945 ...BYHOUR=9,10
```

期間は30分、間隔の開始時点からのオフセットは15分になります。間隔は9時に開始し、再び10時に開始します。予約は9時15分から9時45分まで実行され、再び10時15分から10時45分まで実行されます。同様に、次のように指定すると、

```
-R 0800 -E -1000 ...BYDAY=WE,TH
```

期間は2時間、間隔の開始時点からのオフセットは8時間になります。予約は水曜日の8時から10時まで実行され、再び木曜日の8時から10時まで実行されます。

-Rおよび-Eオプションの引数で指定した要素は、繰り返し規則で指定した要素に変更されます。したがって、次のように指定すると、

```
-R 0730 -E 0830 ...BYHOUR=9
```

期間は1時間ですが、-Rの引数で指定した時刻の要素（7:00）は、繰り返し規則の時刻の要素（9:00）に変更されます。間隔の開始時点からのオフセットは30分のままです。予約は9時30分から10時30分まで実行されます。同様に、16日が月曜日の場合、次のように指定すると、

```
-R 160800 -E 170900 ...BYDAY=TU;BYHOUR=11
```

期間は25時間ですが、曜日と時刻の要素は指定変更されます。この予約は火曜日の11時から25時間実行され、水曜日の12時に終了します。ただし、次のように指定すると、

```
-R 160810 -E 170910 ...BYDAY=TU;BYHOUR=11
```

期間は25時間、間隔の開始時点からのオフセットは10分になります。この予約は火曜日の11時10分から25時間実行され、水曜日の12時10分に終了します。オフセットの分数は繰り返し規則のいかなる要素によっても変更されませんでした。

-Rおよび-Eオプションの引数に指定された値は、先行予約の場合と同様に、継続予約の開始時刻と終了時刻の設定に使用できます。その場合は、繰り返し規則の要素で指定変更しないでください。次のように指定すると、

```
-R 0930 -E 1030 ...BYDAY=MO,TU
```

時間または分の要素は指定変更されません。予約は月曜日と火曜日の9時30分から10時30分まで実行されます。

### 7.4.3.2 継続予約の作成の要件

- ユーザーは開始日と終了日を指定する必要があります。
- また、投入ホストのPBS_TZID環境変数を設定する必要があります。PBS_TZIDの形式はタイムゾーンの場所になります。例：America/Los_Angeles、America/Detroit、Europe/Berlin、Asia/Calcutta。10ページの第1.4.4節「投入ホストのタイムゾーンの設定」をご参照ください。
- 繰り返し規則は改行を入れずに1行で記述する必要があります。
- 繰り返し規則は二重引用符で囲む必要があります。
- 特定のキューからのみジョブを受け入れるよう設定されているvnode（vnodeキュー制約）は、先行予約または継続予約には使用できません。特定のキューからのみジョブを受け入れるように設定されているvnodeの有無について確認するには、PBSの管理者にお問い合わせください。
- 繰り返し規則には空白を入れないでください。

### 7.4.3.3 継続予約の作成例

毎日午前8時から午前10時まで、発生を合計10回繰り返す予約の場合は、次のように指定します。

```
pbs_rsub -R 0800 -E 1000 - r"FREQ=DAILY;COUNT=10"
```

毎週平日の午前6時から午後6時まで、2008年12月10日まで実行する場合は、次のように指定します。

```
pbs_rsub -R 0600 -E 1800 -r "FREQ=WEEKLY;BYDAY=MO,TU,WE,TH,FR;UNTIL=20081210"
```

毎週月曜日、水曜日、金曜日の午後3時から午後5時まで、発生を9回（つまり3週間）繰り返す場合は、次のように指定します。

```
pbs_rsub -R 1500 -E 1700 -r "FREQ=WEEKLY;BYDAY=MO,WE,FR;COUNT=9"
```

# 7.5 ジョブ固有予約

## 7.5.1 ジョブ固有開始予約

PBSは通常通りジョブを実行し、ジョブが開始されると、*ジョブ固有開始予約*を作成して開始し、そのジョブを予約に移動します。PBSは、ジョブが使用しているものと同じリソースを使用して予約を作成します。予約では、ジョブが失敗して再投入する必要がある場合に備えて、そのジョブに必要なリソースを保持し、スケジュールされるのを待たずに再度実行できるようにします。予約はジョブが開始されると開始され、ジョブと同時に終了します。

キューイングされているジョブが失敗するおそれがあり、修正して再投入する必要があると思われる場合は、ジョブ固有の*開始予約*を作成できます。ジョブを投入する際、qsubの-Wオプションを使用して、create_resv_from_job属性を*True*に設定します。

*qsub ...-Wcreate_resv_from_job=true*

たとえば、スクリプト名がmyscript.shのジョブのジョブ固有開始予約を作成する場合は、次のように指定します。

```
qsub -Wcreate_resv_from_job=true myscript.sh
```

キューイングされているジョブに`qalter`を適用して、この属性を設定することもできます。

*qalter -Wcreate_resv_from_job=true <ジョブID>*

たとえば、ジョブ1234.myserverの開始時に開始予約を作成する場合は、次のように指定します。

```
qalter -Wcreate_resv_from_job=true 1234.myserver
```

ジョブ固有開始予約IDは次のようなフォーマットになります。

*R<シーケンス番号>.<サーバー名>*

PBSは、開始予約のreserve_job属性を予約を作成したジョブのIDに、予約のReserve_Owner属性をジョブのJob_Owner属性の値に、予約のresv_nodes属性をジョブのexec_vnode属性に、予約のリソースをジョブのschedselect属性に、予約のResource_List属性をジョブのResource_List属性にそれぞれ設定します。

開始予約の期間および開始時刻は、ジョブのwalltimeおよび開始時刻と同じです。ジョブがピアスケジューリングされている場合、即時予約は抽出コンプレックスで作成されます。

開始予約はジョブが実行を開始すると作成されます。create_resv_from_job属性はいつでも*True*に設定できますが、これが有効になるのはジョブが開始される前に設定した場合のみです。ジョブが実行を開始してからそれに対してジョブ固有予約を作成する必要が生じた場合は、ジョブ固有即時予約を作成します。150ページの第7.5.3節「ジョブ固有即時予約」をご参照ください。

キューイングされているジョブでのみ使用できます。

ジョブアレイ、別の予約に投入されているジョブ、またはその他のユーザーのジョブでは使用できません。

## 7.5.2 ジョブ固有ASAP予約

できるだけ早く開始するには、*ジョブ固有ASAP予約*をスケジュールします。PBSは、キューイングされている特定のジョブが要求しているリソースを使用して*ジョブ固有ASAP予約*を作成し、このジョブをその予約に移動します。

他のジョブについても`qmove`でこのキューに移動するか、`qsub`でこのキューに投入できます。

ASAP予約を作成するには、次のように指定します。

*pbs_rsub -W qmove=<ジョブID>*

たとえば、ジョブ1234.myserverのASAP予約を作成する場合は、次のように指定します。

```
pbs_rsub -W qmove="1234.myserver"
```

ジョブ固有ASAP予約IDは次のようなフォーマットになります。

*R<シーケンス番号>.<サーバー名>*

`-W qmove`オプションを使用する場合、`pbs_rsub`に対する-Rオプションおよび-Eオプションは無効になります。

ジョブアレイでは使用できません。

詳細については、『PBS Professional Reference Guide』の「pbs_rsub」（96ページ）をご参照ください。

ASAP予約は、ジョブのwalltimeを設定するサイトに対してのみ使用することをお勧めします。ジョブのwalltimeのデフォルトは5年です。したがって、システム内のすべてのジョブにwalltimeのデフォルトが設定されている場合、ASAP（できる限り早い）予約の開始時刻は5年後以降になります。

ASAP予約のdelete_idle_time属性のデフォルト値は10分です。

### 7.5.3 ジョブ固有即時予約

PBSは、実行中のジョブが使用しているものと同じリソースで*ジョブ固有即時予約*を作成し、これを開始して、その実行中のジョブをその予約に移動します。予約では、ジョブが失敗して再投入する必要がある場合に備えて、そのジョブに必要なリソースを保持し、スケジュールされるのを待たずに再度実行できるようにします。

実行中のジョブが変更および再投入を必要としていることがわかり、スケジューラがスロットを見つけるまで待ちたくない場合、即時予約を作成することができます。後で、変更されたジョブを予約に投入できます。

*pbs_rsub --job <ジョブID>*

たとえば、ジョブ1234.myserverの実行中に、このジョブの即時予約を作成する場合は、次のように指定します。

```
pbs_rsub --job 1234.myserver
```

ジョブ固有即時予約IDは次のようなフォーマットになります。

*R<シーケンス番号>.<サーバー名>*

PBSは、ジョブのcreate_resv_from_job属性を*True*に、即時予約のreserve_job属性を予約を作成したジョブのIDに、予約のReserve_Owner属性をジョブのJob_Owner属性の値に、予約のresv_nodes属性をジョブのexec_vnode属性に、予約のリソースをジョブのschedselect属性に、予約のResource_List属性をジョブのResource_List属性にそれぞれ設定します。

即時予約の期間および開始時刻は、ジョブのwalltimeおよび開始時刻と同じです。ジョブがピアスケジューリングされている場合、即時予約は抽出コンプレックスで作成されます。

実行中のジョブ（*R*状態でサブステータスが*42*のジョブ）でのみ使用可能です。

ジョブアレイ、すでに予約されているジョブ、その他のユーザーのジョブでは使用できません。

## 7.6 予約の確認の取得

デフォルトでは、予約の確認または拒否について、pbs_rsubコマンドからユーザーに即時通知されることはありません。代わりに、この情報に関する電子メールを受信します。pbs_rsubコマンドに対し、予約の確認を取得するまで待機するように指定するには、-l <*ブロック時間*>オプションを使用します。pbs_rsubコマンドでは、予約の確認または拒否を取得するまで、最大で*ブロック時間*で指定した秒数待機してから、ユーザーにその結果を通知します。*ブロック時間*に負の値が指定され、その時間では予約が確認できない場合、予約は自動的に削除されます。

予約が確認されたかどうかを調べるには、pbs_rstatコマンドを使用します。このコマンドによって予約の状態が表示されます。COおよびRESV_CONFIRMEDは予約が確認されたことを示します。pbs_rstatからの出力に予約が表示されていない場合は、予約が拒否されたことを意味します。

予約に関するメール通知を確実に受け取れるようにするには、pbs_rsubの-M <*電子メールアドレス*>オプションを使用して、予約のMail_Users属性を設定します。デフォルトでは、予約の終了時または確認時に電子メールを受信するように設定されています。予約の確認以外のイベントについて電子メールを受け取れるようにするには、-m <*メールイベント*>オプションを使用して、予約のMail_Points属性を設定します。詳細については、『PBS Professional Reference Guide』の「pbs_rsub」（96ページ）および『PBS Professional Reference Guide』の「Reservation Attributes」（303ページ）をご参照ください。

# 7.7 予約の変更

pbs_ralterコマンドを使用して、既存の予約を変更できます。この予約がジョブ固有予約や先行予約であってもかまわず、継続予約の次または現在のインスタンスであってもかまいません。構文：

*pbs_ralter [-D <duration>] [-E <end time>] [-G <auth group list>] [-l <block time>] [-l select=<select spec>] [-m <mail points>] [-M <mail list>] [-N <reservation name>] [-R <start time>] [-U <auth user list>] <reservation ID>*

予約がアイドル状態の場合、指定した時間の経過後にその予約が自動的に削除されるように先行予約または継続予約を変更できます。継続予約の場合、これは各予約に対して個別に適用されます。継続予約の1つの予約が削除されても、次の予約は指定された時間に開始されます。予約を自動的に削除するには、`pbs_ralter -Wdelete_idle_time=<`許容されるアイドル時間`>`を使用し、秒数を整数で指定するか、その期間を*HH:MM:SS*で指定します。この属性を変更する際は、他のどの予約属性も変更できません。

予約内でジョブが実行中の場合は、その予約の開始時間を変更することはできません。

select指定を変更する場合は、実行中のジョブがあるかどうかによって動作が異なります。

- その予約内でジョブが実行中の場合は、次のことが当てはまります。
  - 予約のジョブが実行されている場所であるチャンクを解放することはできません。
  - ジョブが実行されているvnodeを変更できませんが、それ以外の操作は可能です。
- ジョブがまったく実行されていない場合は、select指定を全面的に変更できます。

チャンクを要求する際は、それぞれのチャンク要求で必ず単一タイプのチャンクを指定してください。

実行中の予約内で未使用のチャンクを見つけるには、その予約のresv_nodes属性を、その予約内で実行中のジョブのexec_vnode属性と比較します。

その予約がまだ開始されていない場合は、select指定を変更すると、その予約が異なるvnodeに移動される可能性があります。

変更が要求されると、その変更は確定または拒否されます。変更を拒否する場合、その予約は削除されずにそのまま残り、サーバーのログに次のメッセージが表示されます。

```
Unable to alter reservation <予約ID>
```

予約が確定されると、サーバーのログに次のメッセージが表示されます。

```
Reservation alter successful for <予約ID>
```

変更が許可されたかどうかを確認するには、以下のようにします。

- `pbs_rstat`コマンドを使用：予約属性が変更されたかどうかを確認
- インタラクティブオプションを使用：ブロック時間が切れた後に確定されたかどうかを確認

予約が開始されず、同じvnodeで確定できない場合、PBSは別のvnodeセットを探します。『PBS Professional Administrator's Guide』の第8.4節「Reservation Fault Tolerance」（401ページ）をご参照ください。

このコマンドを実行するには、予約所有者かPBS管理者である必要があります。

詳細については、『PBS Professional Reference Guide』の「pbs_ralter」（85ページ）をご参照ください。

### 7.7.0.0.i 予約の変更例

例7-1： 予約を拡大します。

既存：

```
select=100:ncpus=20:mem=512gb
```

```
pbs_ralter -l select=150:ncpus=20:mem=512gb
```

例7-2： 予約を拡大および縮小します。

既存：

```
select=100:ncpus=20+10:ncpus=10:mem=512gb
```

**pbs_ralter -l select=150:ncpus=20+5:ncpus=10:mem=512gb**

例7-3： 予約を拡大して、任意タイプのチャンクを削除します。

既存：

```
select=100:ncpus=20+10:ncpus=10:mem=512MB+15:ncpus=40
```

**pbs_ralter -l select=150:ncpus=20+30:ncpus=40**

例7-4： 実行中のジョブがない状態で、selectを全面的に変更します。

既存：

```
select=100:ncpus=20+10:ncpus=10:mem=512GB
```

**pbs_ralter -l select=150:ncpus=20:mem=1024GB+5:ncpus=15:mem=512GB**

例7-5： 1つ目のタイプのチャンクの50個のvnode上でジョブが実行されている状態で、予約を拡大および縮小します。

既存：

```
select=100:ncpus=20+50:ncpus=40
```

**pbs_ralter -l select=50:ncpus=20+100:ncpus=40**

例7-6： 望ましくない例です。1つ目のタイプのチャンクの50個のvnode上でジョブが実行されている状態で、実行中のジョブからチャンクを削除しようとすることで、無効な変更を加えることを試みます。

既存：

```
select =100:ncpus=20+50:ncpus=40
```

**pbs_ralter -l select=25:ncpus=20+100:ncpus=40**

```
ALTER DENIED
```

# 7.8 予約の削除

予約を削除するには、pbs_rdelコマンドを使用します。継続予約の場合は、すべての発生を含む予約全体を削除することしかできません。予約を削除すると、予約に投入されているすべてのジョブも削除されます。予約を削除できるのは、予約のオーナー、PBSオペレータ、またはPBSマネージャです。たとえば、S304.southを削除するには、次のように指定します。

**pbs_rdel S304.south**

または

**pbs_rdel S304**

予約がアイドル状態の場合、指定した時間が経過した後、自動的に削除されるように予約を作成することができます。継続予約の場合、これは各予約に対して個別に適用されます。継続予約の1つの予約が削除されても、次の予約は指定された時間に開始されます。予約を自動的に削除するには、pbs_rsub -Wdelete_idle_time=<許容されるアイドル時間>を使用し、秒数を整数で指定するか、その期間を*HH:MM:SS*で指定します。

# 7.9 予約の状態の表示

次の表には、使用される可能性のある予約の状態をリストで示します。よく使用される状態には、CO、UN、BD、およびRNがあります。ただし、通常は予約が未確認の状態にある期間は非常に短いため、ユーザーがこれらの状態を目にすることはありません。『PBS Professional Reference Guide』の「Reservation States」（367ページ）をご参照ください。

予約の状態を表示するには、pbs_rstatコマンドを使用します。このコマンドでは、PBSサーバーにおけるすべての予約の状態が表示されます。継続予約の場合は、pbs_rstatコマンドを使用すると、最短での予約の状態が表示されます。期間は秒単位で表示されます。pbs_rstatコマンドを使用しても、非表示の設定で作成されたカスタムリソースは表示されません。62ページの第4.3.8節「リソース要求に関する注意事項と制限事項」をご参照ください。このコマンドには３つのオプションがあります。

**表7-1：pbs_rstatコマンドのオプション**

| オプション | 意味 | 説明 |
|---|---|---|
| B | Brief | 予約名のみを表示 |
| S | Short | 予約ごとに予約名、キュー名、オーナー、状態、開始時刻、期間、および終了時刻を表形式で表示 |
| F | Full | 予約ごとに予約名およびデフォルト以外の値が設定されている属性を表示 |
| <なし> | デフォルト | デフォルトはSオプション |

継続予約の詳細リストは、先行予約のリストと同じです。ただし、次の内容が追加されます。

- 繰り返し規則を指定する行：
  reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5
- 継続予約の最短での予約に対して予約されたvnodeのエントリ（このエントリは、先行予約でも表示されますが、発生ごとに異なります）：
  resv_nodes=(<vnode名>:...)
- 継続予約の発生の総数を指定する行：
  reserve_count = 5
- 最短での予約のインデックス：
  reserve_index = 1
- 予約を投入したサイトのタイムゾーンを予約のVariable_List属性に追加　（たとえば、カリフォルニアなど）：
  Variable_List=<他の変数>PBS_TZID=America/Los_Angeles

デフォルトサーバー以外のサーバーにおける予約の状態を取得するには、PBS_SERVER環境変数を照会先サーバーの名前に設定してから、pbs_rstatコマンドを使用します。PBSコマンドでは、新しいサーバーをデフォルトのサーバーとして扱うため、終了時にこの環境変数の設定を解除することをお勧めします。

qstatコマンドを使用すると予約のキューに関する情報を取得することも可能です。『PBS Professional Reference Guide』の「qstat」（200ページ）をご参照ください。

## 7.9.1 pbs_rstatを使用した予約の状態の表示例

この例では、先行予約と継続予約をそれぞれ１つずつ使用します。先行予約は、本日、期間は2時間、正午に開始するという設定です。継続予約は、毎週木曜日、期間は１時間、午後３時に開始するという設定です。本日は４月28日月曜日、時刻は1時です。したがって、先行予約は実行中で、継続予約の最短での予約は5月1日木曜日の午後3時です。

briefの出力例：

```
pbs_rstat -B
Name: R302.south
Name: S304.south
```

shortの出力例：

```
pbs_rstat -S

Name Queue User   State   Start   / Duration / End
---------------------------------------------------------
R302.south R302 user1 RN Today 12:00 / 7200/  Today 14:00
S304.south S304 user1 CO May 1 2008 15:00/3600/May 1 2008 16:00
```

fullの出力例：

```
pbs_rstat -F
Name: R302.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_RUNNING
reserve_substate = 5
reserve_start = Mon Apr 28 12:00:00 2008
reserve_end = Mon Apr 28 14:00:00 2008
reserve_duration = 7200
queue = R302
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 02:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:00:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:00:00 2008
Variable_List = PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydomain.com

Name: S304.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_CONFIRMED
reserve_substate = 2
```

```
reserve_start = Thu May 1 15:00:00 2008
reserve_end = Thu May 1 16:00:00 2008
reserve_duration = 3600
queue = S304
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 01:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5
reserve_count = 5
reserve_index = 2
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:01:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:01:00 2008
Variable_List = PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydomain.com,PBS_TZID=America/Los_Angeles
```

# 7.10 予約へのジョブの投入

ジョブは、予約に関連付けられているキューに投入したり、別のキューから予約キューに移動したりすることができます。ジョブを予約に投入するには、qsub コマンドで-q <*キュー*> オプションを使用して、予約キューを指定します。たとえば、S123.south という名前の継続予約の最短での予約にジョブを投入するには、そのキュー S123 に投入します。

```
qsub -q S123 <script>
```

ジョブを予約キューに移動するには、qmove コマンドを使用します。詳細については、『PBS Professional Reference Guide』の「qsub」（216 ページ）および『PBS Professional Reference Guide』の「qmove」（175 ページ）をご参照ください。たとえば、workq から S123（S123.south という名前の予約のキュー）にジョブ 22.myhost を qmove するには、次のように指定します。

```
qmove S123 22.myhost
```

または

```
qmove S123 22
```

実行する日時を制約せずに継続予約に投入されたジョブは、可能であれば最短での予約時に実行されます。特定の発生に対してジョブを投入するには、qsub コマンドの-a <*開始時刻*> オプションを使用して、開示時刻を希望する発生の時刻に指定します。cron ジョブを使用して特定の時刻にジョブを投入することも可能です。『PBS Professional Reference Guide』の「qsub」（216 ページ）およびマニュアルページの cron(8) をご参照ください。

## 7.10.1 予約を使用できるユーザー

予約はデフォルトで、予約を作成したユーザーからのジョブのみを受け入れ、任意のグループまたはホストから投入されたジョブを受け入れます。どのユーザーおよびグループのジョブを予約で受け入れるかまたは受け入れないかを示すリストを指定するには、pbs_rsub および pbs_ralter の-U <*許可ユーザーリスト*>および-G <*許可グループリスト*>オプションを使用して、予約の Authorized_Users および Authorized_Groups 属性を設定します。ジョブの投入が可能または不可能なホストを指定するには、pbs_rsub の-H <*許可ホストリスト*>オプションを使用して'予約の Authorized_Hosts 属性を設定します。

また管理者は、ユーザーおよびグループごとに予約へのジョブの投入を許可するかどうか、およびジョブの投入が可能または不可能なホストのリストを指定することも可能です。

詳細については、『PBS Professional Reference Guide』の「pbs_rsub」（96ページ）および『PBS Professional Reference Guide』の「Reservation Attributes」（303ページ）をご参照ください。

## 7.10.2 予約に投入されたジョブの状態の表示

予約または継続予約の発生に対して投入されたジョブの状態を表示するには、qstat コマンドを使用します。『PBS Professional Reference Guide』の「qstat」（200ページ）をご参照ください。

たとえば、ジョブ名 MyJob が S304.south という名前の継続予約の最短での予約に投入されている場合、このジョブは S304（キュー名）の下にリストされます。

```
qstat

Job id      Name        User         Time Use  S  Queue
----------  ---------  ------------  --------  --  -----
139.south    MyJob      user1              0  Q  S304
```

## 7.10.3 予約でのジョブの取り扱い方法

確認済みの予約は、随時、ジョブをキューで受け入れます。予約期間に入ると、ジョブは予約からのみ実行されるようにスケジュールされます。

予約内のジョブは、予約で要求したリソースの総量を超えるリソースは使用できません。予約ジョブは、要求した walltime が予約期間に収まるかどうかに関係なく予約で受け入れられます。したがって、たとえば、10時から11時まで実行される予約でジョブの walltime が4時間の場合でも、ジョブは開始されます。

先行予約が終了すると、その予約内で実行中またはキュー待機中のジョブは削除されます。

継続予約の発生が終了すると、その予約内で実行中のすべてのジョブが強制終了されます。その予約内でまだキューに残っているジョブは、引き続きキュー待機状態に置かれます。これらのジョブは、今後の発生で実行可能です。継続予約の最後の発生が終了すると、その予約内に残されているジョブは、キュー待機中か実行中かにかかわらず、すべて削除されます。

予約内のジョブに対しては、優先実行できません。

予約内のジョブは、通常のジョブ環境変数を使用して実行されます。134ページの第6.12節「環境変数の使用」をご参照ください。

### 7.10.3.1 予約でのジョブの取り扱い方法の注意事項

ジョブを予約に投入し、ジョブの walltime は予約期間内に収まっていても、ジョブを投入してから予約が終了するまでの期間がジョブの walltime より短い場合、PBS はジョブを開始して、予約終了時にまだ実行中だった場合は強制終了します。

# 7.11 予約の注意事項とエラー

## 7.11.1 タイムゾーンが正しいこと

投入ホストでは、環境変数PBS_TZIDを設定する必要があります。予約が要求される時刻は、投入ホストで定義されている時刻です。10ページの第1.4.4節「投入ホストのタイムゾーンの設定」をご参照ください。

## 7.11.2 予約間に必要な時間

予約間に予約とジョブのクリーンアップに十分な時間をとってください。E、つまり終了状態であってもジョブによってリソースが消費されます。サイズの大きなファイルをステージングする場合、時間がかかることがあります。予約の終了時にジョブがまだ実行中の場合は、クリーンアップに最大２分かかることがあります。ジョブがクリーンアップされるまでは、予約自体のクリーンアップを終了できません。これにより、予約間にクリーンアップを行うために十分な時間がない場合は、次の予約のジョブの開始時刻に遅延が発生します。

## 7.11.3 アカウンティングログ内の予約情報

PBSサーバーでは、ジョブのアカウンティングファイル内に各予約のアカウンティングレコードが書き込まれます。予約のアカウンティングレコードはジョブのアカウンティングレコードと同じです。予約に属しているジョブのアカウンティングレコードには予約IDが含まれています。『PBS Professional Administrator's Guide』の「Accounting」（529ページ）をご参照ください。

## 7.11.4 予約のフォールトトレランス

ジョブ固有予約、先行予約または継続予約の最短での予約に割り当てられている１つまたは複数のvnodeが使用できなくなった場合、予約の状態は*DG*または*RESV_DEGRADED*になります。degradedとなった予約では、ジョブの実行に際して、予約済みのリソースの一部が使用できません。

PBSではdegradedとなった予約の再確認を試みます。つまり、予約の実行に使用可能な代替のvnodeを探します。予約のretry_time属性では、PBSが次に予約の再確認を試行する時刻を表示します。

PBSでdegradedとなった予約を再確認できる場合、予約の状態は*CO*または*RESV_CONFIRMED*のいずれかになり、予約のresv_nodes属性で新しいvnodeが表示されます。

## 7.11.5 ジョブと予約の専有性は一致する必要がある

ジョブ要求で排他的な配置を要求し、それが予約内にある場合、その予約でも`-l place=excl`により排他的な配置を要求する必要があります。

# 8 ジョブアレイ

## 8.1 ジョブアレイの利点

PBSでは、ジョブアレイを使用できます。ジョブアレイは、ほとんど同一のジョブのグループを扱う場合に役に立ちます。ジョブアレイ内の各ジョブは“サブジョブ”と呼ばれます。サブジョブのスケジュールと取り扱いは、通常のジョブと同様です。例外については、この章で説明します。密接に関わる作業を１つのグループにまとめて、このグループを１つの単位として投入、照会、修正、および表示することができるようにします。同一のプログラムを入力ファイルを変えながら繰り返し実行する場合は、ジョブアレイが役に立ちます。PBSは、通常のジョブを個別に扱うよりも、１つのジョブアレイとして扱うほうが効率よく処理できます。ジョブアレイは、パラメータスイープアプリケーション、メディア/エンターテイメントにおけるレンダリング、EDAシミュレーション、外国為替（履歴データ）などのSIMD演算に適しています。

## 8.2 用語集

**ジョブアレイ識別子**

ジョブアレイの投入が成功した際に返される識別子。フォーマット：

*<シーケンス番号>[]*

**ジョブアレイ範囲**

ジョブアレイ内のサブジョブのグループ。範囲を指定する際、使用されるインデックスは'ジョブアレイのインデックスの有効なメンバーである必要があります。

**シーケンス番号**

１つのジョブまたはジョブアレイ識別子の数値部分、たとえば*1234*。

**サブジョブ**

１つのジョブの数多くのプロパティおよび追加のセマンティクスを有するジョブアレイ内の個々のエンティティ（たとえば、*1234[7]*において、*1234[]*はジョブアレイそのもの、*7*はインデックス）（以下に定義）。

**サブジョブインデックス**

１つのジョブを他のジョブと区別するための固有のインデックス。正の整数でなくてはなりません。

# 8.3 ジョブアレイの説明

ジョブアレイは、複数のジョブをコンパクトに表現するものです。ジョブアレイを構成するジョブは“サブジョブ”と呼ばれます。ジョブアレイ内の各サブジョブの取り扱いは、この章で説明する違いを除いて、通常のジョブとまったく同じです。

## 8.3.1 ジョブアレイのジョブスクリプト

ジョブアレイ内のすべてのサブジョブは、1つのジョブスクリプトを共有します。そこには、PBS指示文およびシェルスクリプトの部分を含みます。ジョブスクリプトは、サブジョブごとに1回実行されます。

ジョブスクリプトでは、サブジョブのインデックスに基づいて、異なるコマンドを呼び出すことができます。当然のことながら、このコマンドとしてスクリプトそのものを使用できます。そのためには、サブジョブインデックスを持つ別のコマンドを指定するか、スクリプトに“if”ステートメントを使用します。

## 8.3.2 ジョブアレイの属性およびリソース

1つのジョブアレイ内のサブジョブはすべて同じ属性を持ちます。これには、リソース要件とリミットが含まれます。

ジョブアレイにあるサブジョブごとに同じジョブスクリプトを実行できます。ジョブスクリプトが他のスクリプトやコマンドを呼び出した場合、それらのスクリプトやコマンドは個々のサブジョブの属性とリソースを変更できません。なぜなら、PBSは、コマンドの処理を開始すると、指示文の処理を停止するからです。

## 8.3.3 ジョブアレイとサブジョブのスケジューリング

スケジューラは、ジョブアレイ内の各ジョブを個別のジョブとして処理します。1つのジョブ内のサブジョブはすべて、同じスケジューリング優先度を有します。

## 8.3.4 識別子構文

シーケンス番号（1234[].<サーバー>における1234）は固有のものであるため、ジョブとジョブアレイは1つのシーケンス番号を共有することはできません。同じジョブアレイ内のサブジョブのジョブ識別子は、それぞれのインデックス以外は同じです。各サブジョブは固有のインデックスを有しています。以下の構文書式を使用して、ジョブアレイまたはジョブアレイの一部を参照できます。

- ジョブアレイオブジェクト自体：フォーマットは*<シーケンス番号>[]*または*<シーケンス番号>[].<サーバー>.<ドメイン>.com*です。

  例：1234[].myserverまたは1234[]

- インデックス*M*を有する1つのサブジョブ：フォーマットは*<シーケンス番号>[M]*または*<シーケンス番号>[M].<サーバー>.<ドメイン>.com*です。

  *M*=17の場合の例：1234[17].myserverまたは1234[17]

- ジョブアレイのサブジョブ群の範囲：フォーマットは*<シーケンス番号>[開始-終了[:ステップ]]*または*<シーケンス番号>[開始-終了[:ステップ]].<サーバー>.<ドメイン>.com*です。

  2から8までで、ステップが3の場合の例：1234[2-8:3].myserverまたは1234[2-8:3]

### 8.3.4.1 識別子構文の使用例

| | |
|---|---|
| 1234[] | 短縮ジョブアレイ識別子 |
| 1234[].myserver.domain.com | 全ジョブアレイ識別子 |
| 1234[73] | ジョブアレイ1234[]の73番目のインデックスの短縮サブジョブ識別子 |
| 1234[73].myserver.domain.com | ジョブアレイ1234[]の73番目のインデックスの全サブジョブ識別子 |

### 8.3.4.2 シェルとアレイ識別子

一部のシェル、たとえばcshやtcshなどでは、"[" と "]" がシェルのメタキャラクターとして解釈されます。したがって、どのPBSコマンドでも、ジョブアレイ名とサブジョブ名を二重引用符で囲む必要があります。

**例**：

```
qdel "1234[5].myhost"
qdel "1234[].myhost"
```

シェル変数置換を使用している場合を除き、一重引用符も使用できます。

## 8.3.5 ジョブアレイの特殊属性

ジョブアレイとサブジョブは、1つのジョブの属性をすべて有しています。また、適切である場合、以下の属性を有します。これらの属性は、読み取り専用です。

**表8-1：ジョブアレイの属性**

| 名前 | 種類 | 適用の対象 | 値 |
|---|---|---|---|
| Array | Boolean | ジョブアレイ | アイテムがジョブアレイである場合、*true*に設定されます。 |
| array_id | 文字列 | サブジョブ | サブジョブのジョブアレイ識別子 |
| array_index | 文字列 | サブジョブ | サブジョブのインデックス番号 |
| array_indices_remaining | 文字列 | ジョブアレイ | キューで待機中のサブジョブのインデックスをリストします。範囲または範囲の一覧、たとえば*500*、*552*、*596-1000* |
| array_indices_submitted | 文字列 | ジョブアレイ | 投入時刻におけるサブジョブのインデックスの完全な一覧。範囲、たとえば*1-100*のように与えられます。 |
| array_state_count | 文字列 | ジョブアレイ | サーバーおよびキューオブジェクトのstate_count属性と同等。各状態にあるサブジョブの数をリストします。 |
| max_run_subjobs | 整数 | ジョブアレイ | 一度に実行できるサブジョブの上限数。 |

## 8.3.6 ジョブアレイの状態

同じジョブアレイ内のサブジョブの状態は、異なる可能性があります。『PBS Professional Reference Guide』の「Job Array States」（363ページ）と『PBS Professional Reference Guide』の「Subjob States」（363ページ）をご参照ください。

### 8.3.7 ジョブアレイのPBS環境変数

**表8-2：ジョブアレイのPBS環境変数**

| 環境変数名 | 使用対象 | 説明 |
|---|---|---|
| PBS_ARRAY_INDEX | サブジョブ | ジョブアレイ内のサブジョブインデックス、たとえば7 |
| PBS_ARRAY_ID | サブジョブ | ジョブアレイの識別子。ジョブアレイのシーケンス番号、たとえば1234[].myserver |
| PBS_JOBID | ジョブ、サブジョブ | ジョブまたはジョブアレイの識別子。サブジョブについては、シーケンス番号と括弧[]に囲まれたサブジョブインデックス、たとえば1234[7].myserver |

### 8.3.8 アカウンティング

ジョブアレイおよびサブジョブについてのジョブのアカウンティング記録は、ジョブについてのそれと同じです。ジョブアレイが１つのサーバーから別のサーバーに移動された際、サブジョブのアカウンティング記録は２つのサーバー間で分割されます。

サブジョブには、“Q”レコードがありません。

### 8.3.9 prologueとepilogue

prologueとepilogueが定義されている場合、それらは各サブジョブの開始時と終了時に実行されますが、アレイオブジェクトについては実行されません。

### 8.3.10 “再実行可能”フラグとジョブアレイ

ジョブアレイは再実行可能にする必要があります。PBSでは、再実行不可能とマーキングされているジョブアレイは受け付けません。再実行可能かどうかを指定せずにジョブアレイを投入することはできますが、その場合は、PBSによって自動的に再実行可能とマーキングされます。

## 8.4 ジョブアレイの投入

### 8.4.1 ジョブアレイ投入構文

ジョブアレイは１つのコマンドで投入します。投入の際には、サブジョブインデックスを指定するほか、一度に実行できるサブジョブの上限数を必要に応じて指定します。

以下のいずれかの方法でインデックスの範囲を指定できます。

- 連続する範囲、たとえば1から100まで
- ステッピング係数を持つ範囲、たとえば、1から100までの１つおきのエントリ（1、3、5、...99）

サブジョブ上限数は、パーセント記号（%）に続けて整数で指定します。

ジョブアレイを投入する構文は以下のとおりです。

*qsub -J <開始インデックス>-<終了インデックス>[:<ステッピング係数>] [%<サブジョブ上限数>]*

パラメータの意味

*開始インデックス*：範囲内の最小インデックス番号

*終了インデックス*：範囲内の最小インデックス番号

*ステッピング係数*：インデックス番号の間隔（オプション）

*サブジョブ上限数*：一度に実行できるサブジョブの上限数

開始インデックスと終了インデックスは整数、ステッピング係数は正の整数でなくてはなりません。終了インデックスは開始インデックスより大きい値である必要があります。終了インデックスが開始インデックスにステッピング係数の倍数を加算した値にならない場合、終了インデックスはインデックス値としては使用されません。終了インデックスより小さい最も大きいインデックス値が使用されます。たとえば、開始インデックスが1、終了インデックスが8、ステッピング係数が3である場合、インデックス値は1、4、7になります。

#### 8.4.1.1 同時に実行するサブジョブの上限数

デフォルトでは、ジョブアレイにあるサブジョブのうち、同時に実行できる最大数のサブジョブが一度に実行されます。ジョブ属性max_run_subjobsの値を設定することで、一度に実行するサブジョブの数を制限できます。たとえば、同じ共有データファイルに各サブジョブがアクセスする必要があるとします。このアクセスのボトルネックに起因する実行速度の低下を防止する場合に、この上限が効果的です。次のように、-Jオプションに%<サブジョブ上限数>を付加することで、この上限を投入時に設定できます。

*qsub -J <開始インデックス>-<終了インデックス>[:<ステッピング係数>] [%<サブジョブ上限数>]*

以下に例を示します。

```
qsub -J 1-20000 %500 myscript.sh
```

次のように、qalterを使用してmax_run_subjobs属性値を設定または変更することもできます。

*qalter -Wmax_run_subjobs=<新しい値> <ジョブID>*

以下に例を示します。

```
qalter -Wmax_run_subjobs=1000 123[].myserver
```

保留中のサブジョブは、max_run_subjobsで設定した上限には算入されません。

### 8.4.2 ジョブアレイの投入の例

例8-1： インデックス1、2、3、...10000を有する10,000個のサブジョブのジョブアレイを投入するには、

```
$ qsub -J 1-10000 job.scr
1234[].server.domain.com
```

例8-2： インデックス500、501、502、...1000を有する500個のサブジョブのジョブアレイを投入するには、

```
$ qsub -J 500-1000 job.scr
1235[].server.domain.com
```

例8-3： インデックス1、3、5、...999を有するジョブアレイを投入するには、

```
$ qsub -J 1-1000:2 job.scr
1236[].server.domain.com
```

例8-4： 1、2、3・・・10000のインデックスを持つ10,000件のサブジョブのジョブアレイを投入し、同時に実行するサブジョブを500件に制限するには次のように指定します。

```
$ qsub -J 1-10000 %500 job.scr
1237[].server.domain.com
```

## 8.4.3 ジョブアレイのためのファイルステージング

ジョブアレイに対してステージングするファイルを準備する場合、ファイルの名前がサブジョブのインデックス番号と一致するように計画します。たとえば、inputfile3は、インデックス値が*3*のサブジョブが使用することを意味します。

ジョブアレイのためにファイルをステージングする場合、通常のジョブと同じメカニズムを使用しますが、サブジョブのインデックスを指定する変数を追加します。この変数の名前は*array_index*です。

### 8.4.3.1 ジョブアレイのためのファイルステージング構文

ジョブの実行前にステージインするファイルおよびジョブの実行後にステージアウトするファイルを指定できます。フォーマット：

*qsub -W stagein=<ステージインファイルリスト> -W stageout=<ステージアウトファイルリスト>*

これらは、qsubのオプションとして、またはジョブスクリプト内の指示文として使用できます。

stageinとstageoutのどちらの*ファイルリスト*も以下の書式で指定します。

*<実行パス>^array_index^@<ストレージホスト>:<ストレージパス>^array_index^[,...]*

*<実行パス><インデックス番号>*は、（実行ホスト上の）ジョブのステージングと実行のディレクトリにあるファイルの名前です。*実行パス*は、ジョブのステージングと実行のディレクトリを基準とした相対パスまたは絶対パスで指定できます。

文字'@'で実行の指定と保存の指定を区切ります。

*<ストレージパス><インデックス番号>*は、*ストレージホスト*で指定されたホスト上のファイル名です。ステージインの場合は、入力ファイルの取得元の場所を示します。ステージアウトの場合は、ジョブの終了時における出力ファイルの格納先です。*ストレージホスト*を指定する必要があります。この名前は、絶対パスまたはストレージマシン上にあるユーザーのホームディレクトリへの相対パスで表すことができます。

ステージインの場合、方向は*ストレージパス***から***実行パス***に**なります。

ステージアウトの場合、方向は*実行パス***から***ストレージパス***に**なります。

複数のファイル名セットをステージングする場合は、ファイル名セットをコンマで区切り、リスト全体を二重引用符で囲みます。

### 8.4.3.2 Windowsでのジョブアレイのステージング構文

Windowsでは、ステージインおよびステージアウトストリングは、^array_index^を使用している際には、二重引用符に囲まれていなくてはなりません。

ステージインの例：

```
qsub -W stagein="foo.^array_index^@host-1:C:¥WINNT¥Temp¥foo.^array_index^" -J 1-5 stage_script
```

ステージアウトの例：

```
qsub -W stageut="C:¥WINNT¥Temp¥foo.^array_index^@host-1:Q:¥my_username¥foo.^array_index^_out" -J
    1-5 stage_script
```

### 8.4.3.3 ジョブアレイのファイルステージングの注意事項

*ストレージパス*には、絶対パス名を使用することをお勧めします。ユーザーのホームディレクトリへのパスはマシンごとに異なる場合があります。また、sandbox = *PRIVATE*を使用している場合は、必ずしもすべての実行マシン上にホームディレクトリを置く必要はありません。

### 8.4.3.4 ジョブアレイのステージングの例

例8-5： 単純な例：

ストレージパス：store:/film

入力として使用されるデータファイル：frame1、frame2、frame3

実行パス：pix

実行可能ファイル：a.out

この例では、a.outは、frame2からframe2.outを生成します。

```
#PBS -W stagein=pix/in/frame^array_index^@store:/film/frame^array_index^
#PBS- W stageout=pix/out/frame^array_index^.out @store:/film/frame^array_index^.out
#PBS -J 1-3 a.out frame$PBS_ARRAY_INDEX ./in ./out
```

stageoutステートメントはすべて１行で記述されます。

結果として、“film”という名前のユーザーのディレクトリにはオリジナルファイルframe1、frame2、frame3および新しいファイルframe1.out、frame2.out、frame3.outが含まれます。

例8-6： この例では、scriptlet1とscriptlet2を呼び出すArrayScriptという名前のスクリプトを使用します。

３つのスクリプトはいずれも/homedir/testdirにあります。

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-2
echo "Main script: index " $PBS_ARRAY_INDEX
/homedir/testdir/scriptlet$PBS_ARRAY_INDEX
```

この例では、scriptlet1とscriptlet2は単にそれらの名称をエコーしています。ArrayScriptはqsubコマンドを使って実行します。

```
qsub ArrayScript
```

例8-7： この例では、StageScriptという名前のスクリプトを使用します。このスクリプトでは２つの入力ファイルdataXおよびextraXを取得し、出力ファイルnewdataXを生成して、どの繰り返し計算がオンになっているかをエコーします。dataXおよびextraXファイルはinputsからworkにステージングされ、newdataXはworkからoutputsにステージングされます。

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="/homedir/work/data^array_index^@host1:/homedir/inputs/data^array_index^, ¥
    /homedir/work/extra^array_index^ @host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=/homedir/work/newdata^array_index^@host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cd /homedir/work
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX >> newdata$PBS_ARRAY_INDEX
```

実行パス：

```
/homedir/work
```

ストレージホスト：

```
host1
```

入力用ストレージパス（オリジナルデータファイルdataXおよびextraX）：

```
/homedir/inputs
```

結果用ストレージパス（計算newdataXの出力）：

```
/homedir/outputs
```

StageScriptは/homedir/testdirに格納されます。そのディレクトリ内で、以下のように入力して実行することができます。

```
qsub StageScript
```

これはホームディレクトリの/homedirで実行されます。そのため、以下の行

```
"cd /homedir/work"
```

がスクリプトに記述されています。

例8-8： この例では、上記と同じスクリプトを使用していますが、実行元はPBSによって作成されたステージングおよび実行ディレクトリです。StageScriptでは２つの入力ファイルdataXおよびextraXを取得し、出力ファイルnewdataXを生成して、どの繰り返し計算がオンとなっているかをエコーします。dataXおよびextraXファイルはinputsからステージングおよび実行ディレクトリにステージングされ、newdataXはステージングおよび実行ディレクトリからoutputsにステージングされます。

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="data^array_index^@host1:/homedir/inputs/data^array_index^, ¥
    extra^array_index^@host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=newdata^array_index^@host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX >> newdata$PBS_ARRAY_INDEX
```

実行パス（ディレクトリ）：PBSによって作成（名前は不明）

ストレージホスト：

```
host1
```

入力用ストレージパス（オリジナルデータファイルdataXおよびextraX）：

```
/homedir/inputs
```

結果用ストレージパス（計算newdataXの出力）：

```
/homedir/outputs
```

StageScriptは/homedir/testdirに格納されます。そのディレクトリ内で、以下のように入力して実行することができます。

```
qsub StageScript
```

スクリプトは、PBSによって作成されたステージングおよび実行ディレクトリ内で実行されます。35ページの第3.2節「入力 / 出力ファイルのステージング」をご参照ください。

## 8.4.4 標準出力と標準エラーのファイル名

サブジョブのstdoutの名前はデフォルトで<*ジョブ名*>.o<*シーケンス番号*>.<*インデックス*>になります。サブジョブのstderrの名前はデフォルトで<*ジョブ名*>.e<*シーケンス番号*>.<*インデックス*>になります。

例8-9： ジョブの名前が“fixgamma”、シーケンス番号が“1234”の場合。

インデックス７のサブジョブは、1234[7].<サーバー名>になります。このサブジョブのstdoutの名前はfixgamma.o1234.7、stderrの名前はfixgamma.e1234.7になります。

## 8.4.5 ジョブアレイの依存関係

ジョブの依存関係は、次の関係でサポートされます。

- ジョブアレイとジョブアレイの間
- ジョブアレイとジョブの間
- ジョブとジョブアレイの間

### 8.4.5.1 ジョブアレイの依存関係の注意事項

ジョブの依存関係は、サブジョブまたはサブジョブの範囲についてはサポートされていません。

## 8.4.6 ジョブアレイの終了ステータス

ジョブアレイの終了ステータスは、完了した各サブジョブの状態によって決定されます。すべての有効なサブジョブが完了した際にのみ使用できます。完了したサブジョブの個々の終了ステータスは epilogue に渡され、そのサブジョブの 'E' アカウンティングログ記録内で参照できます。

**表8-3：ジョブアレイの終了ステータス**

| 終了ステータス | 意味 |
|---|---|
| 0 | ジョブアレイのすべてのサブジョブが、終了ステータス0を返しました。PBSエラーの発生はありません。削除されたサブジョブは考慮されていません。 |
| 1 | 少なくとも1つのサブジョブが、0以外の終了ステータスを返しました。PBSエラーの発生はありません。 |
| 2 | PBSエラーが発生しました。 |

### 8.4.6.1 ジョブアレイの終了まで qsub を待機させる

qsub コマンドをブロックすると、ジョブアレイ全体が完了するまで待機したうえで、ジョブアレイの終了ステータスを戻します。

### 8.4.6.2 ジョブアレイの終了ステータスの注意事項

サブジョブの終了ステータスは、サブジョブがジョブ履歴に存在する場合にのみ使用できます。サブジョブがジョブ履歴に存在しない場合、失敗したサブジョブまたは終了したサブジョブは failed や terminated ではなく、*Finished* の終了ステータスが表示されます。

## 8.4.7 ジョブアレイの投入の注意事項

### 8.4.7.1 ジョブアレイのインタラクティブジョブの投入の禁止

ジョブアレイのインタラクティブ投入は許可されません。

# 8.5 ジョブアレイのステータスの表示

qstatコマンドを使用して、ジョブアレイのステータスを照会できます。デフォルトの出力としては、ジョブアレイを１つの行にリストし、ジョブアレイ識別子を表示します。オプションは組み合わせて使用できます。

qstatコマンドで-fオプションを使用すると、サブジョブのすべての属性を表示できます。

実行中のすべてのサブジョブの状態を表示するには、-t -rを使用します。サブジョブのみの状態を表示するには、-t -Jを使用します。

**表8-4：qstatのジョブアレイとサブジョブのオプション**

| オプション | 結果 |
|---|---|
| -t | ジョブアレイオブジェクトおよびサブジョブの状態を表示します。<br>また、ジョブの状態も表示します。 |
| -J | ジョブアレイのみの状態を表示します。 |
| -p | デフォルトの表示をPercentage Completedのカラム付で出力します。<br>ジョブアレイについては、これは、完了もしくは削除されたサブジョブの数をサブジョブの総数で割ったものです。ジョブについては、使用された時間を要求された時間で割ったものです。 |

## 8.5.1 ジョブアレイのステータス表示の例

ジョブとジョブアレイの実行例は、２つのプロセッサを備えたマシン上で実行します。

demoscript:

```
#!/bin/sh
#PBS -N JobExample
sleep 60
```

arrayscript:

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-5
sleep 60
```

これらのスクリプトをqsubコマンドを使って実行します。

```
qsub arrayscript
1235[].host
qsub demoscript
1236.host
```

qstatのさまざまなオプションを使用して照会します。

**qstat**

```
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
1236.host   JobExample   user1              0 Q workq
```

**qstat -J**

```
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
```

**qstat -p**

```
Job id      Name         User        % done  S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1            0  B workq
1236.host   JobExample   user1           --  Q workq
```

**qstat -t**

```
Job id       Name         User        Time Use S Queue
-----------  ------------ ----------  -------- - -----
1235[].host  ArrayExample user1              0 B workq
1235[1].host ArrayExample user1       00:00:00 R workq
1235[2].host ArrayExample user1       00:00:00 R workq
1235[3].host ArrayExample user1              0 Q workq
1235[4].host ArrayExample user1              0 Q workq
1235[5].host ArrayExample user1              0 Q workq
1236.host    JobExample   user1              0 Q workq
```

**qstat -Jt**

```
Job id       Name         User   Time Use S Queue
------------ ------------ -----  -------- - -----
1235[1].host ArrayExample user1  00:00:00 R workq
1235[2].host ArrayExample user1  00:00:00 R workq
1235[3].host ArrayExample user1         0 Q workq
1235[4].host ArrayExample user1         0 Q workq
1235[5].host ArrayExample user1         0 Q workq
```

最初の２つのサブジョブが終了した後は、

```
qstat -Jtp
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
```

```
qstat -pt
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[].host  ArrayExample user1      40 B workq
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
1236.host    JobExample   user1      -- Q workq
```

ここで、最後のサブジョブのみが実行中になるまで待機すると、

```
qstat -rt
                                                  Req'd  Req'd  Elap
Job ID     Username Queue Jobname   SessID NDS TSK Memory Time S Time
---------- ------  ----- ---------  ---- --- --- ------ ---- - -----
1235[5].host user1  workq ArrayExamp 3048  --   1    --    --  R 00:00
1236.host    user1  workq JobExample 3042  --   1    --    --  R 00:00
```

```
qstat -Jrt
                                                  Req'd  Req'd  Elap
Job ID      Username Queue Jobname  SessID NDS TSK Memory Time S Time
----------- -------- ----- --------- ---- --- --- ------ ---- - -----
1235[5].host   user1 workq ArrayExamp 048 --   1     --   --  R 00:01
```

# 8.6 ジョブアレイに対するPBSコマンドの使用

以下の表に、ジョブアレイ、サブジョブ、または範囲に対するPBSコマンドの使用可否を示します。

**表8-5：ジョブアレイに対するPBSコマンドの使用**

| | コマンドの引数 | | |
|---|---|---|---|
| コマンド | Array[]：アレイオブジェクト | Array[Range]：指定された範囲のサブジョブ | Array[Index]：指定されたサブジョブ |
| qalter | アレイオブジェクト | erroneous | erroneous |
| qdel | アレイオブジェクトと実行中のサブジョブ | 指定された範囲の実行中のサブジョブ | 指定されたサブジョブ |
| qhold | アレイオブジェクトとキューイングされたサブジョブ | erroneous | erroneous |
| qmove | アレイオブジェクトとキューイングされたサブジョブ | erroneous | erroneous |
| qmsg | erroneous | erroneous | erroneous |
| qorder | アレイオブジェクト | erroneous | erroneous |
| qrerun | 実行中および終了したサブジョブ | 指定された範囲の実行中のサブジョブ | 指定されたサブジョブ |
| qrls | アレイオブジェクトとキューイングされたサブジョブ | erroneous | erroneous |
| qsig | 実行中のサブジョブ | 指定された範囲の実行中のサブジョブ | 指定されたサブジョブ |
| qstat | アレイオブジェクト | 指定された範囲のサブジョブ | 指定されたサブジョブ |
| tracejob | erroneous | erroneous | 指定されたサブジョブ |

## 8.6.1 ジョブアレイの削除

qdelコマンドは、ジョブアレイ識別子、サブジョブ識別子、またはジョブアレイ範囲を指定します。指定されたオブジェクトは、その時点で実行中のサブジョブも含め、削除されます。実行中のサブジョブは、実行中のジョブと同様に扱われます。実行中ではないサブジョブは削除され、実行されません。

デフォルトでは削除されたサブジョブごとに電子メールが1通送信されるため、5000個のサブジョブからなるジョブアレイを1つ削除すると、送信される電子メールの数を抑制していない限り、5000通の電子メールが送信されます。『PBS Professional Reference Guide』の「-Wsuppress_email=<N>」（144ページ）をご参照ください。

## 8.6.2 ジョブアレイの変更

qalterコマンドはジョブアレイオブジェクトのみに使用でき、サブジョブまたは範囲には使用できません。ジョブアレイの属性は、ジョブの属性と同じです。

max_run_subjobs属性を変更するには`qalter -Wmax_run_subjobs=<新しい値> <ジョブID>`を使用します。

## 8.6.3 ジョブアレイの移動

qmoveコマンドはジョブアレイオブジェクトのみに使用でき、サブジョブまたは範囲には使用できません。ジョブアレイは、“*Q*”、“*H*”、または“*W*”状態にあり、実行中のサブジョブが存在しない場合に限って、1つのサーバーから別のサーバーへ移動できます。ジョブアレイオブジェクトの状態は、移動中保持されます。ジョブアレイは、新しいサーバー上で完了するまで実行されます。

ジョブアレイがそこから移動されたサーバー上のジョブに使用したqstatは、ジョブアレイを示しません。ジョブアレイオブジェクトに使用したqstatは、新しいサーバーにリダイレクトされます。

## 8.6.4 ジョブアレイの保留

qholdコマンドはジョブアレイオブジェクトのみに使用でき、サブジョブまたは範囲には使用できません。保留は‘*Q*’、‘*B*’、または‘*W*’状態からのみ、ジョブアレイに適用することが可能です。これにより、ジョブアレイは‘*H*’（保留）状態に置かれます。サブジョブが実行中である場合、それらは完了するまで実行を継続します。‘*H*’（保留）状態にあるキュー内のサブジョブは開始されません。

ジョブアレイにシステム保留状態のサブジョブがある場合、ジョブアレイもシステム保留になります。

## 8.6.5 ジョブアレイの保留解除

qrlsコマンドを直接使用できるのはジョブアレイオブジェクトのみで、サブジョブまたは範囲には使用できません。ジョブアレイが‘*Q*’または‘*B*’状態にあった場合、その状態に戻されます。ジョブアレイが‘*W*’状態にあった場合、待機時間に達していない限り、その状態に戻されます。待機時間に達していた場合は‘*Q*’状態に進みます。

サブジョブでは間接的にqrlsを使用できます。ジョブアレイでqrlsを使用した場合、システム保留状態のサブジョブがあるためにジョブアレイがシステム保留状態であると、システム保留により保留されているサブジョブが解放されてから、ジョブアレイに対するシステム保留が解除されます（これには、マネージャ、root、またはPBS管理者権限が必要です）。

## 8.6.6 ジョブアレイの選択

qselectのデフォルトの挙動としては、サブジョブ識別子を返さずにジョブアレイ識別子を返します。

状態選択（-s）オプションを使用して‘*R*’、‘*S*’、‘*T*’、または‘*U*’への設定を制限している場合、ジョブアレイはこれらの状態になり得ないため、qselectはジョブアレイを返しません。ただし、qselectの-tオプションを使って、サブジョブのリストを返すことが可能です。

qselectのオプションは組み合わせて使用できます。たとえば、選択をサブジョブに制限するには、-Jオプションと-Tオプションの両方を使用します。実行中のサブジョブのみを選択するには、-J -T -sRを使用します。

**表8-6：ジョブアレイについてのqselectコマンドのオプション**

| オプション | 選択の対象 | 結果 |
|---|---|---|
| （なし） | ジョブ、ジョブアレイ | ジョブおよびジョブアレイ識別子を表示します。 |
| -J | ジョブアレイ | ジョブアレイ識別子のみを表示します。 |
| -T | ジョブ、サブジョブ | ジョブおよびサブジョブ識別子を表示します。 |

### 8.6.7 キュー内のジョブアレイの順序付け

qorderコマンドはジョブアレイオブジェクトのみに使用でき、サブジョブまたは範囲には使用できません。これは、キュー内の他のジョブまたはジョブアレイに関連してジョブアレイのキュー順序を変更します。

### 8.6.8 ジョブアレイの再キューイング

qrerunコマンドは、ジョブアレイ識別子、サブジョブ識別子、またはジョブアレイ範囲を指定します。ジョブアレイ識別子が引数として与えられている場合、投入時刻にその初期状態に戻されるか、もしくは変更されている場合はその変更された状態に戻されます。その時点で実行中のもの、および処理を完了して削除されたものも含め、ジョブアレイのサブジョブがすべて再キューイングされます。サブジョブまたは範囲が指定されている場合、それらのサブジョブがジョブと同様に再キューイングされます。

### 8.6.9 ジョブアレイの信号送出

qsigにジョブアレイオブジェクト、サブジョブ、またはジョブアレイ範囲が渡されると、指定されたセット内にある実行中のすべてのサブジョブに信号が送出されます。

### 8.6.10 ジョブアレイへのメッセージの送付

qmsgコマンドは、ジョブアレイではサポートされていません。

### 8.6.11 ジョブアレイに関するログデータの取得

tracejobコマンドは、ジョブアレイと個々のサブジョブについて実行することが可能です。tracejobがジョブアレイまたはサブジョブについて実行される際、ジョブアレイに関する追加の情報に加え、ジョブについて実行される場合と同じ情報が表示されます。サブジョブは実行されるまで存在せず、したがって、tracejobは、それらが実行されるまで何の情報も表示しません。tracejobがジョブアレイについて実行される際、表示される情報は、そのジョブアレイオブジェクトについてのみで、サブジョブについては表示されません。ジョブアレイ自体は、いかなるMoMログ情報も生成しません。ジョブアレイについてtracejobを実行すると、なぜサブジョブが開始されなかったかの情報が得られます。

### 8.6.12 ジョブアレイに対するPBSコマンドの使用の注意事項

#### 8.6.12.1 ジョブアレイに対するシェルとPBSコマンド

cshやtcshなどの一部のシェルは、メタキャラクターとして大括弧（“[”、“]”）を使用します。これらのシェルを使用し、さらにPBSコマンドでサブジョブ、ジョブアレイ、またはジョブアレイ範囲を引数として取得している場合、サブジョブ、ジョブアレイ、またはジョブアレイ範囲は二重引用符で囲まれている必要があります。

# 8.7 ジョブアレイの注意事項

## 8.7.1 ジョブアレイは再実行可能にする必要がある

ジョブアレイは再実行可能にする必要があり、デフォルトで再実行可能に設定されます。

## 8.7.2 すべてのサブジョブはリソースが同じであること

異なるハードウェア要件、すなわち-l selectが異なるジョブを結合してアレイを構成することはできません。

## 8.7.3 チェックポイント処理によるジョブアレイの未サポート

チェックポイント処理は、ジョブアレイについてはサポートされていません。チェックポイント処理をサポートしているシステム上であっても、サブジョブはチェックポイントされず、完了するまで実行が継続されます。

## 8.7.4 ジョブアレイの終了ステータスの注意事項

サブジョブの終了ステータスは、サブジョブがジョブ履歴に存在する場合にのみ使用できます。サブジョブがジョブ履歴に存在しない場合、失敗したサブジョブまたは終了したサブジョブはfailedやterminatedではなく、*Finished*の終了ステータスが表示されます。

# 9
# PBSジョブでの作業

## 9.1 ジョブ履歴の使用

PBS Professionalではジョブの履歴情報を入手できます。このジョブ履歴には、投入パラメータの種類、ジョブが実行を開始したかどうか、実行の成否、結果のステージングアウトの成否、および使用されたリソースに関する情報が含まれています。

PBSでは、実行を終了したジョブ、削除されたジョブ、または別のサーバーに移動されたジョブの履歴を保持できます。

### 9.1.1 用語の定義

**移動されたジョブ**
別のサーバーに移動されたジョブ

**終了したジョブ**
何らかの理由で実行が終了したジョブ：

- 実行が正常に終了したジョブ
- 実行中にPBSによって終了されたジョブ
- システムまたはネットワーク障害によって実行に失敗したジョブ
- 実行を開始する前に削除されたジョブ

### 9.1.2 ジョブの履歴情報

PBSでは、以下に示すような、すべてのジョブの属性情報を保持できます。

- 投入パラメータ
- ジョブの実行が開始されたかどうか
- 実行の成否
- 結果のステージングアウトの成否
- 使用されたリソース

PBSでは以下のジョブのジョブ履歴が保持されます：

- 別のサーバーで実行されているジョブ
- 実行が終了したジョブ
- 削除されたジョブ
- 別のサーバーに移動されたジョブ

ジョブの実行中は、そのジョブに関する情報を確認できます。終了したジョブまたは削除されたジョブのジョブ履歴情報は、指定された期間保持されます。管理者は、各ジョブの終了後、または削除後のジョブ履歴情報の保持期間を選択します。PBSは定期的に終了した各ジョブをチェックし、指定された期間を超えて保持されているジョブ履歴を削除します。

サブジョブは、親のアレイジョブが終了するまで、つまりアレイジョブの配下のサブジョブすべてが実行を終了するまでは、終了したジョブとはみなされません。

### 9.1.2.1 移動されたジョブの操作

移動されたジョブには次のコマンドを使用できます。これらのコマンドは、通常のジョブに対してと同じように機能します。

qalter

qhold

qmove

qmsg

qorder

qrerun

qrls

qrun

qsig

移動されたジョブが実行されている間、その状態は *M* です。移動されたジョブが完了すると、そのサブステータスは *92* になります。『PBS Professional Reference Guide』の「Job States」（361 ページ）をご参照ください。

### 9.1.2.2 PBSコマンドと終了したジョブ

上記のコマンドは、ジョブの終了場所（ローカルサーバーまたはリモートサーバー）にかかわらず、終了したジョブには使用できません。これらのジョブはもう実行されていません。PBSではこれらのジョブの情報を保存しており、その情報を変更することはできません。終了したジョブに対して上記のコマンドの使用を試みると、以下のエラーメッセージが表示されます。

<コマンド名>: Job <ジョブID> has finished

## 9.2 ジョブ属性の修正

ジョブがキューで待機中の場合、ジョブのオーナー（またはマネージャもしくはオペレータ）はほとんどの属性を変更することができます。ただし、ジョブの実行が開始されると、cputime、walltime、およびrun_countの値しか修正できません。walltimeを小さくし、run_countを大きくすることができます。

qalter -lオプションを使用して、キューイングしているジョブのリソースリストを変更する場合は、-l selectの変更とジョブの制限の相互関係について理解しておく必要があります。

ジョブが明示的に"-l select="を指定して投入された場合、vnodeレベルのリソースは"-l select="形式を使用してジョブの属性を変更する必要があります。この場合、vnodeレベルのリソースであるRESの属性は、“-l <resource>”形式では変更できません。

以下に例を示します。

ジョブを投入：

```
% qsub -l select=1:ncpus=2:mem=512mb jobscript
```

ジョブIDは230

"-l RES"形式を使用したジョブの属性の変更（qalter）：

```
% qalter -l ncpus=4 230
```

qalterから報告されるエラー：

qalter：Resource must only appear in "select"

specification when select is used: ncpus 230

"-l select="形式を使用したジョブの属性の変更（qalter）：

**% qalter -l select=1:ncpus=4:mem=512mb 230**

qalterによるエラーなし：

%

## 9.2.1 selection指示文の変更

selection指示文が変更されると、指示文内にある消費可能なリソースのジョブに対する制限も変わります。

たとえば、以下のリソースリストでジョブをキューイングする場合、

**select=2:ncpus=1:mem=5gb**

ジョブの制限はncpus=2およびmem=10gbに設定されます。

-l selectの要求を以下のように変更すると

**select=3:ncpus=2:mem=6gb**

ジョブの制限はncpus=6、mem=18gbに再設定されます。

## 9.2.2 ジョブワイドの制限の変更

ジョブワイドの制限が修正されても、selection指示文内の対応するリソースは変更されません。複合的な指示文の変更をどこに適用するかを決めるのは簡単ではありません。

ジョブワイドの制限を縮小し、指示文のリソース合計より少ない値に新たに設定することはお勧めできません。実行時に制限を超過するとジョブが中止される状況が発生します。こうした修正に実効性があるとはいえません。

ジョブのwalltimeはいつでも変更できますが、ジョブがその初期値にかかわらず*Exiting*状態の場合は変更できません。

ジョブをキューイングしている場合、要求する修正内容は、キューおよびサーバーに対するジョブのリソースリミット内にも収まるように調整する必要があります。リソースに要求する修正内容がキューまたはサーバーに対するジョブのリソースリミットを超過すると、リソース要求が拒否されます。

リソースは、-l select内のチャンクで-lオプションを使用して変更します。または、ジョブワイドの修正の場合は、resource_name=valueの組み合わせを使用します。-l selectの形式は次のとおりです。

*-l select=[N:]chunk[+[N:]chunk ...]*

ここで、Nは該当するチャンクの数を示します。チャンクは次のような形式になります。

*<リソース名>=<値>[:<リソース名>=<値> ...]*

ジョブワイドの*<リソース名>=<値>*の修正は次の形式で行います。

*-l <リソース名>=<値>[,<リソース名>=<値> ...]*

vnode上のジョブ配置は、placeステートメントを使用して変更します。

*-l place=<modifier>[:<modifier>]*

ここで、*modifier*は、*group*、*excl*、*exclhost*、または次のいずれか*free|pack|scatter|vscatter*（あるいはそのすべて）の任意の組み合わせです。

qalterの構文は次のとおりです。

*qalter <ジョブリソース> <ジョブリスト>*

次の例は、qalterコマンドの使用方法を示しています。まず、特定のユーザーのジョブをすべてリストします。次に示されているように2つの属性を変更します（経過時間を20分から25分に変更し、ジョブの名前を“airfoil”から“engine”に変更）。

```
qstat -u barry
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
51.south barry  workq airfoil  930  --   1  -- 0:16 R 0:01
54.south barry  workq airfoil  --   --   1  -- 0:20 Q  --

qalter -l walltime=20:00 -N engine 54
qstat -a 54
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq engine   --   --   1  -- 0:25 Q  --
```

qalterコマンドはジョブアレイに使用することができますが、サブジョブまたはサブジョブの範囲には使用できません。ジョブアレイで使用する場合、ジョブアレイ識別子を二重引用符で囲む必要があります。たとえば、以下のようになります。

```
qalter -l walltime=25:00 "1234[].south"
```

qalterコマンド（またはその他のコマンド）を使用しても、非表示または要求不可の設定で作成されたカスタムリソースは変更できません。62ページの第4.3.8節「リソース要求に関する注意事項と制限事項」をご参照ください。

詳細については、『PBS Professional Reference Guide』の「qalter」（130ページ）をご参照ください。

### 9.2.2.1 注意事項

ブーリアンリソースをジョブワイドの制限として使用する場合は注意してください。

## 9.3 ジョブの削除

PBSには、ジョブを削除するためのqdelコマンドがあります。qdelコマンドは、コマンド内に現れるジョブ識別子の順序に従ってジョブを削除します。バッチジョブを削除できるのは、ジョブのオーナー、PBSオペレータ、またはPBS管理者です。管理者やオペレータ以外のユーザーが削除できるのは、各自のジョブのみです。

キュー待機中、保留中、実行中、または中断中のジョブを削除するには、以下を実行します。

*qdel <ジョブID>*

例：

```
qdel 51
qdel 1234[].server
```

ジョブアレイの識別子を二重引用符で囲む必要があります。

### 9.3.1 Force指定によるジョブの削除

ジョブの削除は、その実行ホストが到達可能かどうか、およびジョブがプロビジョニング中かどうかに関係なく、実行できます。

*qdel -W force <ジョブID>*

### 9.3.2 終了したジョブの削除

デフォルトでは、qdelコマンドは終了したジョブには影響を与えません。qdel -xオプションを使用するとジョブの履歴を削除できます。また、このオプションを使用すると、指定されたあらゆるジョブ（キュー待機中、実行中、保留中、中断中、終了済み、移動済み）も削除されます。このオプションでは、ジョブとその履歴が一度に削除されます。-xオプションを指定しないでqdelコマンドを使用すると、ジョブは削除されますが、ジョブの履歴は削除されません。また、終了したジョブを削除することもできません。

終了したジョブを削除するには、それが移動されたかどうかに関係なく、以下を実行します。

*qdel -x <ジョブID>*

-xオプションを指定しないで終了したジョブを削除しようとすると、次のエラーが表示されます。

```
qdel : Job <ジョブID> has finished
```

### 9.3.3 移動されたジョブの削除

qdel -xオプションを使用すると、キュー待機中、実行中、保留中、中断中、終了済み、移動済みのジョブを削除できます。

移動されたジョブを削除するには、以下を実行します。

*qdel <ジョブID シーケンス番号>.<移動元サーバー>*

*移動されて終了したジョブを削除するには、以下を実行します。*

*qdel -x <ジョブID>*

### 9.3.4 電子メールの数の制限

メールはデフォルトでは削除したジョブまたはサブジョブごとに送信されます。qdelに次のオプションを使用して、送信される電子メールの制限数を指定します。

*qdel -Wsuppress_email=<N>*

29ページの第2.5.1.3節「ジョブ削除時の電子メールの数の制限」をご参照ください。

## 9.4 ジョブへのメッセージの送付

ジョブにメッセージを送付するということは、ジョブの１つまたは複数の出力ファイルにメッセージ文字列を書き出すということです。通常は、ジョブの出力に情報を示すメッセージを残すために行います。このようなメッセージは、qmsgコマンドを使用して書き出します。

メッセージは実行中のジョブにのみ送信できます。

qmsgコマンドの構文は次のとおりです。

*qmsg [ -E ][ -O ] <メッセージ文字列> <ジョブID>*

例：

```
qmsg -O "output file message" 54
qmsg -O "output file message" "1234[].server"
```

ジョブアレイの識別子を二重引用符で囲む必要があります。

-Eオプションは、指定されたジョブのエラーファイルにメッセージを書き出します。-Oオプションは、指定されたジョブの出力ファイルにメッセージを書き出します。どちらのオプションも指定されていない場合は、メッセージはジョブのエラーファイルに書き出されます。

最初のオペランド*message_string*は、書き出されるメッセージです。文字列に空白が含まれている場合は、文字列を引用符で囲む必要があります。文字列の最終文字が改行コードでない場合は、ジョブのファイルに書き込むときに改行文字が追加されます。残りのオペランドはすべて、メッセージ文字列を受け取るジョブを指定するジョブIDです。以下に例を示します。

```
qmsg -E "hello to my error (.e) file" 55
qmsg -O "hello to my output (.o) file" 55
qmsg "this too will go to my error (.e) file" 55
```

# 9.5 ジョブへのシグナルの送信

qsigコマンドを使用してジョブにシグナルを送信できます。シグナルは、ジョブのすべてのプロセスへ送られます。

qsigコマンドの構文は次のとおりです。

*qsig [ -s <signal> ] <job ID>*

ジョブアレイの*job ID*を二重引用符で囲む必要があります。

-sオプションが指定されていない場合は、SIGTERMが送信されます。-sオプションが指定されている場合、これは、ジョブに送信する*signal*を宣言します。*signal*引数は、SIGKILLのようなシグナル名、*KILL*のように*SIG*接頭部を含まないシグナル名、*9*のように符号なしのシグナル番号のいずれかです。シグナル名SIGNULLも許可されており、この場合サーバーはジョブに0を送信しますが、何も効果はありません。すべてのシグナル名がqsigで認識されるわけではありません。シグナル名が認識されない場合は、代わりにシグナル番号を使用して再度お試しください。バッチジョブへシグナルを送信する要求は、次の場合は拒否されます。

- ユーザーがジョブへシグナルを送信する権限を持っていない
- ジョブが実行状態にない
- 要求されたシグナルが実行ホストでサポートされていない
- ジョブが終了している
- ジョブがプロビジョニング中である

2つの特殊なシグナル名“suspend”および“resume”（すべて小文字であることに注意）は、ジョブの中断と再開に使用されます。中断された場合、ジョブはそのままシステムリソースを占有しますが、実行はされず、経過時間も発生しません。ジョブを中断または再開するには、マネージャまたはオペレータ権限が必要です。

TERMは便利なシグナルです。このシグナルは、シェルには無視されますが、トラップしてステータスの書き出しなどの便利な処理を実行できます。

次の３つの例はすべて、シグナル9（SIGKILL）をジョブ34に送信します。

```
qsig -s SIGKILL 34
qsig -s KILL 34
```

ジョブスクリプトでシグナルをトラップする場合、ジョブのすべてのシェルでトラップする必要があります。

ほとんどのLinuxシステムで、コマンド“kill -l”（つまり“マイナスエル”）は、使用可能なすべてのシグナルをリストします。

# 9.6 ジョブの順序の変更

PBSには、キュー内またはキュー間で２つのジョブの順序を変更するためのqorderコマンドがあります。2つのジョブの順序を変えるということは、ジョブが属するキュー内、または複数のキュー間での位置を入れ替えるということです'job1がキューAの位置3にあり、job2がキューBの位置4にある場合、qorderコマンドによりjob1がキューBの位置4になり、job2がキューAの位置3になります。

ジョブの属性（優先度など）は一切変更されません。キュー内の順序を変更したことによる影響は、ローカルジョブのスケジュールポリシーに依存します。詳細については、システム管理者にお問い合わせください。

qorderコマンドの構文は次のとおりです。

*qorder < ジョブID>1 < ジョブID2>*

ジョブアレイの識別子を二重引用符で囲む必要があります。

オペランドは両方とも、順序を入れ替えるジョブを指定する*ジョブID*です。

```
qstat -u bob
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south bob    workq twinkie  --   --   1  -- 0:20 Q  --
63[].south bob  workq airfoil  --   --   1  -- 0:13 Q  --

qorder 54 "63[]"
qstat -u bob
                                                 Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
63[].south bob  workq airfoil  --   --   1  -- 0:13 Q  --
54.south bob    workq twinkie  --   --   1  -- 0:20 Q  --
```

## 9.6.1 制約

- ２つのジョブは、同じサーバー内に配置されており、両方のジョブのオーナーが同じである必要があります。ただし、PBSマネージャまたはオペレータは、任意のジョブを入れ替えることができます。
- 実行状態のジョブの順序を変更することはできません。
- qorderコマンドはジョブアレイ全体に使用することができますが、サブジョブまたは範囲には使用できません。ジョブアレイの順序を変更することで、キュー内の他のジョブまたはジョブアレイに関連してジョブアレイのキュー順序が変更されます。

# 9.7 キュー間でのジョブの移動

PBSには、異なるキュー（また異なるサーバー上のキュー）間でジョブを移動するためのqmoveコマンドがあります。ジョブを移動するということは、ジョブが属するキューからジョブを削除し、別のキューで具体化するということです。

実行状態のジョブを移動することはできません。

qmoveコマンドの構文は次のとおりです。

*qmove <destination> <job ID>*

ジョブアレイの*<job ID>*を二重引用符で囲む必要があります。

最初のオペランドは、以下に対する新規宛先です。

```
<queue>
@<server>
<queue>@<server>
```

*destination*オペランドで1つのキューしか記述されていない場合、qmoveによってジョブは、ジョブの現在のサーバーにある指定された名前のキューに移動されます。*destination*オペランドでサーバーのみが記述されている場合は、qmoveによってジョブは、そのサーバーの既定のキューに移動されます。*destination*オペランドにキューとサーバーの両方が記述されている場合は、qmoveによってジョブは、指定されたサーバーの指定されたキューに移動されます。残りのオペランドはすべて、新規*destination*に移動されるジョブを指定する*job ID*です。

qmoveコマンドはジョブアレイオブジェクトのみに使用でき、サブジョブまたは範囲には使用できません。ジョブアレイは、“*Q*”、“*H*”、または“*W*”状態にあり、実行中のサブジョブが存在しない場合に限って、1つのサーバーから別のサーバーへ移動できます。ジョブアレイオブジェクトの状態は、移動中保持されます。ジョブアレイは、新しいサーバー上で完了するまで実行されます。

ジョブアレイがそこから移動されたサーバー上のジョブに使用したqstatは、ジョブアレイを示しません。ジョブアレイオブジェクトに使用したqstatは、新しいサーバーにリダイレクトされます。

サブジョブのアカウンティング記録は、２つのサーバー間で分割されます。

# 10 ジョブ / システムステータスのチェック

## 10.1 検査するジョブの選択

複数のジョブの検査では、すべてを一度に確認できるほか、その一部を選択することもできます。そのように選択するには以下の手順に従います。

- 所定の基準に従い、qsig コマンドを使用してジョブを選択し、ジョブ ID のリストを取得します。これが qstat コマンドへの入力になります。183 ページの第 10.1.1 節「qselect によるジョブの選択」をご参照ください。
- 表示されるジョブを絞り込むオプションを使用して qstat コマンドを実行します。185 ページの第 10.1.2 節「qstat によるジョブのフィルタリング」をご参照ください。

### 10.1.1 qselect によるジョブの選択

qsig コマンドを使用して、選択基準に一致するジョブ、ジョブアレイ、またはサブジョブのジョブ識別子をリスト表示します。このコマンドを実行すると、選択したジョブのリストが標準出力に得られます。名前、優先順位、プロジェクト、状態などに応じてジョブを選択できます。本項では、ジョブを選択するいくつかの方法について説明します。

#### 10.1.1.1 リソースと属性値によるジョブの選択

属性値またはリソース値（あるいはその両方）が特定の値と等しい、等しくない、より大きい、以上である、未満である、以下であることを条件としてジョブを選択できます。デフォルトの関係は“等しいこと”で、“.eq.”で指定されています。

たとえば、次のように、16個を超えるCPUを要求したbarryが所有するジョブをリスト表示すると、該当するジョブがデフォルトのサーバー southに3件あることがわかります。

```
qselect -u barry -l ncpus.gt.16
121.south
133.south
154.south
```

### 10.1.1.2 時間条件によるジョブの選択

qselect -tオプションを使用すると、キュー待機中、実行中、終了済み、および移動済みのジョブ、ジョブアレイ、およびサブジョブをそれぞれの時間属性値に従ってリスト表示できます。-tオプションを2回使用すると、時間の範囲を指定できます。

例10-1： 終了時刻が正午から午後3時の範囲にあるジョブを選択するには、次のように指定します。

```
qselect -te.gt.09251200 -te.lt.09251500
```

例10-2： 終了したジョブまたは移動したジョブのうち、開始時刻が正午から午後3時の範囲にあるジョブを選択するには、次のように指定します。

```
qselect -x -s "MF" -ts.gt.09251200 -ts.lt.09251500
```

例10-3： 作成時刻が正午から午後3時の範囲にあるジョブをすべて選択するには、次のように指定します。

```
qselect -x -tc.gt.09251200 -tc.lt.09251500
```

例10-4： qtimeが午後2時半のすべてのジョブを、終了したジョブと移動したジョブも含めて選択するとします。ここでは、次のように関係を一切指定せずにデフォルトの関係である“.eq.”を使用します。

```
qselect -x -tq09251430
```

### 10.1.1.3 終了したジョブと移動したジョブの選択

終了したジョブと移動されたジョブの識別子は、ジョブ履歴が引き続き保持されている限り、キュー待機中および実行中のジョブと同じ方法でリスト表示できます。履歴の保持期間はPBS管理者が設定します。

qselectコマンドで-xオプションを使用すると、ジョブの状態（実行中、キュー待機中、終了済み、または移動済み）にかかわらず、すべてのジョブのジョブ識別子をリスト表示できます。qselectコマンドで-Hオプションを使用すると、終了したジョブまたは移動されたジョブについてのみジョブ識別子をリスト表示できます。

終了したジョブと移動したジョブのジョブIDをリスト表示するには次のように指定します。

```
qselect -H
```

### 10.1.1.4 選択したジョブのリストをqstatに引き渡す

選択したジョブのIDリストに関する情報を表示するには、qselectコマンドの出力をqstatコマンドの入力として使用します。構文：

*qstat [qstatのオプション] `qselect [qselectのオプション]`*

たとえば、16を超えるCPUを要求したbarryが所有する、キュー待機中のジョブと実行中のジョブをすべて表示するには次のように指定します（代替のフォーマットについては190ページの第10.2.1.2節「拡張したジョブリスト代替フォーマットのジョブステータス」をご参照ください）。

- Linux：

```
qstat -a  `qselect -u barry -l ncpus.gt.16 `
                                                  Req'd Req'd   Elap
Job ID    User  Queue Jobname Sess NDS TSK Mem    Time S Time
--------  ----- ----- ------- ---- --- --- ---    ---- - ----
54.south  barry workq airfoil  --   --   1  --    0:13 Q   --
121.south barry workq airfoil  --   --  32  --    0:01 H   --
133.south barry workq trialx   --   --  20  --    0:01 W   --
154.south barry workq airfoil 930   --  32  --    1:30 R 0:32
```

- Windows（cmdプロンプトで、次のように１行で入力します）：

```
for /F "usebackq" %j in (`qselect -u barry -l ncpus.gt.16`) do ( qstat -a %j )
54.south
121.south
133.south
154.south
```

### 10.1.1.5 終了したジョブと移動したジョブのリストをqstatに引き渡す

qstatを使用して、終了したジョブまたは移動したジョブ（過去のジョブ）のリストを検査するには、次のように、必ず-xオプションを指定したqstatと、-Hオプションを指定したqselectの両方を指示します。

```
qstat -x  `qselect -H [qselectのオプション]`
```

### 10.1.1.6 qselectでジョブを選択する際の制限事項と注意事項

- qselectコマンドの1回の呼び出しでジョブの選択元とすることができるサーバーは1つのみです。
- バッククォートを使用した構文で、qselectの出力をqstatの入力として使用する必要があります。パイプ記号を使用してqselectからqstatにパイプ出力することはできません。

## 10.1.2 qstatによるジョブのフィルタリング

qstatを使用してデフォルトで表示されるのはキュー待機中または実行中のジョブに関する情報です。終了したジョブや移動したジョブの情報は表示されず、ジョブアレイやサブジョブの情報も表示されません。ただし、実行中、キュー待機中、終了済み、または移動済みのすべてのジョブの情報を表示することをqstatに指示することもできます。終了したジョブと移動したジョブの履歴は、管理者が定義した期間保持されます。

ジョブ識別子の情報、ジョブ識別子リストの情報、または宛先（指定のキューやサーバーなど）にあるすべてのジョブの情報を返すことをqstatに指定できます。主に次の3種類のフォーマットでジョブ情報を入手できます：

- *デフォルトフォーマット*：ジョブIDごとに1行として、ジョブオーナーのユーザー名、ジョブの状態、そのキューなどの情報もその行に記述した基本的な表。190ページの第10.2.1.1節「基本的なジョブリスト：デフォルトフォーマットのジョブステータス」をご参照ください。
- *代替フォーマット*：ジョブIDごとに1行として、セッションID、要求時間、経過時間などもその行に記述した詳細な表。190ページの第10.2.1.2節「拡張したジョブリスト代替フォーマットのジョブステータス」をご参照ください。
- *詳細フォーマット*：一度に 1 つのジョブの情報を表示し、各ジョブ属性と各リソースを個別の行に記述したフォーマット。191ページの第10.2.1.3節「すべてのジョブ情報：詳細フォーマットのジョブステータス」をご参照ください。

### 10.1.2.1 ジョブIDリストの拡張とフィルタリング

qselectを使用してジョブIDを選択するほか、以下の基準も使用できます：

- ジョブアレイを表示（サブジョブは除く）：-J
- ジョブアレイとサブジョブを表示：-t
- サブジョブのみを表示：-Jt
- 終了したジョブと移動したジョブを代替フォーマットで表示：-H。188ページの第10.1.2.6.iii節「終了したジョブと移動したジョブに制限した検索」をご参照ください。
- 実行中のジョブとキュー待機中のジョブのほかに、終了したジョブと移動したジョブを表示：-x。188 ページの第10.1.2.6.ii節「終了したジョブと移動したジョブを含む検索」をご参照ください。

ジョブIDのフォーマット：

- ジョブID：

  *<シーケンス番号>[.<サーバー名>][@<サーバー名>]*
- ジョブアレイID：

  *<シーケンス番号>[][.<サーバー名>][@<サーバー名>]*
- サブジョブID：

  *<シーケンス番号>[<インデックス>][.<サーバー名>][@<サーバー名>]*
- サブジョブ:の範囲：

  *<シーケンス番号>[<開始インデックス>-<終了インデックス>][.<サーバー名>][@<サーバー名>]*

  シェルによっては、ジョブアレイ識別子を二重引用符で囲むことが必要なものもある点に注意します。

## 10.1.2.2 宛先の指定

宛先を指定しないと、すべてのキューとデフォルトのサーバーからジョブが得られます。キューまたはサーバー（あるいはその両方）を指定できます。宛先のフォーマットは次のとおりです：

- デフォルトのサーバーの指定したキューにあるすべてのジョブのステータスを表示：

  *<キュー名>*
- 指定したサーバーの指定したキューにあるすべてのジョブのステータスを表示：

  *<キュー名>@<サーバー名>*
- 指定したサーバーのすべてのキューにあるすべてのジョブのステータスを表示：

  *@<サーバー名>*

## 10.1.2.3 ジョブのユーザー別フィルタリング

qstatで“**-u**”オプションを指定すると、ユーザー名リストから指定したユーザーが所有するジョブが表示されます。構文：

*qstat -u <ユーザー名>[@<ホスト>][,<ユーザー名>[@<ホスト>],...]*

ホスト名は必須ではなく、左端にワイルドカードを使用できます。"**.mydomain.com*"。“*@<ホスト>*”を省略して“*<ユーザー名>*”とすると、“*<ユーザー名>@**”と同じ処理になります。

```
qstat -u user1
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --

qstat -u user1,barry
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
54.south barry  workq airfoil  --   --   1  -- 0:13 Q  --
```

### 10.1.2.4 実行中のジョブと保留中のジョブの検索

qstatで“-r”オプションを指定すると、実行中のジョブと保留中のジョブすべてのステータスが代替フォーマットで表示されます。以下に例を示します。

```
qstat -r
host1:
                                          Req'd  Req'd    Elap
Job ID   User  Queue Jobname Sess NDS TSK Mem    Time  S  Time
-------- ----- ----- ------- ---- --- --- -----  ----- -  -----
43.host1 user1 workq STDIN   4693   1   1   --     --  R  00:00
```

### 10.1.2.5 実行中ではないジョブの検索

qstatで“-i”オプションを指定すると、実行中ではないジョブ（キュー待機中、保留中、および待機中）すべてのステータスが代替フォーマットで表示されます。以下に例を示します。

```
qstat -i
host1:
                                            Req'd  Req'd     Elap
Job ID      User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
----------  ----- ----- ------- ---- --- ---  -----  -----  -  -----
44[].host1  user1 workq STDIN     --   1   1     --     --  Q     --
```

### 10.1.2.6 終了したジョブと移動したジョブ（ジョブの履歴）の検索

終了したジョブと移動されたジョブの情報は、PBSによりジョブ履歴が引き続き格納されている限り、キュー待機中および実行中のジョブと同じ方法で表示できます。

#### 10.1.2.6.i 別のサーバーに移動したジョブの検索

別のサーバーで実行しているジョブを検査できます。サイトでピアスケジューリングを使用している場合、ジョブがデフォルト以外のサーバーに移動されることがあります。たとえば、ServerAにジョブを投入し、ジョブIDとして“123.ServerA”が返されたとします。その後で、123.ServerAがServerBに移動されたとします。

- 実行中、キュー待機中、終了済み、移動済みに関係なく、すべてのジョブの情報を表示するには次のように指定します。

  **qstat -x**

- 特定のジョブを表示するには、次のようにそのジョブIDを引数としてqstatに渡します。

  **qstat 123**

  または

  **qstat 123.ServerA**

- ServerBのジョブをすべてリスト表示するには次のように指定します。

  **qstat @ServerB**

例10-5： 移動されたジョブの表示：

- ホスト名がそれぞれServerA、ServerB、およびServerCの3つのサーバーがあります。
- barryがジョブ123をServerAに投入します。
- しばらくたってから、そのジョブをbarryがServerBに移動します。
- さらにその後で、そのジョブを管理者がServerCのQueueCに移動します
- barryが“qstat 123”を実行します。

```
Job id            Name         User    Time Use  S   Queue
----------------  -----------  -----   -------  ---  ------
123.ServerA       STDIN        barry   00:00:00  M   QueueC@ServerC
```

### 10.1.2.6.ii 終了したジョブと移動したジョブを含む検索

qstatコマンドで-xオプションを指定すると、終了したジョブ、移動したジョブ、キュー待機中のジョブ、実行中のジョブをデフォルトフォーマットで検査できます。

- キュー待機中、実行中、終了済み、移動済みのジョブの情報をデフォルトフォーマットで表示するには次のように指定します。

  *qstat -x*
- ジョブの状態に関係なく、あるジョブの情報をデフォルトフォーマットで表示するには次のように指定します。

  *qstat -x <ジョブID>*
- キュー待機中、実行中、終了済み、移動済みのジョブ、ジョブアレイ、サブジョブのステータスを表示するには次のように指定します。

  *qstat -xt*
- キュー待機中、実行中、終了済み、または移動済みのジョブアレイのステータスを表示するには次のように指定します。

  *qstat -xJ*

移動されたジョブの情報が表示される際、宛先キューとサーバーは<queue>@<server>のように表示されます。

例10-6：終了済みおよび移動済みのジョブをキュー待機中および実行中のジョブと共に表示すると、ジョブ102がserver2のdestqに移動したことが表示されます。

```
qstat -x
Job id            Name         User    Time Use  S   Queue
-------------     -----------  ------  -------  ---  ------
101.server1       STDIN        user1   00:00:00  F   workq
102.server1       STDIN        user1   00:00:00  M   destq@server2
103.server1       STDIN        user1   00:00:00  R   workq
104.server1       STDIN        user1   00:00:00  Q   workq
```

### 10.1.2.6.iii 終了したジョブと移動したジョブに制限した検索

qstatコマンドで-Hオプションを使用すると、終了または移動されたジョブのジョブ履歴を代替フォーマットで表示できます。この場合、実行中またはキュー待機中のジョブは表示されません。

- 終了したジョブまたは移動したジョブの情報を代替フォーマットで表示するには次のように指定します。

  *qstat -H*
- 終了済みか移動済みかに関係なく、特定のジョブの情報を代替フォーマットで表示するには次のように指定します。

  *qstat -H <ジョブID>*
- 特定の宛先にある終了したジョブまたは移動したジョブの情報を表示するには次のように指定します。

  *qstat -H <宛先>*

- 終了済みおよび移動済みのジョブ、ジョブアレイ、およびサブジョブのステータスを代替フォーマットで表示するには次のように指定します。

  *qstat -Ht*
- 終了済みまたは移動済みのジョブアレイのステータスを代替フォーマットで表示するには次のように指定します。

  *qstat -HJ*

例10-7： 代替フォーマットのジョブ履歴：

```
qstat -H
                                                    Req'd  Req'd   Elap
Job ID  Username Queue Jobname    SessID NDS TSK Memory Time  S Time
------  -------- ----  ------- ------  --- --- ------ ----- -- -----
101.S1  user1    workq STDIN    5168     1   1    --     --    F  00:00
102.S1  user1    Q1@S2 STDIN     --      1   2    --     --    M   --
```

-Hオプションには-a、-i、-fおよび-rの各オプションとの互換性がありません。

### 10.1.2.7 ジョブのグループ化とIDによるソート

-Eオプションを使用すると、qstat出力でジョブをソートおよびグループ化できます。-Eオプションは、サーバーによってジョブをグループ化し、IDの昇順で各グループを表示します。このオプションでは、qstatのパフォーマンスも向上します。このオプションは、順不同のジョブIDリストをサーバー別にグループ化して順序付けられた結果で表示する場合に役立ちます。次の表は、-EオプションがqstatのIDに与える影響を示しています。

**表10-1：-Eオプションがqstat出力に与える影響**

| qstatの使用方法 | -Eがない場合の結果 | -Eがある場合の結果 |
|---|---|---|
| qstat（ジョブIDは指定されない） | デフォルトサーバーに照会し、結果を表示。 | 動作に変化はなし。-Eオプションがない場合と同じ。 |
| qstat <単一サーバーのジョブIDのリスト> | 指定した順に結果を表示。 | IDの昇順で結果を表示。 |
| qstat <複数サーバーのジョブID> | 指定した順に結果を表示。 | サーバー別にジョブをグループ化。昇順で各グループを表示。 |

# 10.2 ジョブの検査

## 10.2.1 ジョブ情報の表示方法（出力フォーマット）

主に次の3種類のフォーマットでジョブ情報を入手できます：

- *デフォルトフォーマット*：ジョブIDごとに1行として、ジョブオーナーのユーザー名、ジョブの状態、そのキューなどの情報もその行に記述した基本的な表。190ページの第10.2.1.1節「基本的なジョブリスト：デフォルトフォーマットのジョブステータス」をご参照ください。
- *代替フォーマット*：ジョブIDごとに1行として、セッションID、要求時間、経過時間などもその行に記述した詳細な表。190ページの第10.2.1.2節「拡張したジョブリスト代替フォーマットのジョブステータス」をご参照ください。
- *詳細フォーマット*：一度に 1 つのジョブの情報を表示し、各ジョブ属性と各リソースを個別の行に記述したフォーマット。191ページの第10.2.1.3節「すべてのジョブ情報：詳細フォーマットのジョブステータス」をご参照ください。

### 10.2.1.1 基本的なジョブリスト：デフォルトフォーマットのジョブステータス

デフォルトのqstat出力フォーマットでは、1つのジョブを1行としてジョブのリストが表示されます。構文：

*qstat*

*qstat [-E] [-J] [-p] [-t] [-w] [-x] [[<ジョブD> | <宛先>] ...]*

デフォルトの表示には、次の情報が含まれています。

- PBSによって割り当てられたジョブ識別子
- 投入者によって指定されたジョブ名
- ジョブオーナー
- 使用したCPUタイム
- ジョブの状態（『PBS Professional Reference Guide』の「Job States」（361ページ）をご参照ください）
- ジョブが属するキュー

qstatのデフォルト出力フォーマットの例を以下に示します。

```
qstat
Job id    Name        User        Time Use S Queue
--------- ----------- ----------- -------- - -----
16.south  aims14      user1              0 H workq
18.south  aims14      user1              0 W workq
26.south  airfoil     barry       00:21:03 R workq
27.south  airfoil     barry       21:09:12 R workq
28.south  myjob       user1              0 Q workq
29.south  tns3d       susan              0 Q workq
30.south  airfoil     barry              0 Q workq
31.south  seq_35_3    donald             0 Q workq
```

### 10.2.1.2 拡張したジョブリスト代替フォーマットのジョブステータス

代替のqstat出力フォーマットでは、1つのジョブを1行として、基本的なジョブ情報よりも詳しい情報によるジョブのリストが表示されます。構文：

*qstat -a*

*qstat [-a | -H | -i | -r ] [-E] [-G | -M] [-J] [-n [-1]] [-s [-1]] [-t] [-T] [-u <ユーザーのリスト>] [-w] [[<ジョブD> | <宛先>] ...]*

代替フォーマットでは次の各フィールドが表示されます。

- ジョブID
- ジョブオーナー
- ジョブが属するキュー
- ジョブ名
- セッションID（ジョブの実行中のみ表示）
- 要求されたチャンクまたはvnodeの数
- 要求されたCPUの数
- 要求されたメモリの量
- 要求されたCPU時間（CPU時間が要求された場合。CPU時間が要求されなかった場合は要求された経過時間）

- ジョブの状態
- 経過CPU時間（CPU時間が要求された場合。CPU時間が要求されなかった場合は経過した経過時間）

**qstat -a**

```
                                                  Req'd  Elap
Job ID   User   Queue Jobname Ses NDS TSK Mem Time S Time
-------- ------ ----- ------- --- --- --- --- ---- - ----
16.south user1  workq aims14  --  --   1  --  0:01 H  --
18.south user1  workq aims14  --  --   1  --  0:01 W  --
51.south barry  workq airfoil 930 --   1  --  0:13 R 0:01
52.south user1  workq myjob   --  --   1  --  0:10 Q  --
53.south susan  workq tns3d   --  --   1  --  0:20 Q  --
54.south barry  workq airfoil --  --   1  --  0:13 Q  --
55.south donald workq seq_35_ --  --   1  --  2:00 Q  --
```

-1 オプションを使用すると、qstat 出力を 1 行にフォーマットし直すことができます。このオプションは、-n または -s、あるいはその両方のオプションと組み合わせることによってのみ使用できます。

### 10.2.1.3 すべてのジョブ情報：詳細フォーマットのジョブステータス

qstat の詳細フォーマット出力には、ジョブの属性値とリソース値をはじめとして、ジョブに関する情報がすべて表示されます。構文と例：

*qstat -f*

*qstat -f [-F json|dsv [-D <区切り記号>]] [-E] [-J] [-p] [-t] [-w] [-x] [[< ジョブID> | <宛先>] ...]*

```
qstat -f 13
Job Id: 13.host1
    Job_Name = STDIN
    Job_Owner = user1@host2
    resources_used.cpupercent = 0
    resources_used.cput = 00:00:00
    resources_used.mem = 2408kb
    resources_used.ncpus = 1
    resources_used.vmem = 12392kb
    resources_used.walltime = 00:01:31
    job_state = R
    queue = workq
    server = host1
    Checkpoint = u
    ctime = Thu Apr  2 12:07:05 2010
    Error_Path = host2:/home/user1/STDIN.e13
    exec_host = host2/0
    exec_vnode = (host3:ncpus=1)
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Thu Apr  2 12:07:07 2010
```

```
Output_Path = host2:/home/user1/STDIN.o13
Priority = 0
qtime = Thu Apr  2 12:07:05 2010
Rerunable = True
Resource_List.ncpus = 1
Resource_List.nodect = 1
Resource_List.place = free
Resource_List.select = host=host3
stime = Thu Apr  2 12:07:08 2010
session_id = 32704
jobdir = /home/user1
substate = 42
Variable_List = PBS_O_HOME=/home/user1,PBS_O_LANG=en_US.UTF-8,
    PBS_O_LOGNAME=user1,
    PBS_O_PATH=/opt/gnome/sbin:/root/bin:/usr/local/bin:/usr/bin:/usr/X11R
    6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/mit/bin:/us
    r/lib/mit/sbin,PBS_O_MAIL=/var/mail/root,PBS_O_SHELL=/bin/bash,
    PBS_O_HOST=host2,PBS_O_WORKDIR=/home/user1,PBS_O_SYSTEM=Linux,
    PBS_O_QUEUE=workq
comment = Job run at Thu Apr 02 at 12:07 on (host3:ncpus=1)
alt_id = <dom0:job ID xmlns:dom0="http://schemas.microsoft.com/HPCS2008/hpcb
    p">149</dom0:Job ID>
etime = Thu Apr  2 12:07:05 2010
Submit_arguments = -lselect=host=host3 -- ping -n 100 127.0.0.1
executable = <jsdl-hpcpa:Executable>ping</jsdl-hpcpa:Executable>
argument_list = <jsdl-hpcpa:Argument>-n</jsdl-hpcpa:Argument><jsdl-hpcpa:Ar
    gument>100</jsdl-hpcpa:Argument><jsdl-hpcpa:Argument>127.0.0.1</jsdl-hp
    cpa:Argument>
```

各ジョブ属性については『PBS Professional Reference Guide』の「Job Attributes」（327 ページ）をご参照ください。

### 10.2.1.4 デフォルトフォーマットと代替フォーマットでの追加ジョブ情報の表示

詳細フォーマットにはジョブに関するすべての情報が表示されますが、より簡潔なフォーマット（デフォルトまたは代替のフォーマット）でジョブを確認する場合は次の方法があります。

### 10.2.1.4.i ジョブに割り当てられたホストのリスト表示

qstatで“-n”オプションを指定すると、実行中の任意のジョブに割り当てられているホストを代替フォーマットで表示できます。ジョブの直下に、この情報がexec_host情報として表示されます。実行中ではないジョブはテキスト文字列“--”で示されます。次の例では、キューで待機中のジョブと実行中のジョブの違いを確認できます。

```
qstat -n

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
 --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
 --
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
 south/0
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
 --
```

### 10.2.1.4.ii ジョブコメントの表示

qstatで“-s”オプションを指定すると、代替フォーマットで表示される情報のほかにジョブコメントが表示されます。ジョブコメントは、ジョブのすぐ下に出力されます。デフォルトでは、ジョブコメントは、所定のジョブが実行されていない理由があるとき、またはジョブの実行が開始されたときにスケジューラによって更新されます。コメントが設定されていないジョブはテキスト文字列“--”で示されます。次の例は、表示可能なメッセージの種類を示しています。

```
qstat -s

                                                  Req'd  Elap
Job ID   User  Queue Jobname Sess NDS TSK Mem Time S Time
-------- ----- ----- ------- ---- --- --- --- ---- - ----
16.south user1 workq aims14   --   --   1  -- 0:01 H  --
    Job held by user1 on Wed Aug 22 13:06:11 2004
18.south user1 workq aims14   --   --   1  -- 0:01 W  --
    Waiting on user requested start time
51.south barry workq airfoil  930  --   1  -- 0:13 R 0:01
    Job run on host south - started Thu Aug 23 at 10:56
52.south user1 workq my_job   --   --   1  -- 0:10 Q  --
    Not Running: No available resources on nodes
57.south susan workq solver   --   --   2  -- 0:20 Q  --
    --
```

### 10.2.1.4.iii 完了したジョブアレイのパーセンテージ出力

qstatで“-p”オプションを使用すると、デフォルトの表示にPercentage Completed列を付けて出力します。ジョブアレイについては、これは、完了し削除されたサブジョブの数をサブジョブの総数で割ったものです。以下に例を示します。

```
qstat -p
Job ID           Name      User           % done  S  Queue
-------------    --------  -----------  --------  -  -----
44[].host1       STDIN     user1              40  B  workq
```

### 10.2.1.4.iv ジョブの開始時刻の表示

ジョブの開始時刻を調べるには2つの方法があります。ジョブがまだ実行中の場合は、qstat -f を実行して stime 属性を調べることができます。ジョブが終了している場合は、アカウンティングログで該当するジョブの *S* レコードを確認します。アレイジョブの場合は、*S* レコードのみ使用できます。アレイジョブには、stime 属性の値がありません。

### 10.2.1.4.v ジョブの予想開始時刻の表示

予測されるジョブの開始時刻と vnode は、qstat コマンドを使用して表示できます。ジョブ情報を表示する場合、qstat で -T オプションを使用すると、*Elap Time* フィールドは *Est Start Time* フィールドで置き換えられます。実行中のジョブはキューイングされるジョブの上に表示されます。実行中のジョブは stime 属性（開始時刻）に基づいてソートされます。

予測される開始時刻が設定されていない（estimated.start_time = *unset*）キュー待機中のジョブは、予測される開始時刻が設定されているジョブの後に、予測される開始時刻が二重ダッシュ（“--”）で示されて表示されます。予測される開始時刻が過去の時間に設定されているキューイングされるジョブは、その予測される開始時刻が設定されていないものとして処理されます。

表示される時刻は qstat コマンドにローカルな時刻です。現行の週は日曜日から始まります。

予測される開始時刻または vnode 情報が権限のないユーザーに非表示になっている場合は、qstat を使用しても、予測される開始時刻または vnode 情報は表示されません。

出力例：

```
qstat -T
                                                                  Est
                                                Req'd  Req'd      Start
Job ID   Username Queue Jobname SessID NDS TSK Memory Time  S Time
-------  -------- ----- -------- ----- --- --- ------ ----- - -----
5.host1   user1   workq  foojob  12345  1   1  128mb 00:10  R   --
9.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  11:30
10.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q  Tu 15
7.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  Jul
8.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  2010
11.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q  >5yrs
13.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q   --
```

ジョブの開始時刻が予想できない場合、開始時刻は疑問符（“?”）として表示されます。

### 10.2.1.4.vi 予測される開始時刻が変化する理由

以下の理由により、ジョブの予測される開始時刻が変化する可能性があります：

- システムが変更（vnode の停止など）された場合または管理者が vnode をオフラインにした場合
- 優先順位の高いジョブがシステムに出現した場合または既存のジョブの優先順位が変動した場合

## 10.2.1.5 出力フォーマット特性の変更

### 10.2.1.5.i サイズの表示（ギガバイトまたはメガワード）

デフォルトでは、最短の数字で表示できる単位でサイズが表示されます。qstat で -G オプションを指定するとサイズがギガバイト単位で表示され、-M オプションを指定するとメガワード単位で表示されます。どちらのオプションでも表示は代替フォーマットになります。-G を指定して、実際のサイズが 1GB 未満の場合、出力の表示は 1GB に繰り上げられます。１ワードは、8 バイトです。

以下に例を示します。

```
qstat -G
host1:
                                                  Req'd  Req'd     Elap
Job ID     User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
---------  ----- ----- ------- ---- --- ---  -----  -----  -  -----
43.host1   user1 workq STDIN   4693   1   1     --     --  R  00:05
44[].host1 user1 workq STDIN     --   1   1     --     --  Q     --
45.host1   user1 workq STDIN     --   1   1    1gb     --  Q     --
```

以下に例を示します。

```
qstat -M
host1:
                                                  Req'd  Req'd     Elap
Job ID     User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
---------- ----- ----- ------- ---- --- ---  -----  -----  -  -----
43.host1   user1 workq STDIN   4693   1   1     --     --  R  00:05
44[].host1 user1 workq STDIN     --   1   1     --     --  Q     --
45.host1   user1 workq STDIN     --   1   1   25mw     --  Q     --
```

#### 10.2.1.5.ii 広い列でのステータスの表示

qstatで-wオプションを指定すると、デフォルトフォーマットと代替フォーマットではジョブのステータスが広い列で表示されます。表示幅の合計が80文字から120文字に拡大されます。ジョブIDの列幅は最大で30文字、ユーザー名、キュー、およびジョブ名の列幅は最大で15文字です。セッションIDの列幅は最大で8文字、NDSの列幅は最大で4文字です。

このオプションは、-a、-n、または-s qstatオプションと組み合わせてのみ使用できます。

このオプションは、-fと共に使用される-wオプションとは異なります。

#### 10.2.1.5.iii Windowsでのパス表示

マップされたドライブから投入されたジョブを詳細フォーマットで表示する場合、ジョブのOutput_Path属性、Error_Path属性、およびジョブのVariable_List属性のPBS_O_WORKDIRの値はUNCパスで表示されます。

出力ファイルとエラーファイルにUNCパスを指定して投入されたジョブを詳細フォーマットで表示する場合、ジョブのOutput_Path属性とError_Path属性はUNCパスで表示されます。

## 10.2.2 ジョブによるリソース使用量の検査

### 10.2.2.1 実行中のジョブとキュー待機中のジョブによるリソース使用量の検査

次のように指定してジョブを詳細フォーマットで表示すると、それぞれ実行中のジョブ、ジョブアレイ、サブジョブによるリソース使用量を確認できます。

```
qstat -f <ジョブID>
```

### 10.2.2.2 終了したジョブと移動したジョブによるリソース使用量の検査

それぞれ終了および移動したジョブとジョブアレイで使用されたリソースの量を確認できます。終了および移動したサブジョブによる使用量は表示されません。

#### 10.2.2.2.i 終了および移動したジョブとジョブアレイによるリソース使用量の検査

qstatで-f出力オプションを指定すると、詳細フォーマットでリソース使用量を表示できます。それぞれ終了および移動したジョブとジョブアレイで使用されたリソースの量を確認するには、qselectコマンドの出力を使用して、qstatでリスト出力したジョブをフィルタ処理します。すべてのジョブを対象として、詳細フォーマットによるリソースの表示をすべて出力することをqstatに指示します。

Linux：

```
qstat -fx `qselect -H`
```

Windows：

```
for /F "usebackq" %%j in (`"¥Program Files¥ PBSPro¥ exec¥ bin¥qselect" -H`)
    do ("¥Program Files¥PBS¥exec¥bin¥qstat" -fx %%j)
```

#### 10.2.2.2.ii 終了したサブジョブと移動したサブジョブによるリソース使用量の検査

終了したサブジョブと移動したサブジョブによるリソース使用量はアカウンティングログでのみ確認できますが、このログを利用できるのはrootユーザーとPBS管理者のみです。

### 10.2.3 ジョブ情報の注意事項

- MoM は、ホストで実行中のジョブによる使用量に対して定期的にポーリングし、結果を収集してサーバーに報告します。ジョブが終了すると再びポーリングして、そのジョブの最終的な使用量を取得します。

  たとえば、MoMはT1、T2、T4、T8、T16、T24などの時刻に実行中のジョブをポーリングします。

  T8からT16までの時間にqstatを実行した場合の出力は、T8までのリソース使用量を表します。

  T17でqstatを実行すると、T16までの使用量が表示されます。T20にジョブが終了した場合、アカウンティングログ（および最後のログメッセージ、およびジョブ投入で"qsub -me"が使用された場合にユーザーに送信されるメール）には、T20までの使用量が含まれます。
- 最後のレポートはepilogueを含みません。epilogueに必要な時間は、システムオーバーヘッドとして処理されます。
- ジョブが表示される順序は不定です。

## 10.3 サーバーステータスのチェック

サーバー情報をデフォルトフォーマットで表示するには、以下を実行します。

*qstat -B [< サーバー > ...]*

サーバー情報を詳細フォーマットで表示するには、以下を実行します。

*qstat -B -f [-F json|dsv [-D <区切り記号>]] [-w] [< サーバー > ...]*

### 10.3.0.1 宛先の指定

宛先を指定しないと、デフォルトのサーバーが指定されます。サーバーを指定できます。そのフォーマットは次のとおりです。

*< サーバー名>*

例10-8： デフォルトではないサーバー S1のステータスを取得するには次のように指定します。

```
qstat -B S1
Server      Max  Tot  Que  Run  Hld  Wat  Trn  Ext Status
----------- --- ---- ---- ---- ---- ---- ---- ---- ------
S1.example   0   14   13    1    0    0    0    0 Active
```

## 10.3.1 デフォルトフォーマットでのサーバー情報の表示

qstatの"-B"オプションは、指定されたPBSサーバーのステータスを表示します。照会した各サーバーに対して、1行の出力が生成されます。3文字の略語はそれぞれMaximum、Total、Queued、Running、Held、Waiting、Transiting、Exitingを示しています。最後の列は、サーバー自体のステータス（active、idle、またはscheduling）を示しています。

```
qstat -B
Server      Max  Tot  Que  Run  Hld  Wat  Trn  Ext Status
----------- --- ---- ---- ---- ---- ---- ---- ---- ------
fast.domain  0   14   13    1    0    0    0    0 Active
```

## 10.3.2 詳細フォーマットでのサーバー情報の表示

サーバーステータスは、JSONまたは区切り記号で区切られた値のフォーマットで表示できます。『PBS Professional Reference Guide』の「Job, Queue, and Server Status Options」（210ページ）をご参照ください。

ジョブ、サーバー、またはキューを照会する際、"-f"オプションをqstatに追加すると、表示方法を*full*（全体）または*long*（長い）の間で切り替えることができます。たとえば、上記のサーバーステータスは、"-f"を使用することで次のように拡張できます。

```
qstat -Bf
Server: fast.mydomain.com
    server_state = Active
    scheduling = True
    total_jobs = 14
    state_count = Transit:0 Queued:13 Held:0 Waiting:0
        Running:1 Exiting:0
    managers = user1@fast.mydomain.com
    default_queue = workq
    log_events = 511
    mail_from = adm
    query_other_jobs = True
    resources_available.mem = 64mb
    resources_available.ncpus = 2
    resources_default.ncpus = 1
    resources_assigned.ncpus = 1
    resources_assigned.nodect = 1
    scheduler_iteration = 600
    pbs_version = PBSPro_2022.1.41640
```

# 10.4 キューステータスのチェック

キュー情報をデフォルトフォーマットで表示するには、以下を実行します。

*qstat -Q [<宛先> ...]*

キュー情報を代替フォーマットで表示するには、以下を実行します。

*qstat -q [-G | -M] [<宛先> ...]*

キュー情報を詳細フォーマットで表示するには、以下を実行します。

*qstat -Q -f [-F json|dsv [-D <区切り記号>]] [-w] [<宛先> ...]*

## 10.4.1 宛先の指定

宛先を指定しないと、すべてのキューとデフォルトのサーバーからジョブが得られます。キューまたはサーバー（あるいはその両方）を指定できます。

- デフォルトのサーバーにある指定したキューのステータスを表示するには次のように指定します。
  *<キュー名>*
- 指定したサーバーにある指定したキューのステータスを表示するには次のように指定します。
  *<キュー名>@<サーバー名>*
- 指定したサーバーにあるすべてのキューのステータスを表示するには次のように指定します。
  *@<サーバー名>*

## 10.4.2 デフォルトフォーマットでのキュー情報の表示

qstatで"-Q"オプションを指定すると、指定したキューのステータスが表示されます。照会した各キューに対して、1行の出力が生成されます。

```
qstat -Q
Queue Max Tot Ena Str Que Run Hld Wat Trn Ext Type
----- --- --- --- --- --- --- --- --- --- --- ---------
workq   0  10 yes yes   7   1   1   1   0   0 Execution
```

列には、各キューの以下の情報が表示されます。

- Queue キュー名
- Max キュー内の同時実行が可能なジョブの最大数
- Tot キュー内のジョブの総数
- Ena キューが有効か無効か
- Str キューが開始されているか停止されているか
- Que キューイングされているジョブ数
- Run 実行中のジョブ数
- Hld 保留中のジョブ数
- Wat 待機中のジョブ数
- Trn 移動（移行）中のジョブ数
- Ext 既存のジョブ数
- Type キューのタイプ：実行またはルーティング

### 10.4.3 代替フォーマットでのキューの制限の表示

qstatで"-q"オプションを指定すると、要求したキュー（またはデフォルトのキュー）に制限が設定されていれば、それがすべて表示されます。出荷する時点でPBSはキュー制限が設定されていないため、表示される制限はサイトに固有のものです。制限は、次のような形式で表示されます。

```
qstat -q
server: south

Queue  Memory CPU Time Walltime Node Run Que Lm  State
------ ------ -------- -------- ---- --- --- --  -----
workq    --      --       --     --    1   8 --   E R
```

### 10.4.4 詳細フォーマットでのキュー情報の表示

キュー情報は、JSONまたは区切り記号で区切られた値のフォーマットで表示できます。『PBS Professional Reference Guide』の「Job, Queue, and Server Status Options」（210ページ）をご参照ください。

キューの各属性の値を確認するには、詳細フォーマットを使用します。

```
qstat -Qf
Queue: workq
    queue_type = Execution
    total_jobs = 10
    state_count = Transit:0 Queued:7 Held:1 Waiting:1
        Running:1 Exiting:0
    resources_assigned.ncpus = 1
    hasnodes = False
    enabled = True
    started = True
```

### 10.4.5 qstatコマンドの注意事項

qstatで-fオプションを使用してジョブ、キュー、またはサーバーの属性を表示する際、未設定の属性は表示できません。表示されない属性は未設定です。

## 10.5 ライセンスの利用状況の確認

ライセンスが利用可能かどうかを確認できます。それには、以下のいずれかを実行します。

- 現在のホストのライセンス情報を表示：
  **qstat -Bf**
- すべてのホストで利用可能なリソース（ライセンスを含む）を表示：
  **qmgr**
  **Qmgr: print node @default**

サイトがフローティングライセンスを使用している場合、サーバーのlicense_count属性を確認するときは、*Avail_Global*値および*Avail_Local*値の合計を使用します。

# 11 クラウドでのジョブの実行

## 11.1 はじめに

クラウドキューにジョブを置くと、クラウドでジョブを実行できます。そのクラウドキューにジョブを投入するか、他のキューからジョブを移動します。各クラウドキューでは、そこにあるジョブが特定のシナリオにアクセスできます。そのようなシナリオには、特定のインスタンスタイプ、OSイメージ、アプリケーションライセンスが用意されています。シナリオごとにデフォルトのインスタンスタイプとOSイメージがあります。それぞれのジョブでは、シナリオに用意されているあらゆるインスタンスタイプ、OSイメージ、アプリケーションライセンスを要求できます。シナリオに用意されていないインスタンスタイプ、OSイメージ、アプリケーションライセンスをジョブで要求することはできません。

## 11.2 クラウドでのジョブの実行

クラウドの各シナリオには固有のクラウドキューが関連付けられ、同様に各クラウドキューにはクラウドの固有のシナリオが関連付けられています。シナリオごとに、特定のインスタンスタイプ、OSイメージ、アプリケーションライセンスが用意されています。これらの適切な組み合わせを用意しているクラウドキューにジョブを投入します。qsubに-q <キュー名>オプションを指定してキューを指定します。ジョブを投入するときにインスタンスタイプとOSイメージを要求することで、デフォルトのインスタンスタイプとOSイメージに代わってそれらを使用できます。

クラウドで実行できるジョブを投入するには、設定済みのクラウドキューにそのジョブを投入します。構文：

*qsub -q <クラウドキューの名前> -l <リソース要求> <ジョブスクリプト>*

以下に例を示します。

```
qsub -q cloudq -- /bin/sleep 100
```

### 11.2.1 インスタンスタイプの要求

シナリオごとにデフォルトのインスタンスタイプがあり、キューのリソースcloud_instance_typeで指定されています。キューのシナリオに用意されているインスタンスタイプからどれでも選択できます。インスタンスタイプを要求するには、目的のインスタンスをチャンクのリソースcloud_node_instance_typeで次のように指定します。

*qsub -lselect=...:cloud_node_instance_type=<インスタンスタイプ>:... -q <キュー名> <ジョブスクリプト>*

以下に例を示します。

```
qsub -lselect=1:ncpus=2:mem=1gb:cloud_node_instance_type=e2-highmem-8
```

要求するインスタンスタイプが、用意されているインスタンスタイプと正確に一致していることを確認します。チャンクごとに、要求するインスタンスタイプが異なることや、デフォルトのインスタンスタイプが異なることも考えられます。各チャンクは、異なるインスタンスタイプを要求またはデフォルト設定できます。

#### 11.2.1.1 プリエンプション可能インスタンスとスポットインスタンスの要求

スポットインスタンスなどのプリエンプション可能インスタンスがシナリオに用意されていれば、それを要求できます。クラウドのノードを複数要求するときは、プリエンプション可能ではないインスタンスまたはプリエンプション可能なインスタンスのどちらかのみを要求します。両方を要求することはできません。オンデマンドのクラウドノードとプリエンプション可能またはスポットのクラウドノードの混在を要求しないでください。

## 11.2.2 OSイメージの要求

シナリオごとにデフォルトのOSイメージがあり、シナリオのパラメータcloud_default_imageで指定されています。デフォルトのOSイメージを使用できるほか、キューのシナリオに用意されているOSイメージから選択することもできます。OSイメージを要求するには、selectステートメントで、目的のイメージをチャンクのリソースcloud_node_imageで指定します。

*qsub -lselect=...:cloud_node_image=<OS イメージ>:... -q <キュー名> <ジョブスクリプト>*

以下に例を示します。

```
qsub -lselect=1:ncpus=2:mem=1gb:cloud_node_image="myimages/image-1" myscript
```

チャンクごとに異なるOSイメージが使用できます。

## 11.2.3 高速ネットワークで接続したクラウドノードでのジョブの実行

同一の高速ネットワーク上にすべてのジョブノードがあるクラウドノードでジョブを実行することをお勧めします。クラウドプロバイダは、各グループが高速スイッチで接続されているノードグループをPBS Cloudにバーストさせることができます。たとえば、AzureはInfiniBandの*スケールセット*を提供し、OraIceはInfiniBandの*インスタンスプール*を提供しています。わかりやすくするために、ここでは高速ネットワーク上にあるノードのグループを*近接ノードグループ*と呼びます。

PBS Cloudが高速ネットワーク上のノードのグループをバーストするときに、その近接ノードグループのすべてのノードに、同じネットワーク名によるラベルがPBS Cloudによって割り当てられます。実際のネットワーク名を要求する必要はありません。チャンク要求`cloud_network=ib`で、ジョブがそのようなノードグループ上にあることを要求するだけです。以下の例をご参照ください。

### 11.2.3.1 高速ネットワークで接続されたクラウドインスタンスでのジョブの実行とベアメタルでのバースト

さらに、PBSを使用することで、インスタンスがベアメタルでバーストされる高速ネットワークで接続されたクラウドノードでジョブを実行できます。ベアメタルを使用しても使用しなくても、高速ネットワーク上でのジョブの実行に違いはありません。ベアメタルでインスタンスをバーストするには、適切なインスタンスタイプおよび一致するOSを選択していることを確認してください (ただし、これは高速ネットワークを使用するすべてのジョブに当てはまります)。

このバージョンでは、ベアメタル上でのインスタンスのバーストを認めているプロバイダはOracleのみです。

### 11.2.3.2 高速ネットワーク上のジョブに関する注意と制限事項

クラウドプロバイダが高速ネットワークを用意していて、そのネットワークをPBS Cloudがサポートしている場合にのみ、その高速ネットワークを使用してジョブを実行できます。現在のところ、PBS CloudではOracleとAzure上の高速ネットワークをサポートしています。

InfiniBandによるAzureノード数に対するデフォルトの上限は現時点で*100*です。

### 11.2.3.3 高速ネットワーク上のクラウドノードでジョブを実行する方法

高速ネットワークで接続されたクラウドノードのグループ（近接ノードグループ）に対してジョブを実行するには、ジョブのチャンクごとに`cloud_network=ib`を要求します。

ジョブの各チャンクが以下を取得するようになっていて、ジョブのすべてのチャンクでこれらが同じであることを確認します：

- 高速ネットワークが有効なインスタンスタイプ

  cloud_node_instance_typeで指定
- 高速ネットワークに対応したOSイメージ

  cloud_node_imageで指定

必要なインスタンスタイプと OS イメージ (たとえば、InfiniBand が有効になっている Azure シナリオ) を持つバーストシナリオに関連付けられているキューにジョブを投入できます。そこでインスタンスタイプとOSイメージを要求することもできます。

近接ノードグループでジョブを実行して、インスタンスタイプとOSイメージを要求するための構文は次のとおりです。

*qsub -q <クラウドキュー> -lselect=...:cloud_network=ib:cloud_node_instance=<高速ネットワークを有効にしたインスタンスタイプ>:cloud_node_image=<高速ネットワークに対応したOSイメージ> <ジョブスクリプト>*

以下に例を示します。

```
qsub -q cloudq -lselect=1:ncpus=2:mem=1gb:cloud_network=ib:cloud_node_instance_type=
    e2-highmem-8:cloud_node_image="projects/images/myimage" -- /bin/sleep 60
```

cloud_scenarioリソースは要求しないでください。

### 11.2.4 アプリケーションライセンスを必要とするジョブの実行

アプリケーションライセンスを必要とするジョブを実行するには、以下の手順に従います。

- 該当のアプリケーションライセンスが用意されているシナリオを選択します。
- そのアプリケーションライセンスを要求します。

  各アプリケーションライセンスは、2つのPBSリソースによって表されます。1つは静的リソースで、もう1つは動的リソースです。ジョブがアプリケーションライセンスを必要とする場合は、この両方のリソースに対する要求をジョブスクリプトに記述する必要があります。たとえば、リソースapp1_staticとapp1_dynamicによるApp1ライセンスをジョブで必要とする場合は、ジョブスクリプトに次のように記述します。

  ```
  #PBS -l app1_static=1
  #PBS -l app1_dynamic=1
  ```

## 11.3 クラウドジョブのジョブスクリプトの例

### 11.3.1 簡単なsleepジョブスクリプトの例

例11-1： 1分間スリープ (または必要に応じて $sleeptime を調整) してから終了する 10 分間のwalltimeを要求する単純なジョブがあるとします。このジョブは*cloudq*を要求し、サイトの設定に応じて名前を調整します。次のジョブスクリプトを`sleep.sh`として保存します。つづいて、それを次のようにPBSに投入します。

```
qsub sleep.sh
```

スクリプト：

```
#!/bin/bash
#PBS -N testjob
#PBS -j oe
#PBS -m n
#PBS -q cloudq
#PBS -l select=1:ncpus=2:mem=16mb
#PBS -l walltime=0:10:00
sleeptime=60
cmd="sleep $sleeptime"
echo $cmd
$cmd
exit
```

## 11.3.2 Radioss クラウドジョブスクリプトの例

例 11-2： 25 件の Radioss ライセンスを使用するクラウドジョブのジョブスクリプトを取り上げます。このスクリプトでは Intel MPI を使用します。静的リソースは “Rad_stat”、動的リソースは “Rad_dyn” です。

```
#!/bin/bash
#PBS -N RunRad
#PBS -j oe
#PBS -m n
#PBS -q CloudRadq
#PBS -P project1
#PBS -l select=1:ncpus=16:mem=16gb
#PBS -l walltime=2:00:00
#PBS -l Rad_stat=25
#PBS -l Rad_dyn=25
/usr/local/altair/scripts/radioss -mpi i -nt $NCPUS -np 1 -hostfile $PBS_NODEFILE -both
    SEAT_DYREL_0000.rad
```

## 11.3.3 ジョブ出力の表示

ジョブの終了後、ジョブの出力を確認します。これは、ジョブが投入された場所にあります。

# 12 Budgetsの使用

## 12.1 Budgetsコマンド

### 12.1.1 コマンドのパス

Budgetsコマンドを使用するには、次のコマンドを使用して、PATH環境変数にamバイナリのパスをインポートします。

```
export PATH=$PATH:/opt/am/python/bin/
```

### 12.1.2 Budgetsコマンドの使用

すべてのBudgetsコマンドには接頭辞として“amgr”が付加されています。

各コマンドが1行で説明されたBudgetsサブコマンドのリストを表示するには次のように入力します。

```
amgr <Enterキー>
```

コマンドまたはサブコマンドの使用方法を確認するには次のように入力します。

*<コマンド> --help*

*<コマンド> <サブコマンド> --help*

以下に例を示します。

`amgr add --help`と入力すると、`amgr add`コマンドの使用方法が表示されます。

`amgr add period --help`と入力すると、`period`サブコマンドの使用方法が表示されます。

必須の引数を省略してコマンドを入力すると、その引数の入力を求められます。

『PBS Professional Budgets Guide』の「Budgets Commands」（77ページ）をご参照ください。

## 12.2 Budgetsによるジョブの投入

ジョブを投入する前に、そのジョブのコスト見積もりをBudgetsに要求できます。

投入するジョブがキューイングされるためには、そのジョブをBudgetsで検証する必要があります。そのジョブを有効なユーザーアカウントまたはプロジェクトアカウントに課金していることを確認します。

プロジェクトジョブは、そのプロジェクトに関連付けたPBSコンプレックスで実行できます。ユーザー自身のジョブは、そのユーザーに関連付けたアカウントで実行できます。ユーザーのアカウントがアクティブな場合にのみ、そのユーザーが自身のジョブを実行できます。

ジョブ投入者がBudgetsにログインする必要はありません。

## 12.2.1 Budgetsからのジョブコスト見積もりの入手

ジョブを投入する前に、そのジョブのコスト見積もりをBudgetsから入手できます。ジョブを投入した場合に必要になるコストを各種通貨で知ることができます。見積もりを要求すると、ジョブに伴う各種コストの見積もりがPBSから出力されます。クラウドでのジョブまたはオンプレミスでのジョブで見積もりを得ることができます。

Budgetsでは、各ジョブで必要とするリソースに基づいてコストが見積もられますが、実行したジョブの最終的なコストは実際に使用されたリソースに基づいて計算されます。したがって、見積もりと実際のコストが異なることがあります。

管理者がBudgetsに`quote`コマンドを設定していれば、そのコマンドを使用してジョブの見積もりを得ることができます。このコマンドが設定されていない場合は、`qsub`コマンドを使用してジョブの見積もりを入手します。`qsub`コマンドを使用して見積もりを入手する場合は、通常のジョブ投入方法を使用しますが、ジョブが実際に実行されることはありません。

### 12.2.1.1 quoteコマンドによるコスト見積もりの要求

`quote`コマンドを使用してジョブのコスト見積もりを入手するには、ジョブを投入する場合と同じジョブスクリプトまたはリソース要求を使用しますが、それを次のように`quote`コマンドに渡します。

*quote < ジョブスクリプト >*

*quote -l < リソース >=< 値 > -lselect=...*

以下に例を示します。

```
quote my_job_script.sh
quote -l walltime=1:00 -lselect=2:ncpus=4:mem=8GB --/bin/sleep 30
```

#### 12.2.1.1.i quoteコマンドによるコスト見積もり要求の例

例12-1： 30秒間のwalltimeを要求するsleepコマンドを投入すると、そのsleepコマンドが30秒間のスリープを要求し、CPU時間1秒あたりの課金が1セントである場合は次のように入力します。

```
quote -l walltime=30 --/bin/sleep 30
qsub: Budgets estimate for job cost: {"cpu_sec": 30.0, "dollar": 0.30}
```

このジョブを実行すると、30セントの課金が発生します（CPU時間1秒あたり1セントの課金）。

例12-2： 同じく30秒間のwalltimeを要求するsleepコマンドを投入しますが、そのsleepコマンドが20秒間のスリープを要求し、CPU時間1秒あたりの課金が1セントである場合は次のように入力します。

```
quote -l walltime=30 --/bin/sleep 20
qsub: Budgets estimate for job cost: {"cpu_sec": 30.0, "dollar": 0.30}
```

このジョブを実行すると、20セントの課金が発生します（CPU時間1秒あたり1セントの課金）。

### 12.2.1.2 qsubコマンドによるコスト見積もりの要求

`qsub`コマンドを使用してジョブのコスト見積もりを入手するには、ジョブを投入する場合と同様の`qsub`コマンドを使用しますが、次のように“`-l am_job_quote=true`”を指定します。

```
qsub ... -l am_job_quote=true
```

このオプションを指定すると、ジョブは実行されず、評価されるだけとなります。

#### 12.2.1.3 見積もりのフォーマット

Budgetsは、ジョブが要求した見積を通貨ごとに出力します。そのフォーマットは次のとおりです。

*qsub: Budgets estimate for job cost: {"<通貨名>": <値>, "<通貨名>": <値>, ...}*

##### 12.2.1.3.i qsubコマンドによるコスト見積もり要求の例

例12-3： 30秒間のwalltimeを要求するsleepコマンドを投入すると、そのsleepコマンドが30秒間のスリープを要求し、CPU時間1秒あたりの課金が1セントである場合は次のように入力します。

```
qsub -l walltime=30 -l am_job_quote=t --/bin/sleep 30
qsub: Budgets estimate for job cost: {"cpu_sec": 30.0, "dollar": 0.30}
```

このジョブを実行すると、30セントの課金が発生します（CPU時間1秒あたり1セントの課金）。

例12-4： 同じく30秒間のwalltimeを要求するsleepコマンドを投入しますが、そのsleepコマンドが20秒間のスリープを要求し、CPU時間1秒あたりの課金が1セントである場合は次のように入力します。

```
qsub -l walltime=30 -l am_job_quote=t --/bin/sleep 20
qsub: Budgets estimate for job cost: {"cpu_sec": 30.0, "dollar": 0.30}
```

このジョブを実行すると、20セントの課金が発生します（CPU時間1秒あたり1セントの課金）。

#### 12.2.1.4 ジョブコスト見積もりに関する注意と制限事項

ノードによっては、特殊なコストが関連付けられていることがあります。Budgets側では、ジョブがどこで実行されるかを知ることができないので、固有のノードに関連付けられているコストを認識できません。このような特殊なコストが存在する場合、Budgetsにより見積もりではそれが考慮されません。

### 12.2.2 ジョブの実行に必要なクレジットがあるかどうかの確認

特定のジョブ（1件または複数件）を実行するうえで十分なクレジットがあるかどうかをBudgetsに問い合わせて確認できます。確認対象として複数のジョブを記述したリストをコマンドで指定すると、各ジョブが個別に評価され、リストにある他のジョブが考慮されない点に注意が必要です。したがって、たとえば、手元に10クレジットがあるとして、それぞれ10クレジットを必要とするジョブ2件の確認を要求すると、コマンドからは両方のジョブを実行できると回答があります。ジョブについてBudgetsに問い合わせるには`amgr precheck jobs`を使用します。『PBS Professional Budgets Guide』の「Prechecking Jobs」（126ページ）をご参照ください。

### 12.2.3 ユーザーアカウントまたはプロジェクトアカウントへのジョブの課金

ジョブを投入すると、アカウントにそのジョブが課金されます。

- プロジェクトアカウントにジョブを課金するには、`qsub -P <プロジェクト名>`を使用してプロジェクトを指定します。たとえば、ProjectScriptを使用して1時間のジョブを実行し、名前が“Project1”のプロジェクトにそのジョブを課金するには、次のように指定します。
  ```
  qsub -P Project1 -l select=1:ncpus=1:mem=1gb -l walltime=1:00:00 ProjectScript
  ```
- ユーザー自身のアカウントに課金するにはプロジェクトを指定しません。たとえば、MyScript を使用して 1 時間のジョブを実行し、自身のアカウントに課金するには、次のように指定します。
  ```
  qsub  -l select=1:ncpus=1:mem=1gb -l walltime=1:00:00 MyScript
  ```

## 12.2.4 クレジット

各プロジェクトにはそのクレジットバランスがあり、ジョブ投入者にも自身のクレジットバランスがあります。クレジットは、標準のサービス単位で数値化されています。サービス単位は、PBSで追跡しているリソースの使用量であり、CPU時間やGPU時間などがあります。サービス単位を通貨のように扱うことができます。ユーザーアカウントとプロジェクトアカウントへのサービス単位の預け入れはグループマネージャが担当します。

## 12.2.5 後払いモードでのジョブの投入

後払いモードでは、ジョブの実行にクレジットを必要としません。後払いモードでのジョブの実行はクレジットカードの使用に似ていますが、課金できる金額にBudgetsによる制限がなく、決済の必要がない点が異なります。

後払いモードの場合、課金先をユーザー自身のアカウントまたはユーザーが属するプロジェクトアカウントとすることができます。

ジョブが終了すると、そのジョブで消費したクレジットがBudgetsによって引き落とされます。

ユーザーが料金を即金で支払い、管理者がそのジョブについて返金することもできます。

## 12.2.6 前払いモードでのジョブの投入

前払いモードでは、ジョブを実行するためにクレジットが必要です。

前払いモードの場合、ユーザーに十分なクレジットがあれば、ユーザー自身のアカウントにジョブを課金できます。プロジェクトに参加している場合、そのプロジェクトに十分なクレジットがあれば、そのプロジェクトのアカウントにプロジェクトジョブを課金できます。

ジョブで使用するクレジットアカウントに十分なクレジットがある場合にのみ、そのジョブを開始できます。ジョブが始まると、要求されたクレジットがエスクローに置かれます。ジョブが終了すると、使用されなかったクレジットがアカウントに返されます。Budgetsでは、クレジットバランスがマイナスになることが認められません。

ジョブを実行するためのクレジットが不足していると、そのジョブは*ユーザー保留*で保持されます。そのようなジョブを実行できるようにするには、まずジョブを実行できる量のクレジットを用意し、qrlsを使用してジョブに対する保留を解除します。

## 12.2.7 ジョブのリソース要件

各ジョブは、該当のコンプレックスで使用されている請求式で、使用するコンピュータリソースを要求する必要があります。デフォルトの式では、walltimeとncpusがコンピュータリソースです。各PBSジョブによって、その実行時にncpusとwalltimeが要求されていることを確認します。このような要求のジョブごとの値は、ジョブ投入者がジョブ投入時またはそれ以降に`qalter`を使用して設定できるほか、サーバーまたはキューから継承することや、hookによって割り当てることもできます。ncpusについては、サーバー属性default_chunk.ncpusで要件を扱うことができます。

## 12.2.8 アカウンティングポリシー

ジョブは、実行した期間単位で課金されます。プロジェクトジョブは、プロジェクトのアカウンティングポリシーに従って課金されます。ユーザーのジョブは、ユーザーのアカウンティングポリシーに従って課金されます。アカウンティングポリシーは次のいずれかです：

begin_period

ジョブが始まるとユーザーアカウントまたはプロジェクトアカウントに課金されます。

end_period
: ジョブが終了するとユーザーアカウントまたはプロジェクトアカウントに課金されます。

proportionate
: ジョブが実行されている全期間にわたってユーザーアカウントまたはプロジェクトアカウントに課金されます。各期間の課金は、その期間でのリソース使用量に比例します。

## 12.2.9 割当期間

クレジットバランスは特定の期間に割り当てられます。各期間には開始日と終了日が定義されています。

定義されている期間を確認するには次のように入力します。

```
amgr ls period
```

『PBS Professional Budgets Guide』の「Listing Periods」（88ページ）をご参照ください。

前払いモードでは、グループマネージャや管理者が、ある期間から別の期間にクレジットを転送できますが、その期間のクレジットは期間が終了すると使用できなくなります。

## 12.2.10 クレジットバランスの確認

クレジットバランスを確認するには、次のように期間を指定します。

*amgr checkbalance -n <ユーザー名> -p <期間>*

たとえば、ユーザー名がMyUsernameで、目的の期間がQ4であれば、次のように指定します。

```
amgr checkbalance -n MyUsername -p Q4
```

『PBS Professional Budgets Guide』の「Checking Service Unit Balance for User」（122ページ）をご参照ください。

## 12.2.11 クラスタの列挙

Budgetsに関連付けられたPBSコンプレックスであるクラスタを、次のように入力して列挙できます。

*amgr ls cluster [-n <PBS サーバー>] [-a <アクティブかどうかの指定>] [-f ] [ -l | -j ]*

以下に例を示します。

```
amgr ls cluster -n Cluster1 -f
```

『PBS Professional Budgets Guide』の「Listing Clusters」（87ページ）をご参照ください。

## 12.2.12 外部リソースのクォータ

ストレージのように外部で管理されているリソースにはクォータが設定されていることがあります。クォータとは、動的なサービス単位に対する制限です。クォータを確認するには、次のように入力してすべてのサービス単位を列挙します。

```
amgr ls serviceunit
```

『PBS Professional Budgets Guide』の「Listing Service Units」（88ページ）をご参照ください。

### 12.2.13 使用量とトランザクションに関するレポートの入手

次のように入力すると、ユーザー自身による使用量のレポートを入手できます。

*amgr report user -n <ユーザー名> [-s <サービス単位名> | -t <サービス単位の種類>] -h <グループ名> -p <期間> -S <開始日> -E <終了日> [-l ] [ -o <出力ファイル>] [-r]*

たとえば、期間Q4のMyUsernameに関するレポートを入手するには次のように指定します。

```
amgr report user -n MyUsername -p Q4
```

『PBS Professional Budgets Guide』の「Getting User Reports」（104ページ）をご参照ください。

次のように入力すると、ユーザー自身に関係するトランザクションのレポートを入手できます。

*amgr report transaction -i <トランザクションID> [-l ] -N <数> [ -o <出力ファイル>] [ -r ]*

## 12.3 チュートリアル

### 12.3.1 前払いモードでBudgetsを使用するチュートリアル

#### 12.3.1.1 前提条件

PBS Professionalがインストールされ動作する環境で、コンプレックスでジョブの投入と実行ができるアカウントが2つ以上あること。ここの例では、クラスタ名をCluster1、ユーザーをUser1およびUser2とします。User1は、クラスタCluster1とプロジェクトP1に関連付けられています。User1は1,200cpu_hrsのクレジットを持ち、P1は1,000cpu_hrsのクレジットを持っています。

チュートリアルにあるクラスタ、プロジェクト、およびユーザー名を各自の使用する名前に置き換えてください。

#### 12.3.1.2 Budgetsを使用するチュートリアルの手順

##### 12.3.1.2.i ユーザージョブの実行

1. User1としてログインします。
2. 次のようにwalltimeを2分として10秒間のsleepジョブを実行し、ユーザーUser1に課金します。

   ```
   qsub -lwalltime=00:02:00 -- /bin/sleep 10
   ```

3. 次のように、ユーザーUser1のクレジットバランスを確認します。そのクレジットバランスは減少しています。

   ```
   amgr checkbalance user -n User1 -p 2022.Q2
   ```

4. ジョブが終了した後、このバランスを次のように再度確認します。使用されなかった量のクレジットが返却されていることがわかります。

   ```
   amgr checkbalance user -n User1 -p 2022.Q2
   ```

### 12.3.1.2.ii プロジェクトジョブの実行

5. 次のようにwalltimeを1時間として36秒間のsleepジョブを実行し、プロジェクトP1に課金します。

```
qsub -P P1 -lwalltime=01:00:00 -- /bin/sleep 36
```

6. ジョブの実行を確認するには次のように指定します。

```
qstat -sw
```

7. Manager1としてログインします。
8. 次のように、プロジェクトP1のクレジットバランスを確認します。そのクレジットバランスは減少しています。

```
amgr checkbalance project -n P1 -p 2022.Q2
```

9. ジョブが終了した後、このバランスを次のように再度確認します。使用されなかった量のクレジットが返却されていることがわかります。

```
amgr checkbalance project -n P1 -p 2022.Q2
```

### 12.3.1.2.iii プロジェクトユーザーではないユーザーによるプロジェクトジョブの実行

10. User2としてログインします。
11. 次のようにwalltimeを2分として10秒間のsleepジョブを実行し、ユーザー User2に課金します。

```
qsub -lwalltime=00:02:00 --/bin/sleep 10
```

12. 次のように、ユーザー User2のクレジットバランスを確認します。そのクレジットバランスは減少しています。

```
amgr checkbalance user -n User2 -p 2022.Q2
```

13. ジョブが終了した後、このバランスを次のように再度確認します。使用されなかった量のクレジットが返却されていることがわかります。

```
amgr checkbalance user -n User2 -p 2022.Q2
```

14. 次のようにwalltimeを2分として10秒間のsleepジョブを実行し、プロジェクトP1に課金します。

```
qsub -P P1 -lwalltime=00:02:00 -- /bin/sleep 10
```

User2はプロジェクトP1に属していないので、このジョブは実行できません。

例12-5： 次のように、すべての標準サービス単位と最も低コストの期間をレポート出力します。

```
amgr report project -n p1
```

### 12.3.1.2.iv マネージャによるプロジェクトレポートの実行

15. Manager1としてログインします:

```
amgr login
```

16. 次のように指定して、プロジェクトP1に関するレポートを入手します。

```
amgr report project -n P1
```

コマンド出力

```
------------------------------------------------------------------------
name | period   | serviceunit | opening_balance | total_credits
------------------------------------------------------------------------
P1   | 2022     | cpu_hrs     | 0.0             | 1000.0

------------------------------------------------------------------------
| total_debits | total_debits_reconciled | total_debits_authorized
------------------------------------------------------------------------
| 0.01         | 1.99                    | 0.0

--------------------------
| net_balance | metadata
--------------------------
| 999.99      | {}
```

## 12.3.2 後払いモードでBudgetsを使用するチュートリアル

### 12.3.2.1 前提条件

PBS Professionalがインストールされ動作する環境で、コンプレックスでジョブの投入と実行ができるアカウントが2つ以上あること。ここの例では、クラスタ名をCluster1、そのユーザーをUser1およびUser2とします。User1は、クラスタCluster1とプロジェクトP1に関連付けられています。

チュートリアルでは、クラスタ名、プロジェクト名、ユーザー名の代わりに各自に独自の名前を使用します。

### 12.3.2.2 Budgetsを使用するチュートリアルの手順

#### 12.3.2.2.i ユーザージョブの実行

17. User1としてログインします。
18. 次のようにwalltimeを2分として10秒間のsleepジョブを実行し、ユーザー User1に課金します。

    **qsub -lwalltime=00:02:00 -- /bin/sleep 10**

19. ジョブが終了した後、ユーザー User1が使用したクレジットを確認します。使用したクレジットが増加しています。

    **amgr checkbalance user -n User1 -p 2022**

#### 12.3.2.2.ii プロジェクトジョブの実行

20. 次のようにwalltimeを1時間として36秒間のsleepジョブを実行し、プロジェクトP1に課金します。

    **qsub -P P1 -lwalltime=01:00:00 -- /bin/sleep 36**

21. ジョブの実行を確認するには次のように指定します。

    **qstat -sw**

22. Manager1としてログインします。
23. ジョブが終了した後、プロジェクトP1が使用したクレジットを確認します。使用したクレジットが増加しています。

    **amgr checkbalance project -n P1 -p 2022**

### 12.3.2.2.iii プロジェクトユーザーではないユーザーによるプロジェクトジョブの実行

24. User2としてログインします。
25. 次のようにwalltimeを2分として10秒間のsleepジョブを実行し、ユーザー User2に課金します。

    **qsub -lwalltime=00:02:00 --/bin/sleep 10**

26. ジョブが終了した後、ユーザー User2が使用したクレジットを確認します。使用したクレジットが増加しています。

    **amgr checkbalance user -n User2 -p 2022**

27. 次のようにwalltimeを2分として10秒間のsleepジョブを実行し、プロジェクトP1に課金します。

    **qsub -P P1 -lwalltime=00:02:00 -- /bin/sleep 10**

    User2はプロジェクトP1に属していないので、このジョブは実行できません。

### 12.3.2.2.iv マネージャによるプロジェクトレポートの実行

28. Manager1としてログインします:

    **amgr login**

29. 次のように指定して、プロジェクトP1に関するレポートを入手します。

    **amgr report project -n P1**

    コマンド出力

```
---------------------------------------------------------------------------
name    | period   | serviceunit   | total_outstanding   | metadata
---------------------------------------------------------------------------
P1      | 2022     | cpu_hrs       | 000.01              | {}
```

# 13 NEC SX-Aurora TSUBASA へのジョブの投入

## 13.1 NEC SX-Aurora TSUBASA の vnode

NEC SX-Aurora TSUBASA におけるハードウェアの基本的な構成単位は、アクセラレータである *ベクトルエンジン*（VE）の一群にオプションの PCIe を介して接続された *ベクトルホスト*（標準の x86 サーバー）です。この構成単位は、1 つまたは複数の NUMA ノードで構成できます。各ユニットは、1 つまたは複数のホストチャネルアダプターを使用して他の構成単位および環境の中で自身以外の部分と通信します。

通信のオーバーヘッドが増加する順番は、最初が 1 つのベクトルエンジン内部、つづいて共有 PCIe を通じた複数のベクトルエンジン間、共通のベクトルホスト上の PCIe を通じた複数のベクトルエンジン間の順となり、最後が別々のベクトルホスト上にある複数のベクトルエンジン間となります。

PBS では、各 PCIe をそれに関連するベクトルホストとベクトルエンジンとともに 1 つの vnode にグループ化することにより、トポロジー対応 vnode が作成されます。PCIe が存在しない場合は、ベクトルホストとそのベクトルエンジンが vnode にグループ化されます。自身の PCIe がない NUMA ノードは自身の vnode に存在します。トポロジーに基づく vnode の作成は、組み込み hook である PBS_sx_aurora の exechost_startup イベントで処理されます。

PBS では、通信のオーバーヘッドが最小になるようにベクトルエンジン上のジョブプロセスをグループ化することにより、トポロジー対応のスケジューリングが試されます。ジョブからベクトルエンジンの要求があると、複数のベクトルエンジン間の通信によるオーバーヘッドを最小限にするために、単一の vnode からベクトルエンジンが割り当てられます。

## 13.2 用語

**HCA**

ホストチャネルアダプター。vnode で使用するネットワーク相互接続。ベクトルホストごとに HCA を 1 つまたは複数にすることができます。

**ベクトルエンジン（VE）**

ベクトルホストに関連付けたアクセラレータ。並列数値演算および / またはベクトル化数値演算を実行します。

**ベクトルホスト（VH）**

標準の x86 サーバー。入出力などのタスクを実行します。

**VE オフロード**

ベクトルホスト上で実行される主な操作は、並列的な数値演算および / またはベクトル化された数値演算をベクトルエンジンにオフロード（委任）することです。オフロードでは、NEC MPI によって VH 上でプロセスが開始され、それらのプロセスによって、そのジョブに割り当てられた VE 上で他のジョブプロセスが開始されます。220 ページのセクション 13.4.3.4「VE オフロードの使用」をご参照ください。

# 13.3 SX-Aurora TSUBASAのリソース

nves

ホストレベルで消費できる整数。チャンクごとのベクトルエンジンの数を指定できます。PBSでは、1つのvnode上で使用できるVEの数をresources_available.nvesで設定します。resources_available.nvesのデフォルト値は、PCIeに接続されたVEの数です。ジョブ要求のデフォルト値は出荷時点でゼロです。管理者が別途設定しない限り、PBSではゼロが割り当てられています。

nhcas

チャンクレベルで消費できない整数。ジョブのチャンクで要求があった場合は、そのチャンクに応じて環境変数_NEC_HCA_LIST_IOと_NEC_HCA_LIST_MPIが設定されます。チャンクについて要求されない場合、PBSは、ホスト上のすべてのHCAが含まれるように_NEC_HCA_LIST_IOと_NEC_HCA_LIST_MPIを設定します。

ve_mem

ジョブ全体の文字列。ジョブによって使用されるベクトルエンジン上の最大メモリ量を報告するために使用されます。

ve_cput

ジョブ全体の文字列。ジョブによって使用されるベクトルエンジン上の合計CPU時間（秒単位）を報告するために使用されます。

ncpus

PBSは、各vnode上のresources_available.ncpusの値を、“（ホスト全体上のCPU数÷ホスト上のvnode数）-vnode上のVE数”に設定します。VEあたり1つのCPUがVEOSデーモン用に予約されています。

mem

PBSは、各vnode上のresources_available.memの値を、ホスト全体のメモリ量をホスト上のvnode数で割った値に設定します。

# 13.4 NEC SX-Aurora TSUBASA上でのジョブの実行

## 13.4.1 NEC SX-Aurora TSUBASA上でのリソースの要求

ncpusリソースを使用してVH上でCPUを要求できます。VH上で実行するプロセス向けにncpusを要求しないと、PBSによってデフォルトの個数のCPUがジョブに割り当てられます。この値は一般的に1です。

チャンクごとにVEを要求するにはnvesリソースを使用します。

チャンクごとにHCAを要求するにはnhcasリソースを使用します。PBSは、必ずジョブのVEに最も近い位置にあるHCAをそのジョブに割り当てようとします。

### 13.4.1.1 HCAの要求に関する制約事項

HCAが1つのチャンクとHCAが2つのチャンクのように、それぞれ異なる個数のHCAを持つ複数のチャンクを要求する場合は、その要求に`-l place=scatter`を追加します。そのようにしないとパフォーマンスが低下することがあります。`-l place=scatter`を指定しないと、各ベクトルで最初のチャンクにベクトルホストによって割り当てられたnhcasの値が、そのベクトルのすべてのチャンクに適用されます。

## 13.4.2 デフォルトのプロセス分散

NEC SX-Aurora TSUBASA では、mpirun コマンドを使用して、必要に応じ、NEC_PROCESS_DIST 環境変数を通じてプロセス分散を指定します。NEC MPI では、VH 上または VE 上（あるいはその両方）でプロセスが直接開始され、mpirun コマンドラインに応じてプロセスのランク順序が設定されます。デフォルトのプロセス分散を使用する mpirun コマンドの例を以下に示します：

例 13-1： 32個の ve.out プロセスを VE 上で実行する

```
mpirun -np 32 ve.out
```

例 13-2： 1つの vh.out プロセスを VH 上で実行し、12個の ve.out プロセスを VE 上で実行する

```
mpirun -vh -np 1 vh.out : -np 12 ve.out
```

ランク 0 のプロセスは VH 上で vh.out を実行し、ランクが 1 から 12 までの各プロセスは VE 上で ve.out を実行します。

例 13-3： 4 つの VE を持つ 1 つのチャンクで 4 つのプロセスを実行し、4 つの VE を持つ 1 つのチャンクで 13 個のプロセスを実行する

```
qsub -lselect=1:nves=4:mpiprocs=4+1:nves=4:mpiprocs=13
```

ジョブ内部のスクリプト：

```
mpirun -np 17 ve.out
```

### 13.4.2.1 PBS による、チャンクでの VE プロセスの分散

NEC_PROCESS_DIST を指定しないと、チャンクのプロセス数がチャンクの VE 数の整数倍であるかどうかに応じて、本項の以下の説明のように、PBS によってチャンクの中でプロセスが分散されます。各プロセスは NEC MPI によって直接開始されます。プロセス数が VE 数の整数倍であれば、それを“完全な分散”と呼びます。

#### 13.4.2.1.i 完全な分散

チャンクのプロセス数が VE 数の整数倍であれば、そのチャンクの中では各 VE に同数のプロセスが配置されます。たとえば、プロセス数が 12 で VE 数が 4 であれば各 VE に 3 つのプロセスが配置されます。

チャンクにあるすべての VE にわたって可能な限り均一に VE プロセスを分散するには、そのチャンクを要求するときに、mpiprocs リソースを使用して VE の数とプロセスの数を指定します。

例 13-4： 6 つのプロセスと 2 つの VE を持つチャンクが 1 つあり、各 VE で 3 つの VE プロセスを実行するには以下のように指定します。

ジョブ内部のスクリプト：

```
mpirun -np 6 ve.out
```

ジョブを投入：

```
qsub -l select=ncpus=2:nves=2:mpiprocs=6 ...
```

#### 13.4.2.1.ii 不完全な分散

チャンクのプロセス数がVE数の整数倍ではない場合、PBSで認識されたトポロジー上の順序で上位にあるVEほど、より多くのプロセスが割り当てられます。PBSでは、プロセス数をVE数で除算した値よりも大きい最小の整数個数のプロセスがこのようなVEに配置され、残りが次の順序のVEに配置されます。たとえば、プロセス数が10でVE数が4であれば、最上位から3つのVEに3つのプロセス、最下位のVEに1つのプロセスがそれぞれ配置されます。一方、プロセス数が5でVE数が4であれば、最上位から2つのVEに2つのプロセス、その次のVEに1つのプロセスがそれぞれ配置され、最下位のVEにはプロセスが配置されません。

例13-5： 2つのCPUと3つのVEを要求し、mpiprocsに8を指定したとします。この例では、最上位のVEと2番目のVEがそれぞれ3つのVEプロセスを受け取り、3番目のVEが2つのプロセスを受け取ります。ジョブ内部のスクリプト：

```
mpirun -np 8 ve.out
```

ジョブを投入：

```
qsub -l select=ncpus=2:mpiprocs=8:nves=3 ...
```

VEへのプロセスの分散

```
ve=0
ve=0
ve=0
ve=1
ve=1
ve=1
ve=2
ve=2
```

## 13.4.3 プロセス分散の指定

必要に応じ、NEC_PROCESS_DIST環境変数を使用してプロセスの実行場所を指定できます。チャンク別にプロセスの分散を指定します。NEC MPIから直接プロセスを開始するか、VH上で実行するプロセスから、VE上で実行する任意のプロセスを開始することを指定できます。

各チャンクで以下の操作が可能です：

- チャンクにあるVEごとのプロセス数を指定する。そのためには、以下のいずれかの方法を使用します：
  - 各VE上で直接開始するプロセスの数として、それぞれ異なる値を指定する。セクション13.4.3.1「チャンクにあるすべてのVEへのプロセス配置指定」をご参照ください。
  - 最初のVEにのみプロセス数を指定し、他のVEのプロセス数は暗黙的に決まるようにすることで、直接開始するプロセスの分散を均一にする。セクション13.4.3.2「チャンクにあるすべてのVEへのプロセス分散の複製」をご参照ください。
  - PBSを使用して、直接開始するプロセスの分散を可能な限り均一にする。セクション13.4.2.1「PBSによる、チャンクでのVEプロセスの分散」をご参照ください。
  - VEオフロードを使用して、VE上でプロセスを間接的に開始する。セクション13.4.3.4「VEオフロードの使用」をご参照ください。
- ベクトルホスト上でどのプロセスを実行するかを指定する。セクション13.4.3.3「VHへのプロセスの配置」をご参照ください。

また、同等な複数のチャンクで構成するグループがある場合、それらの最初のチャンクへのプロセス配置を指定すると、その配置を他のチャンクにPBSで複製できます。このように同等のチャンクで構成するグループが複数あり、グループとしてはそれぞれが互いに異なる場合は、グループごとに最初のチャンクへのプロセス配置のみを指定すればすみます。セクション13.4.3.5「複数のチャンクへの同じプロセス分散の複製」をご参照ください。

ここまではプロセス分散の概要ですが、以下の各項ではその分散の構文を詳しく説明しています。複数のチャンクへのプロセス分散の構文は次のとおりです。

*NEC_PROCESS_DIST = <チャンク1 での分散>+<チャンク2 での分散> +...+ <チャンクN での分散>*

*<チャンクでの分散>*は次のように指定します。

*[<ベクトルホストにあるプロセスの数>]:[<最初のVE にあるプロセスの数>][:<以降の各VE にあるプロセスの数>]*

各チャンクへの指定は、プラス記号（“+”）で区切ります。

## 13.4.3.1 チャンクにあるすべてのVE へのプロセス配置指定

各チャンクで、NEC MPI から直接開始できるVE プロセスの数を、必要に応じてVE ごとに指定できます。各VE で開始するVE プロセスの数をコロンで区切って記述します。

単一のチャンクでVE へのプロセス分散を指定する構文は以下のとおりです。

*NEC_PROCESS_DIST=<最初のVE のプロセス数>:<2 番目のVE のプロセス数> :...: <n 番目のVE のプロセス数>*

例13-6： 6つのプロセスと3つのVE があり、最初のVE で1つのプロセス、2番目のVE で3つのプロセス、3番目のVE で2つのプロセスをそれぞれ実行するには以下のように指定します。

```
qsub -lselect=1:ncpus=2:nves=3:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2
```

### 13.4.3.1.i チャンクにあるすべてのVE へのプロセス配置に関する制限事項と注意事項

1つのチャンクで複数のVE でのプロセス数を指定する場合は、すべてのVE でのプロセス数を指定する必要があります。

## 13.4.3.2 チャンクにあるすべてのVE へのプロセス分散の複製

チャンクにある最初のVE についてのみ、NEC MPI で直接開始するVE プロセスの数を指定すると、その分散をチャンクにある残りのVE にPBS で複製できます。得られる分散は、チャンクにあるプロセスの数がチャンクにあるVE の数の整数倍であるかどうかに応じて異なります。プロセス数がVE 数の整数倍であれば、それを“完全な分散”と呼びます。最初のVE にのみプロセス数を指定した場合は、そのチャンクにある他のVE へのプロセス配置を*暗黙的に指定*したことになります。

単一のチャンクですべてのVE にプロセスの分散を複製する構文は以下のとおりです。

*NEC_PROCESS_DIST=<最初のVE のプロセス数>*

### 13.4.3.2.i 完全な分散の暗黙的な指定

すべてのVE にわたってNEC MPI でプロセスを直接開始することで完全な分散を暗黙的に指定できます。

例13-7： 6つのプロセスと2つのVE を持つチャンクが1つあり、各VE で3つのVE プロセスを実行するには以下のように指定します。

```
-l select=ncpus=2:nves=2:mpiprocs=6 -v NEC_PROCESS_DIST=3
```

### 13.4.3.2.ii 不完全な分散の暗黙的な指定

プロセス数がVE 数の整数倍ではない場合、PBS で使用される以下の手順に従うことで、NEC MPI によるプロセスの直接開始を通じて不完全な分散を暗黙的に指定できます。最初のVE には、他のVE よりも多くのプロセス数を指定します（プロセス数をVE 数で除算した値よりも大きい最小の整数）。たとえば、プロセス数が7でVE 数が3であれば、最初のVE のプロセス数として1や2ではなく3を指定します。

例13-8： 1つのチャンクに11個のプロセスと3つのVE があり、最初と2番目のVE で4つのVE プロセス、3番目のVE で3つのVE プロセスをそれぞれ実行するには以下のように指定します。

```
-l select=ncpus=2:nves=3:mpiprocs=11 -v NEC_PROCESS_DIST=4
```

## 13.4.3.3 VHへのプロセスの配置

ジョブにあるチャンクごとに、VHプロセスの数を指定することで、1つまたは複数のプロセスをVHに配置できます。VHプロセス数の前に文字“S”を使用します。VEでプロセスを開始する方法にこの記述を組み合わせることができます（直接開始またはVEオフロード）。

VHへのプロセス分散を指定する構文は以下のとおりです。

*NEC_PROCESS_DIST=**S**<VHプロセス数>*

単一のチャンクでVHとVEにプロセスを分散する構文は以下のとおりです（VHプロセス数はチャンク分散の任意の位置に記述できます。ここでは最初の位置に記述しています）。

*NEC_PROCESS_DIST=**S**<VHプロセス数>:<最初のVEのプロセス数>[:<2番目のVEのプロセス数> :...: <n番目のVEのプロセス数>]*

複数のチャンクは以下のようにプラス記号で区切ります。

*NEC_PROCESS_DIST=**S**<VHプロセス数>:<最初のVEのプロセス数>[:<2番目のVEのプロセス数> :...: <n番目のVEのプロセス数>]**+S**<VHプロセス数>:<最初のVEのプロセス数>[:<2番目のVEのプロセス数> :...: <n番目のVEのプロセス数>]*

例13-9： プロセスが5つ、VEが1つあるとします。最初の2つのプロセスをVHで実行し、最後の3つのプロセスをVEで実行するには以下のように指定します。

```
qsub -lselect=1:ncpus=2:nves=1:mpiprocs=5 -v NEC_PROCESS_DIST=S2:3
```

### 13.4.3.3.i VHへのプロセスの配置に関する制限事項と注意事項

- VHへのプロセス配置を指定する場合は、それらのプロセスに使用するCPUを確保するためにncpusリソースを要求する必要があります。
- VHプロセスの分散を指定できるのはチャンクごとに1回のみです。
- VHプロセスの指定では、1つ以上のVHプロセスを指定する必要があります。指定するVHプロセス数をゼロにすることはできません。

## 13.4.3.4 VEオフロードの使用

オフロードでは、NEC MPIによってベクトルホスト上でプロセスが開始され、そのプロセスにより、ベクトルホストに接続されてジョブに割り当てられたベクトルエンジン上で、並列数値演算またはベクトル化数値演算（あるいはその両方）が実行（オフロード）されます。VEにオフロードされたプロセスが、NEC MPIによって直接開始されることはありません。

オフロードされていないVEプロセスは、VE上でNEC MPIによって直接開始されます。

VEオフロードはチャンク単位で適用されるので、オフロードを使用するチャンクを指定する必要があります。どのチャンクでもVEオフロードの使用は任意ですが、VEへのオフロードとNEC MPIによるVE上での直接開始を同じチャンクの中で混用することはできません。

VEにプロセスをオフロードする手順は次のとおりです：

- nvesリソースを使用して、オフロードに必要なVEの数を指定します。
- NEC_PROCESS_DIST環境変数を使用して、1つまたは複数のプロセスをVH上で開始することを指定します。
- VE上ではどのプロセスもNEC MPIで直接開始しないことを指定します。
- `mpirun`コマンドで指定しているプロセス数とmpiprocsの合計数が等しいことを確認します。NEC MPIではVH上でのみプロセスが開始され、オフロードされているプロセスは開始されないからです。

VE の数に関係なく、オフロードの指定には固有の構文を使用します。論理的に等価な構文も機能するように思えますが、実際には機能しません。最初の VE にのみゼロを指定し、他の VE には何も指定しません。VE オフロードの構文は以下のとおりです。

*NEC_PROCESS_DIST=S<VH プロセス数>:0*

例 13-10： VH で 2 つのプロセスを実行し、VE に 3 つのプロセスをオフロードします。接続されている VE 上でプロセス vh.out から ve.out を開始するには以下のように指定します。

```
qsub -l select=ncpus=2:mpiprocs=2:nves=2 -v NEC_PROCESS_DIST=S2:0
```

ジョブ内部のスクリプト：

```
mpirun -vh -np 2 vh.out
```

#### 13.4.3.4.i VE オフロードに関する制限事項と注意事項

VE オフロードの構文は、***S<VH プロセス数>:0***, not S<VH プロセス数>:0:0:0 or S<VH プロセス数> です。末尾の 2 つの式は論理的に等価に見えますが、両方を記述する必要があります。

### 13.4.3.5 複数のチャンクへの同じプロセス分散の複製

同等な複数のチャンクで構成するグループがある場合、それらの最初のチャンクへのプロセス配置を指定すると、その配置を他のチャンクに PBS で複製できます （同等なチャンクごとに同じ指定を繰り返してもかまいませんが、その必要はありません）。複数の同等なチャンクで構成する 1 つのグループにプロセス分散を指定する構文は次のとおりです。

*NEC_PROCESS_DIST = <グループにあるすべてのチャンクへの分散>*

例 13-11： 2 つの等価なチャンクがあり、それぞれに 6 つのプロセスと 3 つの VE があるとします。これらのチャンクごとに、最初の VE で 1 つのプロセス、2 番目の VE で 3 つのプロセス、3 番目の VE で 2 つのプロセスをそれぞれ実行します。最初のチャンクへの分散を指定し、その分散を残りの同等なチャンクへ PBS で複製するには以下のように指定します。

```
qsub -lselect=2:ncpus=2:nves=3:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2
```

このような同等なチャンクのグループが複数あり、グループとしてはそれぞれが互いに異なる場合は、グループごとに最初のチャンクへの分散のみを指定すればすみます。複数の同等なチャンクで構成する複数のグループにプロセス分散を指定する構文は次のとおりです。

*NEC_PROCESS_DIST = <グループ 1 のチャンク分散>+<グループ 2 のチャンク分散> +...+ <グループ N のチャンク分散>*

例 13-12： チャンクのグループが 2 つあるとします。

最初のグループは、それぞれ 6 つのプロセスと 3 つの VE を持つ 2 つのチャンクで構成されています。最初の VE で 1 つのプロセス、2 番目の VE で 3 つのプロセス、3 番目の VE で 2 つのプロセスをそれぞれ実行します。

2 つのチャンクで構成する 2 番目のグループには、2 つの VE に均等分散された 6 つのプロセスがあります。

これらのグループごとに最初のチャンクへの分散を指定し、その分散を各グループの他のチャンクへ PBS で複製するには以下のように指定します。

```
qsub -lselect=2:ncpus=2:nves=3:mpiprocs=6+2:nves=2:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2+3
```

### 13.4.3.6 プロセス分散の指定例

以下の各例では、VH 上で実行するプロセスの名前を vh.out、VE 上で実行するプロセスの名前を ve.out とします。

例 13-13： 6 つのプロセスと 3 つの VE があり、最初の VE で 1 つのプロセス、2 番目の VE で 3 つのプロセス、3 番目の VE で 2 つのプロセスをそれぞれ実行するには以下のように指定します。

```
qsub -lselect=1:ncpus=2:nves=3:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2
```

ジョブ内部のスクリプト：

```
mpirun -np 6 ve.out
```

例13-14： 6つのプロセスと2つのVEを持つチャンクが1つあり、各VEで3つのVEプロセスを実行するには以下のように指定します。

```
qsub -l select=ncpus=2:nves=2:mpiprocs=6 -v NEC_PROCESS_DIST=3
```

ジョブ内部のスクリプト：

```
mpirun -np 6 ve.out
```

例13-15： 2つのチャンクがあるとします。

最初のチャンクには6つのプロセスと3つのVEがあり、最初のVEで1つのプロセス、2番目のVEで3つのプロセス、3番目のVEで2つのプロセスをそれぞれ実行します。

2番目のチャンクには6つのプロセスと2つのVEがあり、各VEで3つのVEプロセスを実行します（13-13と13-14の例の組み合わせ）。このための構文は以下のようになります。

```
qsub -lselect=1:ncpus=2:nves=3:mpiprocs=6+1:ncpus=2:nves=2:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2+3
```

ジョブ内部のスクリプト：

```
mpirun -np 12 ve.out
```

例13-16： 3つのチャンクがあるとします。

最初のチャンクには5つのプロセスと1つのVEがあります。最初の2つのプロセスをVHで実行し、最後の3つのプロセスをVEで実行します。

2番目のチャンクには6つのプロセスと3つのVEがあり、VHで1つのプロセス、VEで5つのプロセスをそれぞれ実行します。

3番目のチャンクではVEで4つのプロセスを実行します。このための構文は以下のようになります。

```
qsub -l select=1:ncpus=2:nves=1:mpiprocs=5+1:ncpus=2:nves=3:mpiprocs=6+1:ncpus=2:nves=1:mpiprocs=4
    -v NEC_PROCESS_DIST=S2:3+2:1:S1:2+4
```

ジョブ内部のスクリプト：

```
mpirun -vh -np 2 vh.out : -np 6 ve.out : -vh -np 1 vh.out : -np 6 ve.out
```

例13-17： 2つのチャンクがあるとします。

最初のチャンクでは、接続されているVEで3つのプロセスve.outをプロセスvh.outから開始します。

2番目のチャンクでは2つのVEで4つのプロセスを実行します。このための構文は以下のようになります。

```
qsub -l select=ncpus=2:mpiprocs=2:nves=2+1:ncpus=2:nves=2:mpiprocs=4 -v NEC_PROCESS_DIST=S2:0+2
```

ジョブ内部のスクリプト：

```
mpirun -vh -np 2 vh.out : -np 4 ve.out
```

例13-18： チャンクのグループが3つあるとします。

最初のグループは、それぞれ6つのプロセスと3つのVEを持つ2つのチャンクで構成されています。最初のVEで1つのプロセス、2番目のVEで3つのプロセス、3番目のVEで2つのプロセスをそれぞれ実行します。

2つのチャンクで構成する2番目のグループには、2つのVEに均等分散された6つのプロセスがあります。

3番目のグループには3つのチャンクがあり、VHで1つのプロセス、VEで2つのプロセスをそれぞれ実行します。

これらのグループごとに最初のチャンクへの分散を指定し、その分散を各グループの他のチャンクへPBSで複製するには以下のように指定します。

```
qsub -lselect=2:ncpus=2:nves=3:mpiprocs=6+2:nves=2:mpiprocs=6+3:ncpus=2:nves=1:mpiprocs=3  -v
    NEC_PROCESS_DIST=1:3:2+3+S1:2
```

ジョブ内部のスクリプト：

```
mpirun -np 24 ve.out : -vh -np 1 vh.out : -np 2 ve.out : -vh -np 1 vh.out : -np 2 ve.out : -vh -np 1
    vh.out : -np 2 ve.out
```

例13-19： 3つのチャンクがあるとします。

最初のチャンクには5つのプロセスと1つのVEがあります。最初の2つのプロセスをVHで実行し、最後の3つのプロセスをVEで実行します。

2番目のチャンクには3つのプロセスがあり、そのすべてをVHで実行します。

3番目のチャンクではVEで4つのプロセスを実行します。このための構文は以下のようになります。

```
qsub -l select=1:ncpus=2:nves=1:mpiprocs=5+1:ncpus=3:mpiprocs=3+1:ncpus=2:nves=1:mpiprocs=4 -v
    NEC_PROCESS_DIST=S2:3+S3+4
```

ジョブ内部のスクリプト：

```
mpirun -vh -np 2 vh.out : -np 3 ve.out : -vh -np 3 vh.out : -np 4 ve.out
```

### 13.4.3.7 プロセスの分散指定に関する制限事項と注意事項

- ジョブにNEC_PROCESS_DIST環境変数を使用する場合は、そのジョブのすべてのチャンクでmpiprocsリソースを要求する必要があります。
- 各ジョブでselect指定のチャンク数がNEC_PROCESS_DISTの値と同じであることを確認します。
- それぞれのチャンクで、指定したVEプロセス数とVHプロセス数の合計がmpiprocsの合計と等しい必要があります。219ページのセクション13.4.3.2.i「完全な分散の暗黙的な指定」の説明にあるように、均一な分散を暗黙的に指定できます。
- ジョブにあるいずれかのVEに分散を指定する場合は、そのジョブのすべてのチャンクにあるすべてのVEに分散を指定する必要があります。219ページのセクション13.4.3.2.i「完全な分散の暗黙的な指定」の説明にあるように、均一な分散を暗黙的に指定できます。
- VEオフロードを使用していない場合、どのVEにもプロセス数としてゼロを指定することはできません。

# 13.5 NEC SX-Aurora TSUBASA上のジョブアカウンティング

PBSは、アカウンティングレコードを書き込む際に、nvesをResource_List.nvesとresources_assigned.nvesの両方に記録します。PBSは、ジョブのresources_used属性の値の一部として、ve_memとve_cputも書き込みます。

# 13.6 NEC MPIの環境変数

PBSでは、ベクトルエンジンを要求するジョブを開始する前に、実行ホストごとに以下の環境変数が設定されます：

_VENODELIST

VH上でこのジョブに割り当てられているVE番号のリスト。

たとえば、PBSによってVH上でジョブにVE番号0～3が割り当てられていると、`export _VENODELIST="0 1 2 3"`が得られます。VH上でジョブにVEが割り当てられていない場合は`export _VENODELIST=""`が得られます。

VE_NODE_NUMBER

VH上でこのジョブに割り当てられている最小のVE番号。

たとえば、PBSによってVHにVE番号0～3が割り当てられていると、`export VE_NODE_NUMBER="0"`が得られます。VH上でジョブにVEが割り当てられていない場合は`export VE_NODE_NUMBER=-1`が得られます。

_NECMPI_VE_NUM_NODES

このジョブに割り当てられていて、NEC MPIで直接使用するベクトルエンジンの数。VEオフロードで使用するベクトルエンジンではありません。

たとえば、PBSによってVH上のジョブにVE番号0～3が割り当てられていると、`export _NECMPI_VE_NUM_NODES=4`が得られます。

_NECMPI_VE_NODELIST

このジョブに割り当てられていて、NEC MPIで直接使用するVE番号のスペース区切りリスト。VEオフロードで使用するベクトルエンジンではありません。

たとえば、PBSによってVH上のジョブにVE番号0～3が割り当てられていると、`export _NECMPI_VE_NODELIST="0 1 2 3"`が得られます。

_NEC_HCA_LIST_IO

このベクトルホスト上でこのジョブに割り当てられているHCAリストのスペース区切りリスト。

HCAリストは、各VEでこのジョブに割り当てられているHCAのコンマ区切りリストです。

ジョブのチャンクにジョブからnhcasの要求があると、そのチャンクに応じてPBSで_NEC_HCA_LIST_IOが設定されます。チャンクにnhcasが要求されないと、ホスト上ですべてのHCAを扱うように_NEC_HCA_LIST_IOが設定されます。

たとえば、2つのVEと2つのHCA（ラベルをmlx5_0:1およびmlx5_1:1とします）がVHに接続されていて、各VEで両方のHCAを使用する場合は、_NEC_HCA_LIST_IO=mlx5_0:1,mlx5_1:1 mlx5_0:1,mlx5_1:1となります。

_NEC_HCA_LIST_MPI

このジョブで使用する、VEごとのHCAリストのスペース区切りリスト。オフロードで使用するHCAリストではありません。環境変数_NEC_HCA_LIST_IOの説明をご参照ください。

ジョブのチャンクにジョブからnhcasの要求があると、そのチャンクに応じてPBSで_NEC_HCA_LIST_MPIが設定されます。チャンクにnhcasが要求されないと、ホスト上ですべてのHCAを扱うように_NEC_HCA_LIST_MPIが設定されます。

たとえば、2つのVEと2つのHCA（ラベルをmlx5_0:1およびmlx5_1:1とします）がVHに接続されていて、各VEで両方のHCAを使用すると同時に、一方のVEをオフロードに使用する場合は_NEC_HCA_LIST_IO=mlx5_0:1,mlx5_1:1となります。

# 14 PBS Professionalでの MLSの使用

## 14.1 SELinux PBS Professionalについて

このバージョンの PBS Professional では、RHEL 7上で1つまたは複数のMLS ポリシーで使用される SELinux 強制モードをサポートする個別のソフトウェアパッケージを提供しています。本章では、SELinux PBSの使用方法のみを取り上げます。

## 14.2 ジョブ投入の要件

PBSへのジョブの投入は、このバージョンのPBSコマンドを実行するマシンから実行する必要があります。このバージョンのPBSにはsecurity_contextジョブ属性があります。この属性がないと、エラーが発生してサーバーのメッセージがログに記録され、ジョブが拒否されます。

## 14.3 ジョブの表示と操作

PBSでユーザーがそのジョブに対して問い合わせや操作を実行するには、そのジョブのsecurity_context属性が要求元のセキュリティコンテキストに一致している必要があります。要求元のクレデンシャルがそのジョブのクレデンシャルよりも優位であるだけでは不十分です。

### 14.3.1 セキュリティコンテキストの確認

PBSでは、ジョブのセキュリティコンテキストがそのジョブのsecurity_context属性に書き込まれます。たとえば、ジョブとユーザーのセキュリティコンテキストを比較するには以下の手順に従います：

次のようにジョブのセキュリティコンテキストを探し出します。

```
bash-4.2$ qstat -f | grep security
security_context = user_u:user_r:user_t:s3:c1,c2
```

次のようにユーザーのセキュリティコンテキストを探し出します。

```
bash-4.2$ id -Z
user_u:user_r:user_t:s3:c1,c2
```

## 14.4 削除したジョブのクレデンシャル

`qdel` または `qdel -x` によってジョブが削除されると、ジョブのクレデンシャルは、PBS が使用されていない場合と同じように処理されます。

# 14.5 注意事項

サイトでポリインスタンス化を使用していて、ジョブの実行中に MoM (実行サービス プロセス) が終了するか、終了するように指示された場合、ジョブの終了後にそれらのジョブ用に作成されたディレクトリやファイルを手動でクリーンアップする必要がある場合があります。

# 14.6 エラーとロギング

## 14.6.1 ロギング

PBSでは、以下のメッセージがサーバーのログまたはMoMのログ（あるいはその両方）に記録されることがあります。

**表14-1：ログメッセージ**

| メッセージ | ログレベル |
|---|---|
| no security context found（セキュリティコンテキストが見つかりません） | エラー（0x0001） |
| failed to read credential file（クレデンシャルファイルを読み取れませんでした） | エラー（0x0001） |
| job credential <context>（ジョブのクレデンシャル<コンテキスト>） | デバッグ（0x0080） |
| unable to create the job directory（ジョブのディレクトリを作成できません） | エラー（0x0001） |
| the staging and execution directory <dir> already exists（ステージングと実行のディレクトリ<dir>がすでに存在します） | デバッグ3（0x0400） |
| could not set security context for <dir>（<dir>のセキュリティコンテキストを設定できませんでした） | エラー（0x0001） |
| unable to change permissions on staging and execution directory <dir>（ステージングと実行のディレクトリ<dir>に対するアクセス許可を変更できません） | デバッグ（0x0080） |
| unable to open user session（ユーザーセッションを開くことができません） | デバッグ4（0x0800） |
| unable to start job, no security context found or not able to set security context（ジョブを開始できないか、セキュリティコンテキストが見つからないか、セキュリティコンテキストを設定できません） | エラー（0x0001） |
| cannot get current socket context（現在のソケットコンテキストを取得できません） | エラー（0x0001） |
| cannot set socket context for <jobid>（<jobid>のソケットコンテキストを設定できません） | エラー（0x0001） |
| cannot restore socket context（ソケットコンテキストを復元できません） | エラー（0x0001） |
| failed to set file context for <dir>（<dir>のファイルコンテキストを設定できませんでした） | エラー（0x0001） |
| could not save security context（セキュリティコンテキストを保存できませんでした） | デバッグ（0x0080） |
| saved security context（セキュリティコンテキストを保存しました） | デバッグ（0x0080） |
| client connection no security context information present（クライアント接続のセキュリティコンテキスト情報が存在しません） | セキュリティ（0x0020） |

表14-1：ログメッセージ

| メッセージ | ログレベル |
|---|---|
| client connection security context information present, but no job security context information present（クライアント接続のセキュリティコンテキスト情報は存在しますが、ジョブのセキュリティコンテキスト情報が存在しません） | セキュリティ（0x0020） |
| client connection security context information not present, but job security context information present（クライアント接続のセキュリティコンテキスト情報が存在しませんが、ジョブのセキュリティコンテキスト情報は存在します） | セキュリティ（0x0020） |
| client connection security context information job security context information do not match（クライアント接続とジョブでセキュリティコンテキスト情報が一致しません） | セキュリティ（0x0020） |

### 14.6.2 エラー

PBSでは、ジョブ投入者の標準エラー出力に以下のエラーメッセージが書き込まれることがあります。

```
"pbs_iff:  fgetfilecon(1) returned -1"
"pbs_iff:  filecon overflow"
"pbs_iff:  malloc extendarg failed"
```

## 14.7 SELinuxのドキュメント

- DIRECTOR OF CENTRAL INTELLIGENCE DIRECTIVE 6/3 PROTECTING SENSITIVE COMPARTMENTED INFORMATION WITHIN INFORMATION SYSTEMS
  http://www.fas.org/irp/offdocs/DCID_6-3_20Manual.htm
- Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide
  https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/index
- Getting Started with Reference Policy
  http://oss.tresys.com/projects/refpolicy/wiki/GettingStarted

# 15 プロビジョニングの使用

PBSでは、プロビジョニングが設定されたvnode上で、OSまたはアプリケーションの自動プロビジョニングを使用できます。使用可能だが実行されていないOSや、インストールされていないアプリケーションをジョブが要求した場合、PBSによって該当するOSまたはアプリケーションがvnodeにプロビジョニングされます。

## 15.1 用語の定義

**AOE**

vnode上の環境。vnodeをプロビジョニングした後の環境、または既存の環境のいずれか

**プロビジョニング**

OSまたはアプリケーションをインストールすること、またはインストールや設定を行うスクリプトを実行すること

**プロビジョニングされたvnode**

プロビジョニング処理によってOSまたはアプリケーションがインストールされたvnode、またはスクリプトが実行されたvnode

## 15.2 プロビジョニングの動作

プロビジョニングは、プロビジョニングが有効になっているvnode上でのみ実行できます。有効かどうかはprovision_enable属性で示されます。

プロビジョニングでは次の処理を実行できます。

- OSまたはアプリケーションを直接インストール
- 設定またはインストールを行うスクリプトの実行

vnodeのresources_available.aoe属性で示した使用可能なAOEのリストを使用して、各vnodeにはプロビジョニングが個別に設定されます。

各vnodeのcurrent_aoe属性は、vnodeの現在のAOEを示します。スケジューラは、各vnodeのaoeリソースおよびcurrent_aoe属性を照会し、プロビジョニングを行うvnodeをジョブごとに判断します。

プロビジョニングはインタラクティブジョブで使用できます。

ジョブのwalltimeクロックは、ジョブ用のプロビジョニングが終了した時点で開始します。

## 15.2.1 vnodeをプロビジョニングする要因

AOEはジョブまたは予約に対して要求できます。ジョブでAOEを要求する場合、要求したAOEが動作しているvnode上でジョブが実行されます。予約でAOEを要求する場合、要求したAOEを使用可能なvnodeが予約されます。このAOEを要求しているジョブを実行する場合にのみ、予約済みのvnode上でAOEがインスタンス化されます。

AOEを要求する各ジョブをスケジューラで実行する場合、スケジューラではジョブの要件を満たすvnodeを探すか、要求されたvnodeをプロビジョニングします。たとえば、vnodeセットでSLESを利用でき、その他の要件がジョブに適している場合、ジョブにはSLESを要求できます。その場合は、ジョブを開始する前にこのvnodeセット上で動作していたOSの種類にかかわらず、ジョブの実行を開始する際にはSLESが稼働しています。

## 15.2.2 AOEの使用

ジョブでAOEを要求する場合、ユーザーのサイトで設定されているAOEのいずれかを要求する必要があります。たとえば、vnodeで利用可能なAOEが“*rhel*”および“*sles*”である場合、要求できるのはこの2つのみで、“*suse*”は要求できません。

ジョブの実行では、要求されたAOEをプロビジョニングによって提供することも、AOEがすでに要求を満たしていればこれを提供することもできます。ジョブの一部のチャンクは、要求されたAOEとすでに一致しているプロビジョニングできないvnodeで実行でき、一部のチャンクは、要求されたAOEに一致するようにプロビジョニングすることが可能なvnodeで実行できます。

特定のAOEを備えたvnodeの予約を要求することができます。このようにして、特定のAOEを必要としているジョブをこの予約に対して投入できます。これにより、特定のAOEを必要としているジョブは、そのAOEを備えたvnode上で実行されることが保証されます。

予約ごとに最大で1つのAOEを指定できます。この予約内で実行されるジョブでは、予約に要求されたAOEとは異なるAOEを要求できません。つまり、ジョブではAOEを要求しないか、予約と同じAOEを要求するかのいずれかの場合に、予約内でジョブを実行できます。

## 15.2.3 Jobのサブステータスとプロビジョニング

プロビジョニング処理中のジョブは、サブステータスが*provisioning*になります。このサブステータスについて、以下で説明します。

プロビジョニング
: ジョブは、要求したAOEがvnodeにプロビジョニングされるのを待機中です。整数値は*71*です。ジョブのサブステータスのリストについては、『PBS Professional Reference Guide』の「Job Substates」（361ページ）をご参照ください。

以下の表に、プロビジョニングのイベントがジョブのステータスとサブステータスに与える影響を示します。

**表15-1：プロビジョニングのイベントとジョブのステータス／サブステータス**

| イベント | ジョブの初期ステータス、サブステータス | ジョブの結果ステータス、サブステータス |
|---|---|---|
| ジョブの投入 | | *Queued（キューで待機中）、選択は可能* |
| プロビジョニングの開始 | *Queued（キューで待機中）、Queued（キューで待機中）* | *Running（実行中）、Provisioning（プロビジョニング中）* |
| プロビジョニングの開始に失敗 | *Queued（キューで待機中）、Queued（キューで待機中）* | *Held（保留）、Held（保留）* |

表15-1：プロビジョニングのイベントとジョブのステータス／サブステータス

| イベント | ジョブの初期ステータス、サブステータス | ジョブの結果ステータス、サブステータス |
|---|---|---|
| プロビジョニングに失敗 | *Running（実行中）*、*Provisioning（プロビジョニング中）* | *Queued（キューで待機中）*、*Queued（キューで待機中）* |
| プロビジョニングに成功してジョブを実行 | *Running（実行中）*、*Provisioning（プロビジョニング中）* | *Running（実行中）*、*Running（実行中）* |
| 内部エラーの発生 | *Running（実行中）*、*Provisioning（プロビジョニング中）* | *Held（保留）*、*Held（保留）* |

# 15.3 要件と制約

## 15.3.1 ホストの制約

### 15.3.1.1 単一vnodeのホストのみ

PBSでは単一vnodeのホストのみをプロビジョニングします。vnodeが複数あるホストでは、プロビジョニングを使用しないでください。

### 15.3.1.2 サーバーホストはプロビジョニングできない

サーバーホストはプロビジョニングできません。サーバーホスト上でMoMは実行できますが、MoMのvnodeはプロビジョニングできません。vnode属性provision_enable、resources_available.aoe、およびcurrent_aoeは、サーバーホスト上で設定できません。

## 15.3.2 AOEの制約

vnode上で一度にインスタンス化できるAOEは１つのみです。

ジョブで要求できるaoeリソースの種類は１つのみです。たとえば、受け入れ可能なジョブは、次のように要求できます。

```
-l select=1:ncpus=1:aoe=suse+1:ncpus=2:aoe=suse
```

### 15.3.2.1 vnodeジョブの制約

以下のジョブを持つvnodeは、プロビジョニングの対象として選択されません。

- 実行中の複数のジョブ
- 中断されたジョブ
- バックフィルジョブ

### 15.3.2.2 プロビジョニングジョブの制約

AOEを要求するジョブはバックフィルされません。

#### 15.3.2.3 vnode予約の制約

vnodeに確認済みの予約があり、その予約の開始時刻がジョブMyJobの終了時刻よりも前に設定されている場合、このvnodeは、ジョブMyJobに対するプロビジョニングの対象として選択されません。

vnodeに確認済みの予約R2があり、R1の発生とR2の発生の時刻が重複し、この２つの発生がvnodeを共有しながら異なるAOEを要求している場合、このvnodeは、予約R1内のジョブに対するプロビジョニングの対象として選択されません。

### 15.3.3 ジョブの要件

#### 15.3.3.1 AOEを要求する場合は必ずすべてのチャンクで同じAOEを使用する

ジョブの任意のチャンクがAOEを要求する場合、明示的にAOEを要求しなくても、すべてのチャンクでそのAOEを使用する必要があります。たとえば、ジョブで次のように要求する場合、

```
-l select=2:ncpus=1:aoe=suse+4:ncpus=2
```

すべてのチャンクで*suse* AOEを使用する必要があります。

AOEを要求するジョブを予約に投入する場合は、その予約でも同じAOEを要求する必要があります。

## 15.4 プロビジョニングの使用

### 15.4.1 プロビジョニングの要求

ジョブの実行に必要なリソースおよびAOEを予約するには、AOEを指定して予約を要求します。ジョブにAOEが必要な場合は、そのジョブに必要なAOEを要求します。ジョブまたは予約のプロビジョニングを要求するには、同じ構文を使用します。

ジョブ全体または予約全体に対してAOEを要求できます。

*-l aoe = <AOE>*

例：

```
-l aoe = suse
```

*-l <AOE>*形式は*-l select*と組み合わせて使用できません。

単一チャンクのジョブまたは予約に対してAOEを要求できます。

*-l select=<チャンク要求>:aoe=<AOE>*

例：

```
-ls select=1:ncpus=2:aoe=rhel
```

ジョブまたは予約の各チャンクに対して同じAOEを要求できます。

*-l select=<チャンク要求>:aoe=<AOE> + <チャンク要求>:aoe=<AOE>*

例：

```
-l select=1:ncpus=1:aoe=suse + 2:ncpus=2:aoe=suse
```

ジョブまたは予約の一部のチャンク（すべてのチャンクではなく）に対してAOEを要求できます。

*-l select=<チャンク要求>:aoe=<AOE> + <チャンク要求>*

例：

```
-l select=1:ncpus=1:aoe=suse + 2:ncpus=2
```

### 15.4.2 コマンドとプロビジョニング

サブステータスが *provisioning* のジョブで PBS コマンドの使用を試みると、コマンドの動作が他の場合と異なります。vnode のプロビジョニングはコマンドによる影響を受けません。プロビジョニングがすでに開始している場合は続行されます。以下の表に、影響を受けるコマンドを示します。

**表 15-2：サブステータスが Provisioning のジョブに対するコマンドの影響**

| コマンド | サブステータスが provisioning の場合の動作 |
|---|---|
| qdel | （force 指定なし）ジョブは削除されません |
| | （force 指定あり）ジョブは削除されます |
| qsig -s suspend | ジョブは中断されません |
| qhold | ジョブは保留されません |
| qrerun | ジョブは再キューイングされません |
| qmove | プロビジョニング中のジョブでは使用できません |
| qalter | プロビジョニング中のジョブでは使用できません |
| qrun | プロビジョニング中のジョブでは使用できません |

### 15.4.3 プロビジョニングがジョブに与える影響

AOE を要求したジョブは優先実行されません。したがって、AOE を要求したジョブを実行するために他のジョブが終了することはありません。

AOE を要求したジョブはバックフィルされません。

## 15.5 注意事項とエラー

### 15.5.1 要求されたジョブの AOE と予約の AOE は一致する必要がある

AOE を要求するジョブは、同じ AOE を要求していない予約に投入しないでください。予約済み vnode で対象の AOE を提供していない可能性があり、その場合はジョブが実行されません。

### 15.5.2 予約には十分な時間を確保する

ジョブの walltime に近い期間が設定された予約にジョブを投入すると、プロビジョニングが原因で、ジョブの実行が完了する前にジョブが終了するか、またはジョブが開始されない可能性があります。AOE を要求するジョブを受け入れるように設計された予約では、プロビジョニングに対応するため、予約に余分な時間を十分に確保します。

## 15.5.3 ジョブまたは予約に対して複数のAOEを要求する

１つのジョブまたは予約に対して複数のAOEを要求しないでください。これを行うと、ジョブが実行されないか、または予約が未確認のままになります。

## 15.5.4 ジョブの保留または再キューイング

以下の理由により、ジョブはシステムによって保留されます。

- 無効なプロビジョニング要求または内部システムエラーが原因でプロビジョニングが失敗する
- プロビジョニング後、vnodeによってレポートされたAOEが、ジョブの要求したAOEと一致しない

保留状態は、PBS管理者によって、問題の調査、修正の後に解除されます。

以下の理由により、ジョブは再キューイングされます。

- タイムアウトが原因で、プロビジョニングのhookに失敗する
- vnodeでバックアップがレポートされない

## 15.5.5 リソース要求の競合

プロビジョニングによって、リソースarchおよびvnodeの値が変更される可能性があります。同じジョブに対して、AOEと、archまたはvnodeのいずれかを要求しないでください。

## 15.5.6 ジョブの投入と変更では要件が同じである

`qsub`コマンドを使用してジョブを投入する場合も、`qalter`コマンドを使用してジョブを変更する場合も、ジョブは最終的には同じ要件を満たす必要があります。要件を満たしているジョブを投入した後で、要件を満たさなくなるような変更をジョブに加えることはできません。

# 16
# アカウンティングの使用

## 16.1 アカウンティングの使用

### 16.1.1 アカウンティング文字列の指定

Account_Name ジョブ属性の値を設定することにより、ジョブにアカウンティング文字列を関連付けることができます。この属性にはデフォルト値はありません。ユーザーはコマンドラインで、または PBS 指示文で、Account_Name の値を設定できます。

**qsub -A <アカウンティング文字列>**

#PBS Account_Name=<アカウンティング文字列>

<*アカウンティング文字列*>には、任意の文字列を指定できます。PBSはこの文字列を解釈しようとはしません。

qalterコマンドを使用すると、Account_Name ジョブ属性の値を、ジョブがキューで待機中に変更し、ジョブの実行中には変更しないようにすることができます。

### 16.1.2 Comprehensive System Accounting の使用

CLE 5.2 を実行している Cray システム上で CSA を使用できます。HPE システムの CSA に対する PBS のサポートは利用できません。HPE システムの CSA 機能は PBS から**削除**されています。

CSAは、ユーザージョブアカウンティングと呼ばれるユーザージョブに関するアカウンティング情報を提供します。

CSA は、PBS と共に、もしくは単独で稼働します。ユーザージョブアカウンティングを実行するには、生アカウンティング情報が書き出されるファイルを指定するか、もしくは環境変数を設定する必要があります。環境変数は、ACCT_TMPDIR です。これは、生アカウンティングデータが書き出されるテンポラリファイルのディレクトリです。

ユーザージョブアカウンティングを実行するには、CSA コマンド“ja <ファイル名>”を実行するか、または環境変数 ACCT_TMPDIR が設定される場合は“ja”を実行します。アカウンティングレポートを作成するには、コマンド“ja -<オプション>”を実行します。このオプションでは、レポートの作成とその種類を指定します。ユーザージョブアカウンティングを終了するには、コマンド“ja -t”を実行します。この-t オプションは、これ以前のオプションのセットに含めることが可能です （詳細については、マニュアルページ ja を参照）。

jaコマンドの開始と終了は、ユーザーがモニタリングを希望する他のコマンドの前と後ろに使用されなければなりません。コマンドラインとスクリプトの例を以下に示します。

コマンドラインで：

```
qsub -N myjobname -l ncpus=1
    ja myrawfile
    sleep 50
    ja -c > myreport
    ja -t myrawfile
ctrl-D
```

ジョブ（sleep 50）についてのアカウンティングデータは、myreportに書き出されます。

これらのコマンドでジョブスクリプトfooを作成する場合は、

```
#PBS -N myjobname
#PBS -l ncpus=1
ja myrawfile
sleep 50
ja -c > myreport
ja -t myrawfile
```

これで、qsubを使用してジョブスクリプトを実行し、前の例と同じ操作を実行できます。

```
qsub foo
```

## 16.1.3 アカウンティングでの依存関係の使用

終業時アカウンティングを実行する必要がある場合は、依存関係を使用できます。113ページの第6.2節「ジョブの依存関係の使用」をご参照ください。

## 16.1.4 アカウンティングを使用する場合のアドバイスと注意事項

### 16.1.4.1 統合されたMPIの使用

PBSに統合されたMPIは多数あります。PBSは、それらのほとんどを統合するツールを提供します。いくつかのMPIは統合が可能です。ジョブが統合されたMPIで実行されると、PBSはジョブのすべてのプロセスに対してリソース使用量を追跡し、ジョブプロセスにシグナルを送信し、アカウンティングを実行できます。

ジョブがPBSに統合されていないMPIで実行されると、PBSはプライマリvnode上のジョブの管理のみに制限されるため、リソース追跡、ジョブへのシグナルの送信、アカウンティングはプライマリvnode上のプロセスにのみ行われます。

Windowsでは、MPICHなど一部のMPIはPBSに統合されていません。

86ページの第5.2.1節「統合されたMPIの使用」をご参照ください。

# 索引