Rev.2 2013 年 1 月 16 日

Beowulf Cluster Reference System

Grid Scheduler

・簡易操作マニュアル

・管理マニュアル

NABE³ International

目次

Grid Scheduler 簡易操作マニュアル

Grid Scheduler 管理マニュアル

Grid Scheduler 簡易操作マニュアル

Grid Scheduler（以下 GS）は、Sun Microsystems 社が立ち上げたオープンソースコミュニティ“Grid Engine Project”により開発されたソフトウェア Grid Engine が元になっており、クラスタ内の計算リソースを管理します。このソフトウェアを使用することにより、ユーザはどのノードのリソースが空いているかを気にすることなくジョブを投入・実行することができます。リソースが空いていない場合には待機中となり、いずれかのノードのリソースが空くと自動的に実行開始されます。

このマニュアルではジョブの投入、削除など基本的な操作を扱います。その他の操作に関しては Grid Scheduler のユーザーズマニュアルをご参照ください。

出荷時の設定

出荷時の設定では、全ノードを含むひとつのクラスタキュー(all.q)が作成され、all.q には各ノードごとに 1 つのキューインスタンス(all.q@hostname)が設定されています。各キューインスタンスで同時に実行可能なジョブの数（スロット数）は CPU コア数と同等です。ジョブ投入時にノードを指定する必要はなく、GS がリソースの空いているキューインスタンスを探してディスパッチします。

また、一人のユーザが実行可能なジョブ数に制限が設けられており、一部のユーザが大部分のプロセッサを占有することがないように設定されています。

スロット数やユーザのジョブ数制限は変更可能です（後の管理マニュアル参照）。実際の使用状況に合わせて設定してください。

ユーザアカウント

ホストにログインしてジョブを投入します。ホストと各ノードに同じユーザアカウントを作成してください（Takeru オペレーションマニュアルのユーザ追加方法を参照）。

並列ジョブを実行する場合には、MPI が動作する条件が満たされている必要があります。（パスワード無しで ssh または rsh が実行できる必要があります。）

GS の起動と停止（root アカウントのみ）

GS の init スクリプトが/etc/init.d ディレクトリにあります。出荷時はマシン起動時にこのスクリプトが実行されるよう設定しています。

・起動
マスターデーモンを先に起動します。

```
[root@n000 root]# /etc/init.d/sgemaster.p6444  start

# ホストで sge_qmaster デーモンと sge_schedd デーモンを起動
```

[root@n000root]# /etc/init.d/sgeexecd.p6444 start;./allnode /etc/init.d/sgeexecd.p6444 start

ホストと全ノードで sge_execd デーモンを起動

・停止
実行デーモンを先に停止します。

[root@n000 root]# /etc/init.d/sgeexecd.p6444 stop;./allnode /etc/init.d/sgeexecd.p6444 stop

ホストと全ノードで sge_execd デーモンを停止

[root@n000 root]# /etc/init.d/sgemaster.p6444 stop

ホストで sge_qmaster デーモンと sge_schedd デーモンを停止

sge_qmaster デーモン停止の時点で実行中のジョブがあった場合、デーモン停止後もそのスクリプトは最後まで実行され続けますが、デーモン再起動後に再スケジューリング(一度待機中となり、その後再実行)されます。
待機中のジョブは次に sge_qmaster デーモンを起動した時に再び待機中となります。

ジョブを投入する

qsub [jobscript]

GS でジョブを投入するには、基本的にはシェルスクリプトにコマンドを記述する形になります。以下の例では、シェルスクリプトに投入コマンド qsub のオプションと実行したいプログラムを記述し、qsub コマンドの引数としてスクリプトファイル名を指定します。

実行例

```
[beowulf@n000 GS-demo]$    vi getest.sh  # スクリプトの作成
```

スクリプトの記述例

```
#!/bin/sh                                    ①

#$ -o out.txt                                ②

sleep 30                                     ③
hostname
date
```

```
[beowulf@n000 GS-demo]$ qsub getest.sh ④
        your job 142 ("getest.sh") has been submitted
```

①シェルを記述
②行頭に“#$”と記述し、qsub コマンドのオプションとその引 数を記述。この例では標準出力ファイルとして out.txt を作成。（デフォルトは“スクリプト名.o ジョブ ID”）
③実行したいプログラムを記述。この例では実行して 30 秒経過した後、ノードのホスト名と日付を出力。
④ qsub コマンドでジョブを投入、getest.sh というジョブ名が与えられ、ジョブ ID142 が割り振られる

```
[beowulf@n000 GS-demo]$ qstat                ⑤
```

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
142	0.55500	getest.sh	beowulf	r	07/29/2008 09:28:35	all.q@n000	1	

```
[beowulf@n000 GS-demo]$ ls                   ⑥
        getest.sh    getest.sh.e142 out.txt

[beowulf@n000 GS-demo]$ cat out.txt          ⑦
        n000
        Tue Jul 29 09:29:05 JST 2008
```

⑤投入されたジョブの状態を表示（all.q@n000 はジョブが所属するキューの名前）
⑥作成される標準出力ファイル・標準エラー出力ファイル
⑦保存された標準出力ファイルの表示

出荷時の設定では、1 つのキューインスタンスにおいて同時に実行可能なジョブの数（スロット数）が、対応するノードのプロセッサコア数と同じになっています。投入されたジョブの数が全キューインスタンスの合計スロット数に達すると、空きリソースなしとみなされジョブは待機中となります。例えばノード数 4、プロセッサコア数 32 の場合、qsub コマンドを 32 回以上連続して実行（すなわち、32 個以上のジョブを連続して投入）すると全てのプロセッサが使用されます。

ジョブ投入時、スクリプトに引数を渡すこともできます。例えばスクリプトを以下のようにします。実行結果は前述の例と同じです。

```
[beowulf@n000 GS-demo]$ vi getest2.sh
-----------------------------------------------------------------------------------------
スクリプトの記述例
#!/bin/sh

#$ -o out2.txt

sleep $1                                    ⑧
hostname
date
-----------------------------------------------------------------------------------------
[beowulf@n000 GS-demo]$ qsub getest2.sh 30   ⑨
        your job 143 ("getest2.sh") has been submitted
```

⑧引数として渡した秒数だけ sleep する。
⑨ジョブ投入時にスクリプトの引数を渡す

スクリプトを書かずに、コマンドを直接渡すことも可能です。オプション”-b y”を付けます。

```
[beowulf@n000 GS-demo]$ qsub -b y hostname  ⑩
        your job 144 ("hostname") has been submitted

 [beowulf@n000 GS-demo]$ cat hostname.o144 ⑪
        n000
```

⑩ジョブ投入時にコマンドを渡す。
⑪保存された標準出力ファイルの表示

主なオプション
　スクリプトに記述する他、コマンドラインからも指定可能です。

-a [date]	実行開始日時を指定（書式は MMDDhhmm）
-b [y|n]	バイナリモードの on/off（デフォルトは n）
-cwd	ジョブをカレントディレクトリで実行 （出荷時設定済み）
-e [path]	標準エラーを指定したファイルに保存（デフォルトは“スクリプト名. e ジョブ ID”）
-h	ホールド状態でジョブを投入（qrls コマンドを実行するまでジョブは実行されない。qrls コマンドについては後述）
-j [y|n]	標準出力と標準エラーを 1 つのファイルに保存
-m [mail option]	ジョブがある状態になるとメールで通知（デフォルトはメール通知なし） a　ジョブが削除された時、再スケジュールされた時にメールで通知 b　ジョブの実行が始まった時にメールで通知 e　ジョブの実行が終了した時にメールで通知 s　ジョブがサスペンド・再開された時にメールで通知 n　メール通知なし
-masterq [queue instance]	並列ジョブ実行時にマスタープロセスを実行するノードを指定する。 *などの正規表現を使用可能。　-masterq all.q@n000 のように指定する。
-N [job_name]	ジョブ名の指定　（デフォルトはスクリプトファイル名）
-now [y|n]	待機中リストに入れず、すぐに実行する。空リソースがない場合は実行されずに終了。
-o [path]	標準出力を指定したファイルに保存（デフォルトは“スクリプト名.o ジョブ ID”）
-p [priority]	ジョブのプライオリティを設定（デフォルトは 0、設定可能範囲は-1023～1024、一般ユーザは負の値のみ設定可能）
-pe [pe_name] [num_proc]	並列ジョブ実行時の並列環境と使用プロセス数（並列ジョブについては後述） *などの正規表現を使用可能。
-q [cluster queue|queue instance|hostgroup]	使用キューやノードの指定（デフォルトは空いているところへ自動的に投入） *などの正規表現を使用可能。 ・queue はクラスタキュー。書式は-q [cluster queue]。-q all.q のように指定する。 ・queue instance はクラスタキューに含まれるノード、hostgroup はホストグループ。 　書式は-q [cluster queue]@[hostname|hostgroup]。-q all.q@n000 のように指定する。ホストグループの場合は名前の最初に@が付くので、-q all.q@@XXX となる。

※　ジョブ名の冒頭の文字が数字の場合、実行が受け付けられません。必ずアルファベットにして下さい。-N オプションで指定しない場合、スクリプトファイルの名前がジョブ名となりますので、スクリプトファイル名 がアルファベットで開始される必要があります。

ジョブの状態を見る

qstat [option]

投入したジョブの状態を表示します。投入したジョブが実行中かどうか、どのノードで実行されているか、現在どのくらいの数のジョブが投入されているかを確認できます。

実行例
[beowulf@n000 GS-demo]$ qstat ①

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

[beowulf@n000 GS-demo]$ qstat -u “*” ②

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
456	0.55500	tkrtst.sh	takeru	r	07/29/2008 10:35:34	all.q@n001	1	
458	0.55500	tkrtst.sh	takeru	r	07/29/2008 10:35:35	all.q@n001	1	
460	0.55500	tkrtst.sh	takeru	qw	07/29/2008 10:35:38			
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:37:39			

①ジョブの状態を表示（自分のジョブのみ表示されます）
②ジョブの状態を表示（全てのユーザのジョブが表示されます）

各項目の説明

job ID	投入後のジョブに対して割り振られる番号
prior	ジョブのプライオリティ（プライオリティを設定しない場合はどのジョブも同じ数字となる）
name	投入時に与えられたジョブの名前（デフォルトはスクリプトファイル名）
user	ジョブ所有者
state	ジョブの状態 r 実行中 qw 待機中のジョブ（リソースが空けば実行可能な状態

	h　ホールド中（qrls コマンドを実行するまではジョブが実行されない状態） d　削除中 t　移動中（待機中から実行に移される間の状態） s　サスペンド（一時停止）状態 S　サスペンド状態（ジョブが所属するキューがサスペンドされている） T　サスペンド状態（suspend threshold を超えたためにサスペンドされている） Rq　再スケジューリングされ待機中のジョブ Rr　再スケジューリングされ実行中のジョブ E　エラー状態
submit/start at	投入/開始日時
queue	ジョブが所属しているキューの名前（デフォルトのキュー名は“all.q@hostname”）
slots	ジョブが使用している slots の数
ja-task-D	ジョブアレイタスク ID（アレイジョブでない場合は空欄）

主なオプション

-f	クラスタキューごとにジョブ情報を表示
-g t (g と t の間はスペース)	並列ジョブのスレーブプロセスまで表示（オプションなしだとマスタープロセスのみ表示）
-ne	現在ジョブのないキューを除外（-f オプションと組み合わせて使用）
-q [cluster queue|queue instance]	指定したキューについて表示（指定キュー内のジョブのほか、待機中のジョブも表示される）
-s [p | r | h]	指定したステイタスのジョブのみ表示 [pending （待機中）| running | hold]
-u [username]	指定したユーザのジョブを表示

ジョブを削除する

qdel [option] [jobID]

実行中・待機中などジョブの状態に関わらず、ジョブを削除します。
ジョブを削除する権限を持つのは、そのジョブを投入したユーザアカウントか root アカウントのみです。

実行例
[beowulf@n000 GS-demo]$ qstat

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

[beowulf@n000 GS-demo]$ qdel 459 461 ①
beowulf has registered the job 459 for deletion
beowulf has deleted job 461

[beowulf@n000 GS-demo]$ qstat ②

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	

①ジョブ ID459、461 のジョブを削除（実行中ジョブと待機中ジョブでは異なるメッセージが出現）
② qstat で見てみると、ジョブ ID459 と 461 がない。

ジョブ名を指定することもできます。同じ名前のジョブは全て削除されます。*など正規表現を使うこともできます。

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

[beowulf@n000 GS-demo]$ qdel getest.sh ③
beowulf has registered the job 457 for deletion
beowulf has registered the job 459 for deletion
beowulf has deleted job 461

③ getest.sh という名前のジョブを全て削除

主なオプション

-u　[username]	指定したユーザのジョブを全て削除(一般ユーザは自分のみ指定可能)

待機中のジョブをホールドする

qhold [jobID]

　待機中のジョブをホールドします。ホールド中のジョブは、実行可能な状態にするコマンド（qrls）を実行するまで待機中の状態が維持されます。リソースが空いていてもホールド中のジョブがディスパッチされることはなく、その次に待機中のジョブが先にディスパッチされます。
　ジョブをホールドする権限を持つのは、そのジョブを投入したユーザアカウントか root アカウントのみです。また、ホールドの対象となるのは待機中のジョブのみです。

実行例
[beowulf@n000 GS-demo]$ qstat

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

[beowulf@n000 GS-demo]$ qhold 461　　①

[beowulf@n000 GS-demo]$ qstat　　②

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	hqw	07/29/2008 10:28:39			

①ジョブ ID461 をホールドする。
②ジョブ ID461 がホールド状態になっている。

ホールド状態のジョブをリリースする

qrls [jobID]

qhold コマンドなどでホールドしたジョブを再び実行可能な状態にします。
ジョブをリリースする権限を持つのは、そのジョブを投入したユーザアカウントか root アカウントのみです。

実行例
[beowulf@n000 GS-demo]$ qstat

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	hqw	07/29/2008 10:28:39			

[beowulf@n000 GS-demo]$ qrls 461 ①

[beowulf@n000 GS-demo]$ qstat

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

①ジョブ ID461 をリリースする。
②ジョブ ID461 がホールド状態から通常の待機状態に戻っている。

※ root アカウントでホールドしたジョブは、ユーザアカウントでもリリース可能です。

実行中のジョブをサスペンドする

qmod –s [jobID]

　実行中のジョブを一時的に停止させます。サスペンドしたジョブはリソースを占有した状態となり、そのリソースに新たなジョブがディスパッチされることはありません。
　ジョブをサスペンドする権限を持つのは、そのジョブを投入したユーザアカウントか root アカウントのみです。また、サスペンド可能なのは実行中のジョブのみです。

実行例
[beowulf@n000 GS-demo]$ qstat

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

[beowulf@n000 GS-demo]$ qmod -s 459　①

[beowulf@n000 GS-demo]$ qstat　②

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	s	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

①ジョブ ID459 をサスペンドする。
②ジョブ ID459 がサスペンド状態になっている。

※ サスペンド中のジョブは、1 ユーザが同時に実行可能なジョブ数にカウントされません。

※ 並列ジョブに関しては、ジョブを停止してもスレーブプロセスが動作し続けます。

サスペンドしたジョブを再開する

qmod –us [jobID]

サスペンド中のジョブを再開します。プログラムは、サスペンドされた時点で停止した部分から再開されます。
ジョブを再開する権限を持つのは、そのジョブを投入したユーザアカウントか root アカウントのみです。

実行例
[beowulf@n000 GS-demo]$ qstat

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	s	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

[beowulf@n000 GS-demo]$ qmod -us 459 ①

[beowulf@n000 GS-demo]$ qstat ②

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

①ジョブ ID459 を再開する。
②ジョブ ID459 がサスペンド状態から実行状態になっている。

※ root アカウントでサスペンドしたジョブはユーザアカウントでも再開可能です。

ジョブの属性を変更する

qalter [option] [jobID]

qsub コマンド のオプションとして指定可能な属性の設定値を、ジョブ投入後に変更します。たとえばこのコマンドを用いてジョブのプライオリティを変更することで、待機中のジョブがディスパッチされる順序を変更できます。下記の例でその方法を示します。

ジョブの属性を変更する権限を持つのは、そのジョブを投入したユーザアカウントか root アカウントのみです。

実行例

[beowulf@n000 GS-demo]$ qstat

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
460	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:38			
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			

[beowulf@n000 GS-demo]$ qalter –p -10 460 ①

[beowulf@n000 GS-demo]$ qstat ②

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
459	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:35	all.q@n000	1	
457	0.55500	getest.sh	beowulf	r	07/29/2008 10:28:34	all.q@n000	1	
461	0.55500	getest.sh	beowulf	qw	07/29/2008 10:28:39			
460	0.55012	getest.sh	beowulf	qw	07/29/2008 10:28:38			

①ジョブ ID 460 の プライオリティを-10 下げる。
② prior の値が再計算され、ジョブ ID460 の優先順位が他のジョブより下がっている（prior の数値はコマンドラインで指定したものとは異なる）。

※ プライオリティの設定範囲は-1023～1024 で、デフォルトの値は 0 です。一般ユーザはプライオリティを下げることしかできません。root アカウントのみ、プライオリティを上げることができます。

主なオプション

-a [date]	実行開始日時を変更（書式は MMDDhhmm）
-cwd	ジョブをカレントディレクトリで実行するよう変更
-e [path]	標準エラーを指定したファイルに保存（デフォルトは“スクリプト名.e ジョブ ID”）
-j [y|n]	標準出力と標準エラーを 1 つのファイルに保存
-m [mail option]	ジョブがある状態になると実行ノードがメールで通知 （デフォルトはメール通知なし） a　ジョブが中止になったとき、再スケジュールされたときにメールで通知 b　ジョブの実行が始まったときにメールで通知 e　ジョブの実行が終了したときにメールで通知 s　ジョブがサスペンド・再開されたときにメールで通知 n　メール通知なし
-N [job_name]	ジョブ名の指定　（デフォルトはスクリプトファイル名）
-o [path]	標準出力を指定したファイルに保存（デフォルトは“スクリプト名.o ジョブ ID”）
-p [priority]	ジョブのプライオリティを設定（デフォルトは 0）一般ユーザは下げることしかできない
-pe [pe_name] [num_proc]	並列ジョブ実行時の並列環境と使用プロセス数（並列ジョブについては後述） *などの正規表現を使用可能。
-q [cluster queue|queue instance|hostgroup]	使用キューやノードの指定（デフォルトは空いているところへ自動的に投入） *などの正規表現を使用可能。 ・queue はクラスタキュー。書式は-q [cluster queue]。-q all.q のように指定する。 ・queue instance はクラスタキューに含まれるノード、hostgroup はホストグループ。 書式は-q [cluster queue]@[hostname|hostgroup]。-q all.q@n000 のように指定する。ホストグループの場合は名前の最初に@が付くので、-q all.q@@XXX となる。

ノードの状態を見る

qhost [option]

実行ノードが利用可能な状態かどうか、及び各実行ノードの属性（利用可能なハードウェアリソースや使用中のリソース等）を表示します。また、各実行ノード上で設定されているキューの状態を併せて表示できます。

実行例
[beowulf@n000 GS-demo]$ qhost ①

HOSTNAME	ARCH	NCPU	LOAD	MEMTOT	MEMUSE	SWAPTO	SWAPS
global	-	-	-	-	-		
n000	linux-x64	8	0.71	8.0G	540.3M	517.7M	0.0
n001	linux-x64	8	-	8.0G	-	517.7M	-

[beowulf@n000 GS-demo]$ qhost -q ②

HOSTNAME	ARCH	NCPU	LOAD	MEMTOT	MEMUSE	SWAPTO	SWAPS
global	-	-	-	-	-		
n000	linux-x64	8	0.71	8.0G	540.3M	517.7M	0.0
all.q	BIP 4/8						
n001	linux-x64	8	0.00	8.0G	205.4M	517.7M	0.0
all.q	BIP 0/8						
n002	linux-x64	8	0.00	8.0G	205.3M	517.7M	0.0
all.q	BIP 0/8	d					
n003	linux-x64	8	-	8.0G	-	517.7M	-
all.q	BIP 0/8	au					

①全実行ノードの状態を表示
②ノードの状態と共にキューの状態を表示

n000 は 4 つのジョブが投入されている状態です。n000 のキュー all.q の列の 4/8 という数値は 8 つのスロット数のうち 4 つが使用されている状態を示しています。

n001 は現在実行されているジョブがない状態です。n001 のキュー all.q の列の 0/8 という数値は 8 つのスロット数のうち使用されている数が 0 である状態を示しています。

n002 は使用不可の状態です（d = disable）。実行ノード n002 自体は利用可能ですが、Grid Scheduler 上では all.q のキューインスタンス n002 が停止状態であるため、Grid Scheduler 経由で投入したジョブが n002 実行されることはありません。

n003 は Grid Scheduler のマスターデーモンと通信できない状態です（a = alarm u = unknown）。該当ノードがダウンしていないかどうか、ネットワークに接続されているかどうか、GS のデーモン（sge_execd が起動しているかどうかを確認してください。alarm 状態は実行ノードが過負荷状態にある場合にも表示されます。

並列ジョブを投入する

Grid Scheduler では並列プログラムを実行する為の並列環境が定義可能であり、これを用いることで並列プログラムを実行できます。Takeru に構築済みの並列環境は以下の通りです。
MPICH
MPICH2
OpenMPI(Gigabit Ethernet/Infiniband 使用)
MVAPICH2(Infiniband 使用)

以下、clustalw-mpi というプログラムを各 MPI ライブラリを使って 12CPU で実行する場合において、ジョブスクリプトの記述の仕方について説明しています。

・MPICH

実行例

```
[beowulf@n000 GS-demo]$    vi mpiscript.sh
 ---------------------------------------------------------------------------------------------
 #!/bin/sh

#$ -pe mpich1 12                          ①
#$ -N MPIJOB

 /usr/local/mpich/bin/mpirun –np $NSLOTS –machinefile $TMPDIR/machines  $PWD/clustalw-mpi $PWD/db10.input
                                  ②
 ---------------------------------------------------------------------------------------------
```

①-pe オプションは GS に定義してある並列環境と、並列プロセス数を指定する。-pe 直後にある mpich1 は出荷時に設定済みの並列環境名であり、並列ライブラリ MPICH を用いたジョブを投入する場合に指定しなければならない。この例では 12CPU コアを要求するために、並列プロセス数に 12 を指定している。

② MPICH の実行コマンド mpirun を記述。斜体部分はサンプルプログラムと引数。-np は mpirun で実行されるプロセスの数を指定し、変数$NSLOTS には qsub コマンドの-pe オプションで指定された並列プロセス数が入る。-machinefile は mpirun 実行時に使用されるノードのリストを指定する。GS は要求された CPU 数に応じて利用可能なノードのリストを自動的に作成、提供するので、ここではそのファイル（$TMPDIR/machines）を指定している。

```
[beowulf@n000 GS-demo]$ qsub mpiscript.sh       ③
   your job 142 ("mpiscript.sh") has been submitted

[beowulf@n000 demos]$ qstat                ④
```

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
142	0.55500	mpiscript.sh	beowulf	r	07/29/2008 09:28:35	all.q@n000	12	

```
[beowulf@n000 demos]$ qstat -g t           ⑤
```

job-Id	prior	name	user	state	submit/start at	queue	master	ja-task-ID
142	0.55500	mpiscript.sh	beowulf	r	07/29/2008 09:28:35	all.q@n000	MASTER	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
142	0.55500	mpiscript.sh	beowulf	r	07/29/2008 09:28:35	all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	

[beowulf@n000 demos]$ ls ⑥
mpiscript.sh mpiscript.sh.o142 mpiscript.sh.e142 mpiscript.sh.pe142 mpiscript.sh.po142 PI1234

[beowulf@n000 demos]$ cat mpiscript.sh.po142 ⑦
n000
n000
n000
n000
n000
n000
n001
n001
n001
n001
n001
n001

③ジョブの投入（ジョブ ID142 が割り振られる）
④マスタージョブだけ表示される。slots の欄が 12 になっている。
⑤ スレーブジョブまで表示。
⑥標準出力ファイル・標準エラーファイルのほか、GS が並列環境を使用開始した時の標準出力ファイル（*.po*）と標準エラーファイル（*.pe*）が作成される。PI1234 は MPICH の出力ファイル。
⑦ Grid Scheduler が並列環境（この実行例の場合、-pe オプションで指定した mpich）を使用した際の標準出力。使用ノードのリスト。

-pe オプションと共に-masterq オプションを使用してキューインスタンスを指定すると、マスタージョブの実行場所を指定することができます（デフォルトは指定なし）。

[beowulf@n000 GS-demo]$ qsub –masterq all.q@n001 mpiscript.sh
your job 143 ("mpiscript.sh") has been submitted

[beowulf@n000 GS-demo]$ qstat -g t

job-Id	prior	name	user	state	submit/start at	queue	master	ja-task-ID
143	0.55500	mpiscript.sh	beowulf	r	07/29/2008 09:30:11	all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
143	0.55500	mpiscript.sh	beowulf	r	07/29/2008 09:30:11	all.q@n001	MASTER	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	

※ 並列ジョブは、マスタージョブとスレーブジョブを合わせて 1 つのジョブとして数えられます。

※ qdel コマンドで並列ジョブを削除すると、mpirun は終了しますが各ノードの実行プログラム(スレーブプロセス)は終了しません（qstat コマンドで確認すると、Grid Scheduler 上ではスレーブジョブも含め全て削除されたように見えます）。スレーブプロセスを手動で kill してください。

・MPICH2

MPICH2 を使用する場合は指定する並列環境や実行コマンドを以下のようにしてください。

実行例

```
[beowulf@n000 GS-demo]$    vi mpich2script.sh
 ---------------------------------------------------------------------------------------
 #!/bin/sh

#$ -pe mpich2 12                    ①
#$ -N MPICH2JOB

 /usr/local/mpich2/bin/mpirun –np $NSLOTS –f $TMPDIR/machines  $PWD/clustalw-mpi
$PWD/db10.input
                                ②
 ---------------------------------------------------------------------------------------
```

①-pe オプションの指定の仕方は同じ。mpich2 という名の並列環境を選ぶ。

② MPICH2 の実行コマンド mpirun を記述。斜体部分はサンプルプログラムと引数。-np の指定は mpich の場合と同様である。マシンファイルの指定は-f を使う。

※ MPICH2 のジョブは、qdel コマンドでジョブを削除すると各ノードのスレーブプロセスまで kill することができます。

・OpenMPI

OpenMPI を使用する場合は指定する並列環境や実行コマンドを以下のようにしてください。

実行例

```
[beowulf@n000 demos]$    vi ompiscript.sh
 ----------------------------------------------------------------------------------------------
 #!/bin/sh

#$ -pe openmpi 12                    ①
#$ -N OMPIJOB

 /usr/local/openmpi/bin/mpirun –np $NSLOTS –hostfile $TMPDIR/machines $PWD/clustalw-mpi $PWD/db10.input

 ----------------------------------------------------------------------------------------------
```

①-pe オプションの指定の仕方は同じ。openmpi という名の並列環境を選ぶ。

② OpenMPI の実行コマンド mpirun を記述。斜体部分はサンプルプログラムと引数。-np と指定は mpich の場合と同様である。マシンファイルの指定は-hostfile を使う。

※ Infiniband クラスタの場合、OpenMPI は Infiniband 版と GIgabit Ethernet 版の 2 種類ありますこれらは、mpirun コマンドのインストール場所が違います。構成によっては、並列環境名も異なる場合があります。

※ OpenMPI のジョブは、qdel コマンドでジョブを削除すると各ノードのスレーブプロセスまで kill することができます。

・MVAPICH2

MVAPICH2 は Infiniband で使用する並列ライブラリです。使用する場合は指定する並列環境や実行コマンドを以下のようにしてください。

実行例

```
[beowulf@n000 demos]$    vi mv2script.sh
 ----------------------------------------------------------------------------------------------
 #!/bin/sh

 #$ -pe mvapich2 12                  ①
 #$ -N MV2JPB

 /usr/local/mvapich2/bin/mpirun –np $NSLOTS –f $TMPDIR/machines  $PWD/clustalw-mpi
```

$PWD/db10.input

②

---

①-pe オプションの指定の仕方は同じ。mvapich2 という名の並列環境を選ぶ。

② MVAPICH2 の実行コマンド mpirun_rsh を記述。斜体部分はサンプルプログラムと引数。-np の指定は mpich の場合と同様である。マシンファイルの指定は-f を使う。

※ MVAPICH2 のジョブは、qdel コマンドでジョブを削除すると各ノードのスレーブプロセスまで kill することができます。

並列ジョブにおける 1 台あたりの CPU 数指定について

前項の並列ジョブ実行では、1 台あたりに確保する CPU 数は決まっていません。ある時点での他のジョブの実行状況に応じて変動します。たとえば、並列環境 mpich1 の場合、Grid Scheduler が作成する machines ファイルは以下のようになります。

n000
n000
n000
n001
n001
n002

といったようにホスト名が複数記述されることで使用する CPU の数が表現されます。

これに対し、1 台あたりで確保する CPU コア数をあらかじめ指定した並列環境（mpich1.X, mpich.X, openmpi.X, mvapich2.X）が用意してあります。並列環境 mpich1 と mpich1.X はいずれも MPICH を使用しますが、Grid Scheduler が使用する CPU コアの取得方法と、Grid Scheduler により提供される machines ファイルに違いがあります。

並列環境 mpich1.X の場合、各ホストから必ず X 個の CPU コアを取得するように制限がかけられており、提供される machines ファイルも、[ホスト名]:[cpu 数]といった書式になります。例えば mpich1.8 という名の並列環境の場合、

n000:8
n001:8
n002:8

のように記述されます。

これにより。並列プログラムによっては、こちらの方が性能がよくなる場合があります。計算ジョブの種類や量、また、Takeru のCPU コア数によって最適な並列環境を選択してください。

実行例

ここでは、並列環境 mpich1.4 を使用して並列ジョブを実行する方法について扱います。以下の例では clustalw-mpi という並列プログラムを 8CPU で実行する場合において、ジョブスクリプトとその実行状況や出力について記述しています。

[beowulf@n000 GS-demo]$　vi　mpi4script.sh

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

#$ -pe mpich1.4 8                    ①

/usr/local/mpich/bin/mpirun –np $NSLOTS –machinefile $TMPDIR/machines  $PWD/clustalw-mpi $PWD/db10.input
---------------------------------------------------------------------------------------------
```

①-pe オプションは Grid Scheduler に定義してある並列環境と、並列プロセス数を指定する。-pe 直後にある mpich1.4 は、1 台あたり 4 コアを確保する。この例では 12CPU を要求するために、並列プロセス数に 12 を指定している。4CPU 以上空いているマシンが 2 台確保できたら実行される。並列環境 mpich1 と違い、2CPU 空いているマシンが 4 台あったとしてもこのスクリプトは実行されない。

[beowulf@n000 GS-demo]$ qsub mpi4script.sh

your job 142 ("mpi4script.sh") has been submitted

[beowulf@n000 GS-demo]$ qstat -g t

job-Id	prior	name	user	state	submit/start at	queue	master	ja-task-ID
142	0.55500	mpi4script.sh	beowulf	r	07/29/2008 09:28:35	all.q@n000	MASTER	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
						all.q@n000	SLAVE	
142	0.55500	mpi8script.sh	beowulf	r	07/29/2008 09:28:35	all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	
						all.q@n001	SLAVE	

この例では、
n000:4
n001:4
という machines ファイルが Grid Scheduler より提供され、n000 が master となって並列プログラムが実行されています。

マルチスレッド並列ジョブを投入する

GS では OpenMP や pthread を用いたマルチスレッド並列プログラムを実行する為の並列環境が定義可能であり、これを用いることで並列プログラムを実行できます。ここではこのマルチスレッド並列ジョブを実行する方法について扱います。以下の例では blastall というマルチスレッド並列対応のプログラムを 6CPU コアで実行する場合において、ジョブスクリプトとその実行状況や出力について記述しています。

実行例

```
[beowulf@n000 GS-demo]$    vi ge_blastall-a6.sh   #スクリプトの作成
 ---------------------------------------------------------------------------------------------
 #!/bin/sh

#$ -l nc=6                                    ①
 /usr/local/ncbi/bin/blastall -p blastx -a6 -d $PWD/blastdb/nr -i $PWD/test.seq –o $PWD/nr-
test.out
                                    ②
 ---------------------------------------------------------------------------------------------
```

① -l オプションはジョブが使用するリソースを定義するために使います。この場合、nc というリソースを 6 つ使用することを宣言しています。nc は 1 台あたりそのホストの CPU コア数分まで使用可能で、特に指定しないと1ジョブにつき 1 つ消費されます。このジョブは nc が 6 つ確保できない限り実行されません。また、このジョブが実行されているホストでは(CPU コア数ー 6)コアのみ使用可能です。
② blastall の-a6 オプションで 6 つの CPU コアを 使用するように指定しています。

```
[beowulf@n000 GS-demo]$ qsub ge_blastall-a6.sh
       your job 143 ("ge_blastall-a6.sh") has been submitted

[beowulf@n000 GS-demo]$ qstat
```

job-Id	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
143	0.55500	ge_blastall-a8.sh	beowulf	r	07/29/2003 10:28:35	all.q@n000	1	

qstat で見ると slos は 1 ですが、このジョブは CPU コアを 6 個消費します。

CPU コアを固定してジョブを実行する

通常、ジョブ実行時に使用する CPU コアは OS が判断するため、プロセス実行状況によって使用コアが移動します。qsub オプションで、使用する CPU コアを固定することができます。

特に、マルチスレッド並列ジョブや MPI 並列ジョブをマルチソケット・マルチコアのハードウェアで実行する場合に 同じ CPU ソケット内のコアを使用するようにするとパフォーマンスがよい場合があります。

・通常のジョブの場合

```
[beowulf@n000 GS-demo]$ vi getest-cb.sh
---------------------------------------------------------------------------
#!/bin/sh

#$ -binding linear:1                          ①
#$ -o out.txt

/usr/local/bowtie/bowtie -v 2 -t $PWD/s_cerevisiae/s_cerevisiae  -f $PWD/s_4_sorted.fasta
---------------------------------------------------------------------------
```

①-binding オプションは使用コアを固定する。上記は 1 コアで固定する場合。

・マルチスレッド並列ジョブの場合

以下、bowtie というプログラムを 4 コアでマルチスレッド実行する場合において、ジョブスクリプトの記述の仕方について説明しています。4 コアの CPU を搭載したシステムにおいて、同じ CPU のコアのみで実行したいときは以下のように記述します。

```
[beowulf@n000 GS-demo]$ vi getest-cb4.sh
---------------------------------------------------------------------------
#!/bin/sh

#$ -l nc=4                                    ②
#$ -binding linear:4                          ③

/usr/local/bowtie/bowtie -p 4 -v 2 -t  $PWD/s_cerevisiae/s_cerevisiae -f  $PWD/s_4_sorted.fasta
---------------------------------------------------------------------------
```

②マルチスレッド並列ジョブの実行に必要なリソース定義のオプション。
③-binding オプションは使用コアを固定する。linear:4 とした場合、連続した 4 コアの使用を要求する。

・MPI 並列ジョブの場合

以下、clustalw-mpi というプログラムを MPI ライブラリを使って 4 コアで実行する場合において、ジョブスクリプトの記述の仕方について説明しています。4 コアの CPU を搭載したシステムにおいて、同じ CPU のコアのみで実行したいときは以下のように記述します。

```
[beowulf@n000 GS-demo]$ vi mpi4-cb.sh
---------------------------------------------------------------------------
```

```
#!/bin/sh

#$ -pe mpich2.4 4                    ④
#$ -binding linear:4                 ⑤

/usr/local/mpich2/bin/mpirun –np $NSLOTS –f $TMPDIR/machines $PWD/clustalw-mpi
$PWD/adb10.input
---------------------------------------------------------------------------
```

④並列ジョブの実行に必要な並列環境指定のオプション。
⑤-binding オプションは使用コアを固定する。linear:4 とした場合、連続した 4 コアの使用を要求する。

・使用中コアの確認方法

qhost コマンドのオプションで指定したホストの使用中コアを確認できます。以下は 4 コアの CPU が 2 ソケットある場合です。

実行例

```
[beowulf@n000 GS-demo]$ qhost -F -h [hostname] |grep m_topology_inuse
 hl:m_topology_inuse=SCCCCSCCCC                                    ⑥

[beowulf@n000 GS-demo]$ qsub -binding linear:4 -l nc=4 [script]
  Your job 4756 ("[script]") has been submitted

[beowulf@n000 GS-demo]$ qhost -F -h [hostname] |grep m_topology_inuse
  hl:m_topology_inuse=SccccSCCCC                                   ⑦
```

⑥ジョブが実行されていないホストの状態です。大文字の SCCCC で 1 ソケットの CPU のトポロジーを示しています。
⑦ -binding linear:4 オプションを付けてジョブを投入した後、トポロジーを見てみると、使用コアが小文字の c となっています。

※　確認できるのは-binding オプションで指定されたジョブが使用する CPU コアのみです。

・1CPU のコア数より少ないコア数で実行するとき

-binding linear:x オプションが指定されると、可能ならば連続したコアを使用しようとしますが、該当する CPU がない場合は連続しないコアを使用してジョブを実行します。例えば、4 コアの CPU が 2 ソケット搭載されているホストにおいて、他のジョブが 3 コアずつ使用している場合、-binding linear:2 を指定されたジョブは 2 つの CPU の残り 1 個ずつを使用します。

この状態を防ぐには、qhost -F で表示される m_topology_inuse を要求リソースとしてジョブ投入時に指定します。

```
[beowulf@n000 GS-demo]$ vi bowtie-cb2.sh
---------------------------------------------------------------------------
#!/bin/sh

#$ -l nc=2
```

```
#$ -binding linear:2
#$ -l utopo=*CC*                                    ⑧

/usr/local/bowtie-0.12.7/bowtie -v 2 -t -p 2 s_cerevisiae/s_cerevisiae  -f s_4_sorted.fasta
 ------------------------------------------------------------------------------------
```

⑧２コア以上連続であいている CPU を使用するためのオプション。連続した２コアが確保されるまでジョブが待機状態となる。

※　スケジューリングのタイミングによっては、トポロジー変更が反映されるまでのタイムラグにより要求した通りではなく連続しないコアが使用されることがあります。

make を Grid Scheduler 経由で行う

GS には、qmake というコマンドが用意されており、make を Grid Scheduler 経由でで行うことができます。qmake を実行するときは make という並列環境を使用します。プログラムのコンパイルだけでなく、make の機能を使ったプログラムも実行できます。使い方は make コマンドと同様で、Makefile のあるディレクトリで

```
[beowulf@n000 GS_make]$ qmake -pe make 16 --
```

と実行します。--は、Grid Scheduler のオプションと make のオプションおよびターゲットを区別するものです。例えば make parallel というコマンドを qmake で実行したい場合は

```
[beowulf@n000 GS_make]$ qmake -pe make 16 -- parallel
```

と実行します。

GUI でジョブを操作する

qmon

X 上で qmon コマンドを実行すると、ジョブの投入や削除などの操作をグラフィカルなユーザインタフェース上で行うことができます。

下記の画像は qmon のメインコントロールウィンドウです。各ボタンの上にマウスカーソルを置くとメニューの名前が表示されます。ここでは、ジョブの投入・削除などを行うメニュー（Job Control）と各ノードの状態をモニターするメニュー（Queue Control）について述べます。Exit ボタンをクリックすると qmon が終了します。

[beowulf@n000 beowulf]$ qmon

Job Control

JobContol ボタンをクリックすると、以下のウィンドウが出現します。

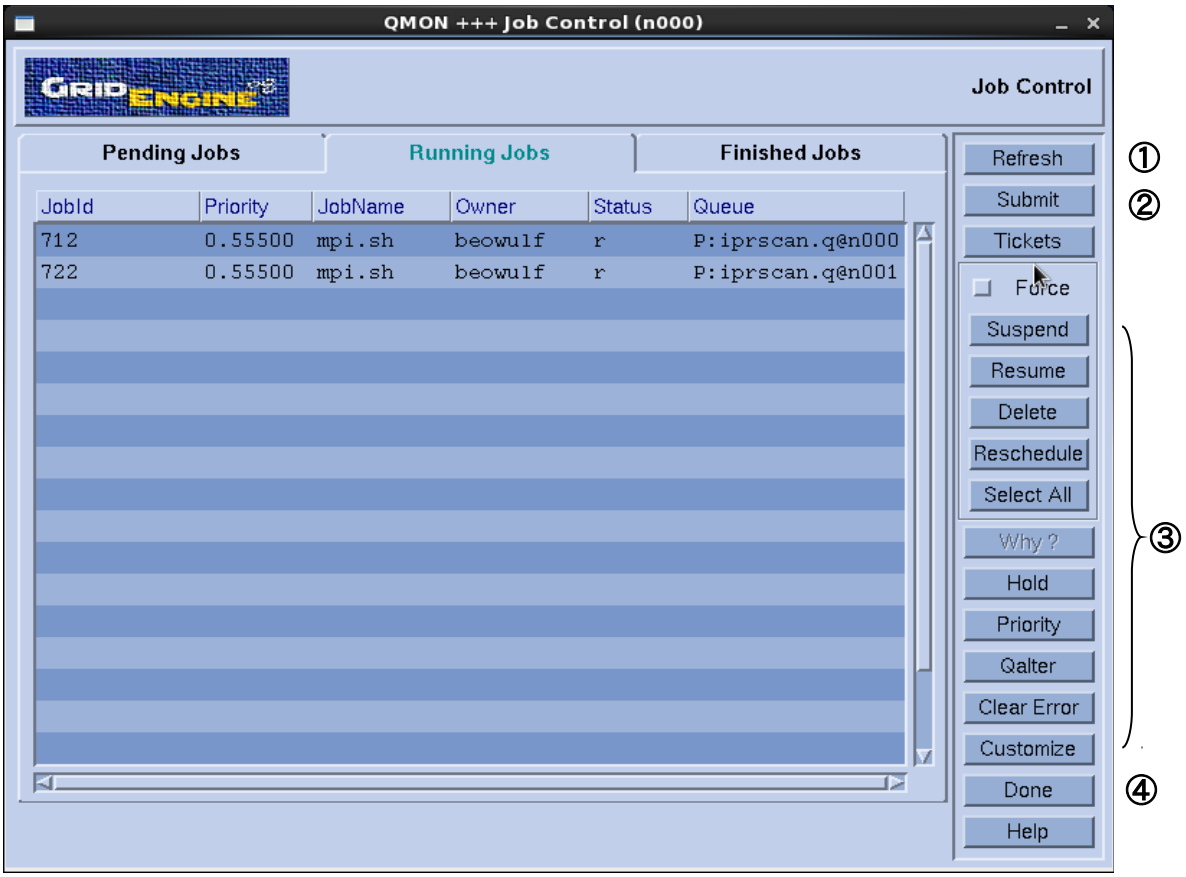
　下図は Running Jobs タブを選択した状態であり、ここでは現在実行中のジョブの一覧が示されています。左側の Pending Jobs タブを選択すると待機中のジョブの一覧が、Finished Jobs タブを選択すると終了したジョブの一覧(最近終了したものについて 100 個まで) が表示されます。

① Refresh ボタンをクリックすると、最新の状態に更新されます。

② Submit ボタンをクリックすると、ジョブ投入のためのウィンドウが出現します。GUI 上からジョブを投入する場合、標準の出力ファイルは qmon コマンドを実行したディレクトリに作成されます。

③ジョブ一覧からマウスでジョブを選択して右側の各種ボタンをクリックすると、ジョブ削除などの操作を行うことができます。(ジョブを選択する際に Shift キーや Ctrl キーを使うと、複数のジョブを選択することが可能です。)

④ Done ボタンをクリックすると、メニューが終了します。



Queue Control

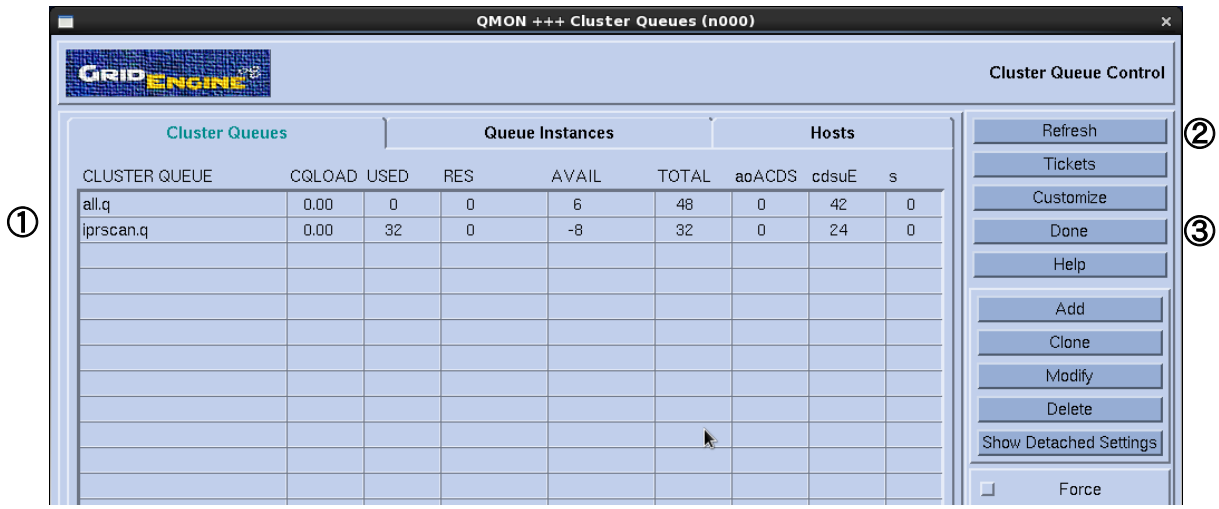
Queue Control ボタンをクリックすると、クラスタキュー、キューインスタンス、各ノードの状態を確認できます。この画面からキューの設定変更を行うことも可能ですが、操作できるのは root アカウントのみとなります。

Cluster Queues タブ

①クラスタキューについて名前、全体のスロット数、使用中のスロット数などが示されています。

② Refresh ボタンをクリックすると、最新の状態に更新されます。

③ Done ボタンをクリックすると、ウィンドウが終了します。

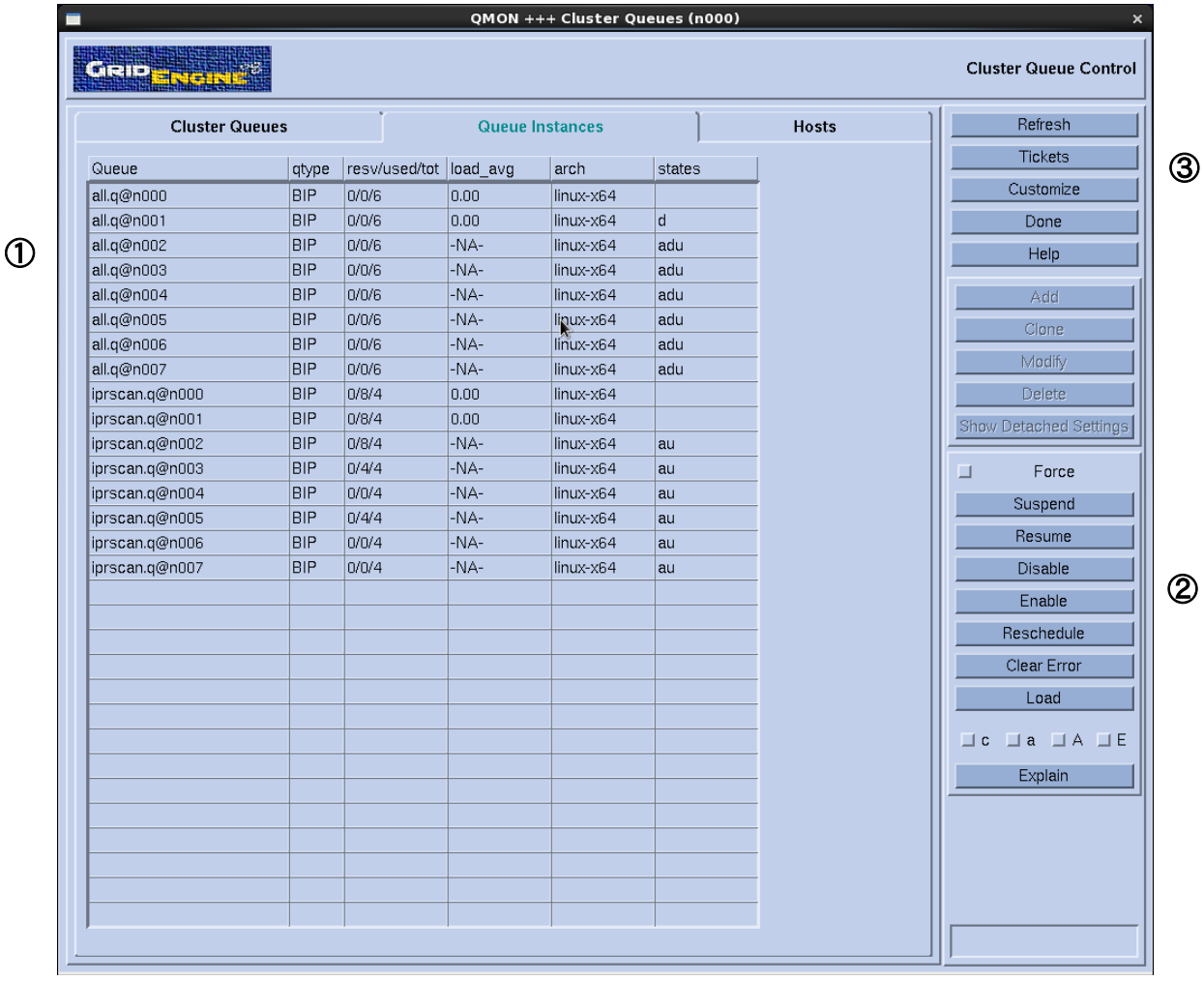


Queue Instances タブ

①クラスタキューに含まれるキューインスタンスの一覧が表示され、各キューインスタンスで利用可能なスロット数、と利用中のスロット数（used/total）、利用可能な状態かどうか(states)等の情報が表示されています。

②キューインスタンスごとに Disable/Enable できます。

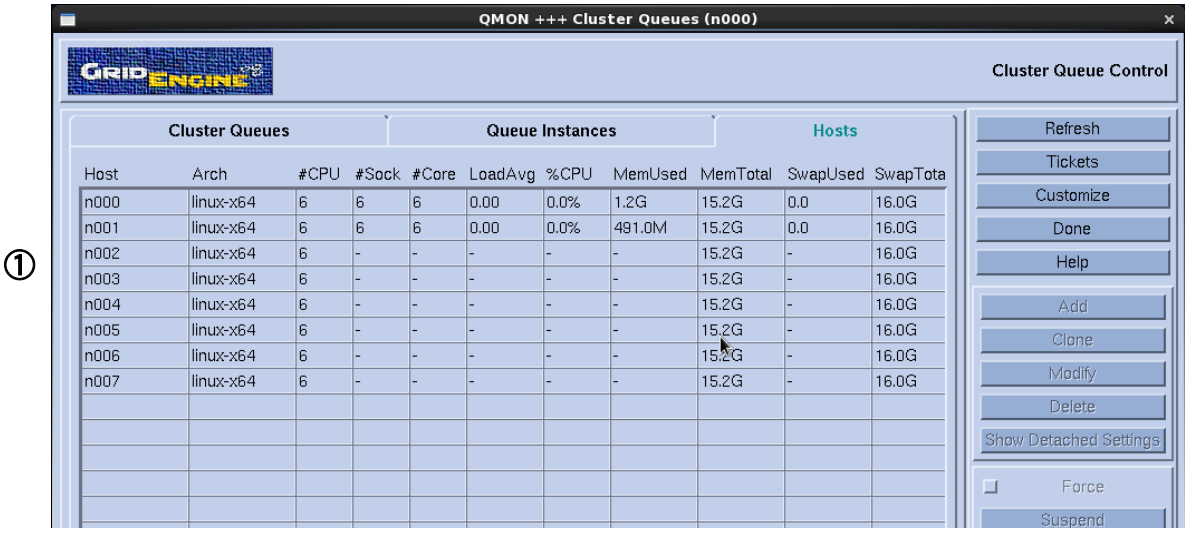
③ Cluster Queues タブと同様に Refresh ボタンと Done ボタンがあります。



Hosts タブ

①各ノードの一覧が表示されます。ノードごとの CPU 数、メモリ容量などが表示されます。

② Cluster Queues タブや Queue Instances タブと同様、Refresh ボタンと Done ボタンがあります。



Grid Scheduler 管理マニュアル

このマニュアルでは、以下の 3 点について出荷時の設定と変更方法を記述します。その他の詳細な設定に関しては、Grid Scheduler6.1 のアドミニストレーションマニュアルをご参照ください。

・デフォルトで使用可能なリソース
・実行ノードのグループ化
・障害時の対応
・1 ユーザあたりのジョブ数制限

Grid Scheduler はユーザが投入したジョブを受け取り、利用可能な計算リソースを割り当てて実行します。計算リソースの情報はキューと呼ばれるものがもっており、具体的な計算資源や、実行可能なジョブ数などが定義されています。

Grid Scheduler6.1 では、1 つのキューに複数のホストを対応させることができます。このようなキューはクラスタキューといいます。クラスタキューに含まれる各ホストはキューインスタンスと呼ばれます。クラスタキューで統一したリソースを定義することもできますし、キューインスタンス（各ホスト）ごとに個別のリソースを定義することもできます。

ホスト(ある 1 台のハードウェア)とキューインスタンスとの概念の違いは、1 つのホストに1つのキューインスタンスとは限らない、という点です。1つのホストは複数のクラスタキューに所属することができます。たとえば AAA.q と BBB.q いうリソース定義の異なるクラスタキューがあり、n000 というホストが両方のクラスタキューに所属している場合、Grid Scheduler 上ではキューインスタンス AAA.q@n000 と BBB.q@n000 は別のものと考えます。したがって、片方を停止して、片方だけ使用可能にするということが可能です。n000 自体にジョブを流さないようにするには、両方とも停止する必要があります。

Grid Scheduler の主な設定項目には以下のものがあります。目的によっては複数のファイルを修正する必要があります。それらのファイルは全て qconf コマンドで修正可能であり、指定するオプションによって操作の対象となる設定ファイルが変わります。各種設定の変更は root アカウントのみ可能です。

キュー設定　同時に実行可能なジョブ数など、計算資源をキューごとに設定
グローバル設定　　通信不能なノード上のキューを停止させるまでのインターバルなどを設定
スケジューラ設定　　ユーザーが同時に実行可能なジョブ数など、スケジューリング規則に関する設定

・キュー設定内容の表示
[root@n000 root]# qconf -sq [cluster queue|queue instance]

・ホストグループに含まれるホスト名の表示
[root@n000 root]# qconf -shgrp [hostgroup]

・グローバル設定内容の表示
[root@n000 root]# qconf –sconf

・スケジューラ設定内容の表示
[root@n000 root]# qconf –ssconf

本マニュアルでは主にコマンドラインで設定を変更する方法について説明しますが、後述のように GUI を用いて設定を変更することもできます。

デフォルトで利用可能なリソース

出荷時の設定では、全ホストを含むクラスタキュー all.q が設定され、その下に all.q@ホスト名というキューインスタンスが存在します(構成によっては、出荷時の段階で別のクラスタキューを作成してある場合もあります)。all.q および各キューインスタンスの設定が、各ホストで利用可能なリソースを決定しています。計算リソースに関してはさまざまな設定項目がありますが、本マニュアルでは 1 台のノードで同時に実行可能なジョブ数に関する項目のみ記述します。

出荷時のリソース設定では、プロセッサを効率的に使用するために、1 つのプロセッサコアに対して1つのジョブを割り当てるように設定しています。すなわち 1 台で同時に実行可能なジョブ数はプロセッサコア数と同数です。負荷の低いジョブを多数投入したい場合にはノードに割り当てるジョブ数を増やすことも可能ですが、通常は出荷時の状態で運用することをお勧めします。

主な設定項目と出荷時の設定(クアッドコアのプロセッサが 2 つ搭載されたノードの場合)

・キュー設定　**slots**
キュー内で同時に実行可能なジョブの数 (デフォルトは 8)
・キュー設定　**load_thresholds　np_load_avg**
負荷しきい値(この値を超えると、そのキューインスタンスにはそれ以上ジョブが投入されない。デフォルトは 8.75)

全てのキューインスタンスについて一度に変更する場合

```
[root@n000 root]# qconf -mq all.q
```

vi が呼び出されます。変更したい個所を書き換え、通常の vi と同様に保存して終了してください。

キューインスタンスごとに設定を変更する場合

```
[root@n000 root]# qconf -mq all.q
```

vi が呼び出されます。変更したい個所を書き換えるとき、以下の例のように記述してください。
(load_thresholds を変更する場合)

```
load_thresholds    np_load_avg=8.75,[n000=np_load_avg=4.0]
```

これは、all.q の設定は負荷しきい値が 8.75 であるが、n000 のみ 4.0 であることを表します。

vi を呼び出さずにコマンドから直接変更する場合

```
[root@n000 root]# qconf -mattr queue load_thresholds np_load_avg=4.0 all.q
```

vi を用いず、設定項目と値を指定して変更できます。この例では全ノードの負荷しきい値を 4.0 に変更します。

一部のキューインスタンスのみ変更する場合は、最後のキュー名指定のところをキューインスタンス名指定にします。

[root@n000 root]# qconf –mattr queue load_thresholds np_load_avg=4.0 all.q@n000

出荷時には全ホストを Grid Scheduler で利用できるよう設定しています。特定のホストを他の目的で使用したい場合には、qmod コマンドを用いてキューインスタンスを使用不可にすることで、一時的に切り離すことができます。複数のクラスタキューを設定している場合には、該当する全てのキューインスタンスを使用不可にしてください。

キューインスタンスを使用不可にする場合
[root@n000 root]# qmod –d all.q@n001

キューインスタンス all.q@n001 を使用不可にします。all.q@n001 にはそれ以上ジョブが割り振られません。

キューインスタンスを使用可能にする場合

[root@n000 root]# qmod –e all.q@n001

キューインスタンス all.q@n001 を使用可能にします。

実行ノードのグループ化

実行ノードをグループ化することによって、ジョブ投入時に使用するノードの範囲を限定したり、グループごとに割り当てるリソースを設定することができます。
グループ化には2つの方法があります。1つはクラスタキューを設定する方法、もう1つはホストグループを設定する方法です。それぞれ、作成したグループに対して指定可能なパラメータが異なります。

・クラスタキューについて

クラスタキューは、複数のノードに同じリソースを割り当てることができるキューです。リソースは、例えばスロット数、再実行可能かどうかのパラメータ、負荷しきい値、並列環境などです。同じクラスタキューには基本的に同じリソースが設定されますが、例外として個別のノードごとに異なるリソースを設定することも可能です。1つのノードが複数のクラスタキューに所属することができます。

出荷時の設定

出荷時には all.q というクラスタキューが設定されています。all.q のメンバーとして、ホストグループ@allhost が設定されています。すなわち、全ノードが all.q に属していることになります。その他、構成によっては追加のクラスタキューを設定する場合もあります。

クラスタキューのメンバーを変更する場合

```
[root@n000 root]# qconf -mq all.q
```

キュー設定の中の hostlist パラメータで、属するホストを決定します。ホスト名とホストグループ名のどちらでも記述することができます。

新規にクラスタキューを作成する場合

```
[root@n000 root]# qconf -aq
```

キュー設定ファイルのテンプレートが開きます。qname で名前を、hostlist で属するノードを決定してください。その他のパラメータは適宜変更してください。

・ホストグループについて

ホストグループとは、複数の実行ノードをまとめたもので、単にホスト名の代わりとなるものです。ホストグループ単位で直接リソースを設定するということはできず、必ずどこかのクラスタキューに所属したうえでの設定が必要となります。
新規に作成する場合、名前の最初に@を付ける必要がありますのでご注意ください。

出荷時の設定

全てのノードを含む@allhosts というグループが設定されています。構成や用途によっては、別のグループを追加する場合があります。

グループのメンバーを変更する場合

```
[root@n000 root]# qconf -mhgrp [groupname]
```

vi が呼び出されます。hostlist の行を書き換えてください。ホスト名あるいはホストグループ名を記述します。グループ名にの前には必ず@を付けてください。

新規にグループを作成する場合

```
[root@n000 root]# qconf -ahgrp
group_name @template
hostlist NONE
```

vi が呼び出され、ホストグループ設定のテンプレートが開きます。@template のところにグループ名を上書きし、NONE のところにノード名をスペース区切りで上書きしてください。グループ名の前には必ず@を付けてください。

実行ノード障害時の対応

sge_qmaster デーモンは、実行ノードがダウンした時など sge_execd デーモンと一定時間通信できなかった場合に該当ノードのキューを unknown 状態とします。その後キューの unknown 状態が一定時間続くと、sge_qmaster デーモンは該当キューで実行されていたジョブの再スケジューリングを試みます。このとき sge_qmaster デーモンは該当キューで実行されていたジョブを待機中リストに戻そうとしますが、running 状態のまま残ってしまうことがあります。その場合は qmod –r コマンドでジョブ ID を指定し、手動で再スケジューリングを行ってください。

関連する設定項目と出荷時の設定

・キュー設定　**rerun**
キュー内のジョブを再実行可能にするかどうか（デフォルトは TRUE）
・グローバル設定　**max_unheard**
ノードが通信不能になってからキューが unknown 状態になるまでの時間（デフォルトは 10 分）
・グローバル設定　**reschedule_unknown**
キューが unknown 状態になってからジョブを再スケジュールするまでの時間（デフォルトは 5 分）

設定を変更する場合

[root@n000 root]# qconf –mq all.q

vi が呼び出され、キュー設定ファイルの内容を変更できます。rerun 項目の値を書き換えてください。

[root@n000 root]# qconf –mconf
vi が呼び出され、グローバル設定ファイルの内容を変更できます。max_unheard 項目、reschedule_unknown 項目の値を書換えてください。

手動で再スケジューリングする場合

[root@n000 root]# qmod –r 146
ジョブ ID を指定して再スケジューリングします。ジョブ ID146 は待機中リストに戻された後、別のキューにディスパッチされます。

ユーザが同時に実行可能なジョブ数の制限

一部のユーザが大部分のプロセッサを独占してしまうことがないように、出荷時の設定では1ユーザが実行可能な最大ジョブ数を 32 としています。実際の利用状況に合わせて調整してください。1 ユーザが設定値を超える数のジョブを投入すると、制限を超えた分のジョブは待機中となります。待機中のジョブ数に制限はありません。

関連する設定項目と出荷時の設定

・スケジューラ設定　　maxujobs

1 ユーザが同時に実行可能なジョブの数（デフォルトは 32）

設定を変更する場合

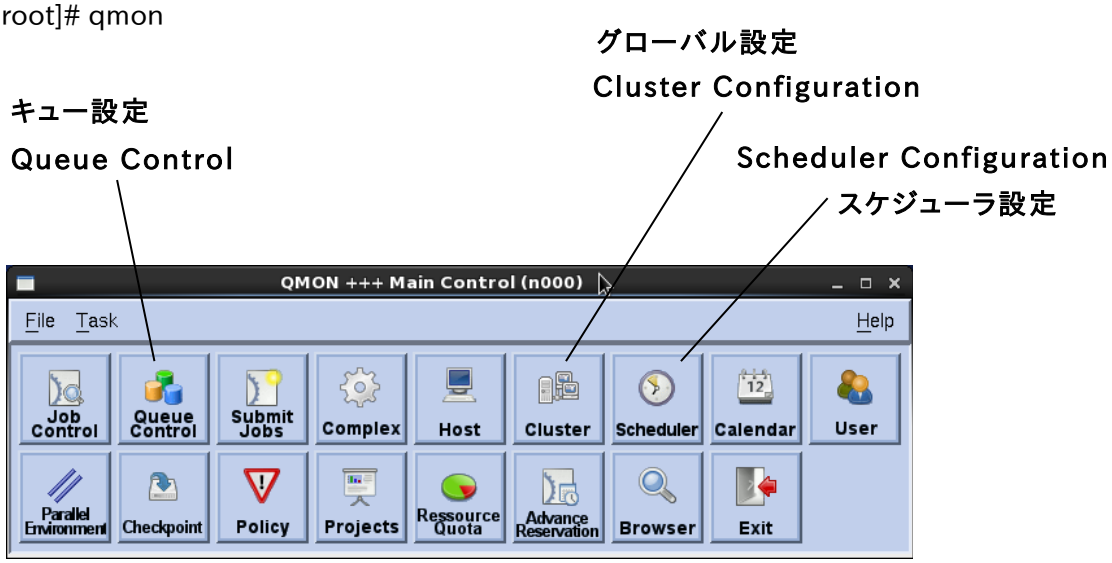
[root@n000 root]# qconf –msconf

vi が呼び出され、スケジューラ設定ファイルの内容を変更できます。maxujobs 項目の値を書き換えてください。

GUI で設定を変更する

GUI 上で設定変更を行うことも可能です。

```
[root@n000 root]# qmon
```



キュー設定

① Queue Control ボタンをクリック、Cluster Queues タブを表示し、all.q を選択して右欄の Modify ボタンをクリックすると、設定ウィンドウ（図 3）が出現　します。

② タブを選択して目的の設定値を表示させ、変更します。図 3 は General Configuration タブを選択した状態です。

slots, rerun　→General Configuration タブを選択
load_thresholds →Load/Suspend Thresholds タブを選択

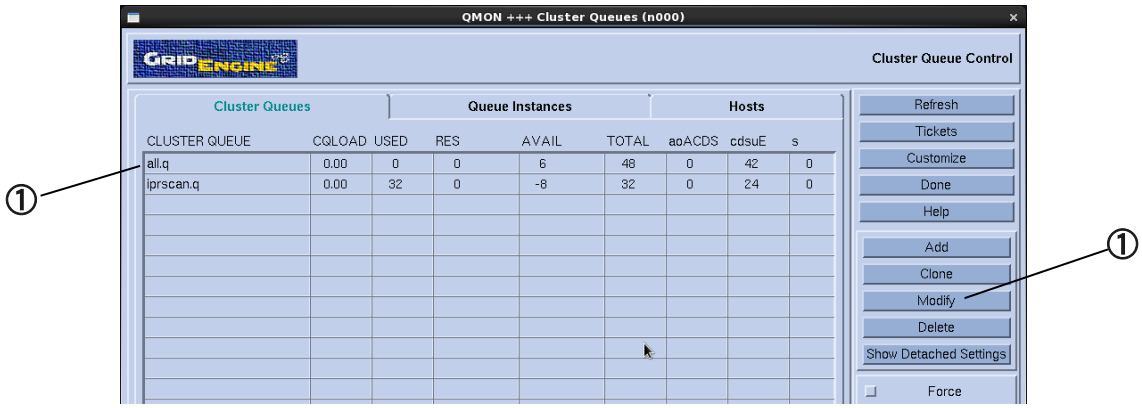


図 2

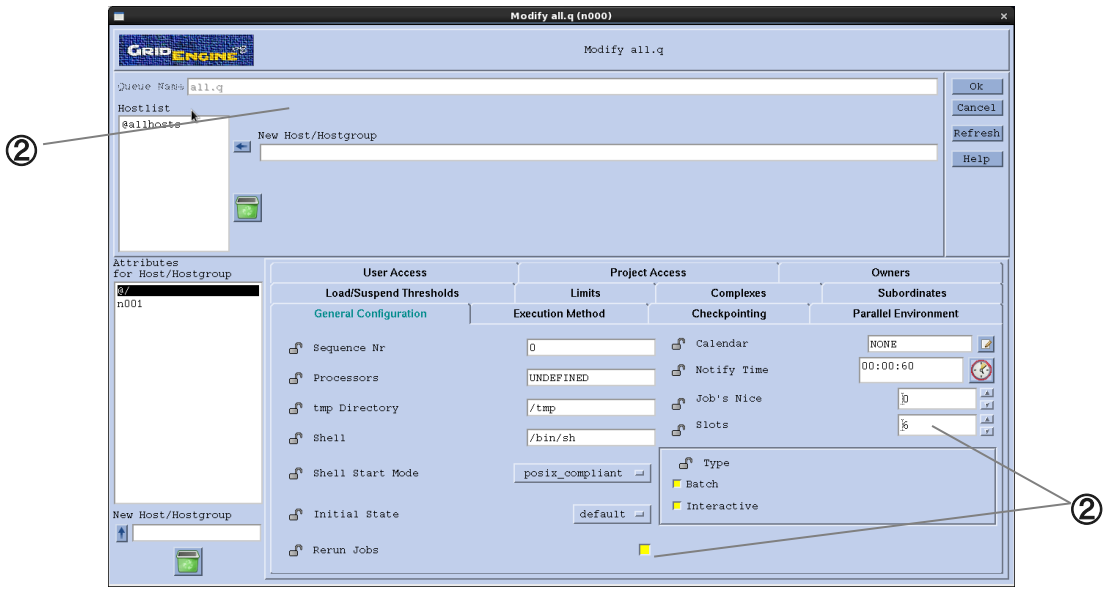


図 3

グローバル設定

図 4 は、メインウィンドウの Cluster Configuration ボタンをクリックした時に出現するウィンドウです。

① global が選択された状態で Modify ボタンをクリックすると、設定ウィンドウ（図 5）が出現します。

② 目的の設定値を変更します。max_unheard 項目や reschedule_unknown 項目は右上段にあります。

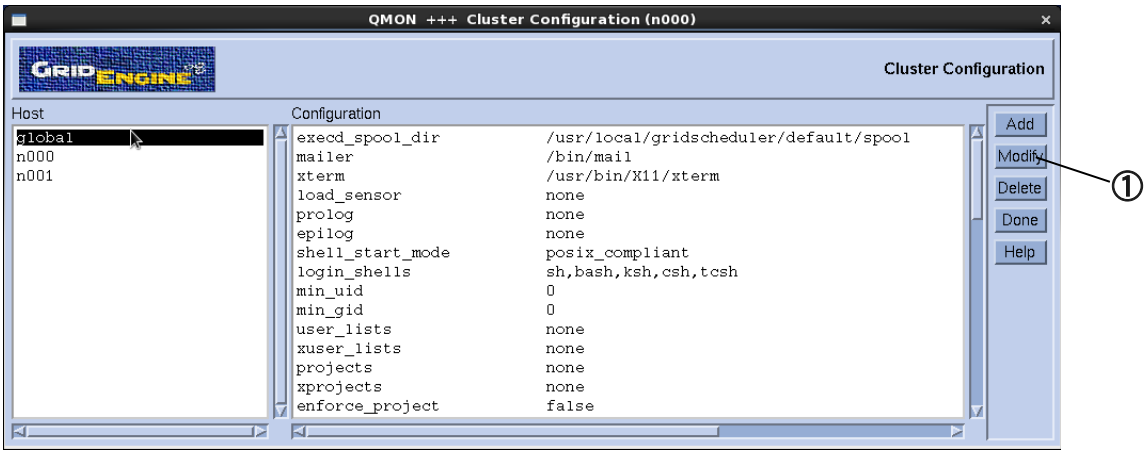


図 4

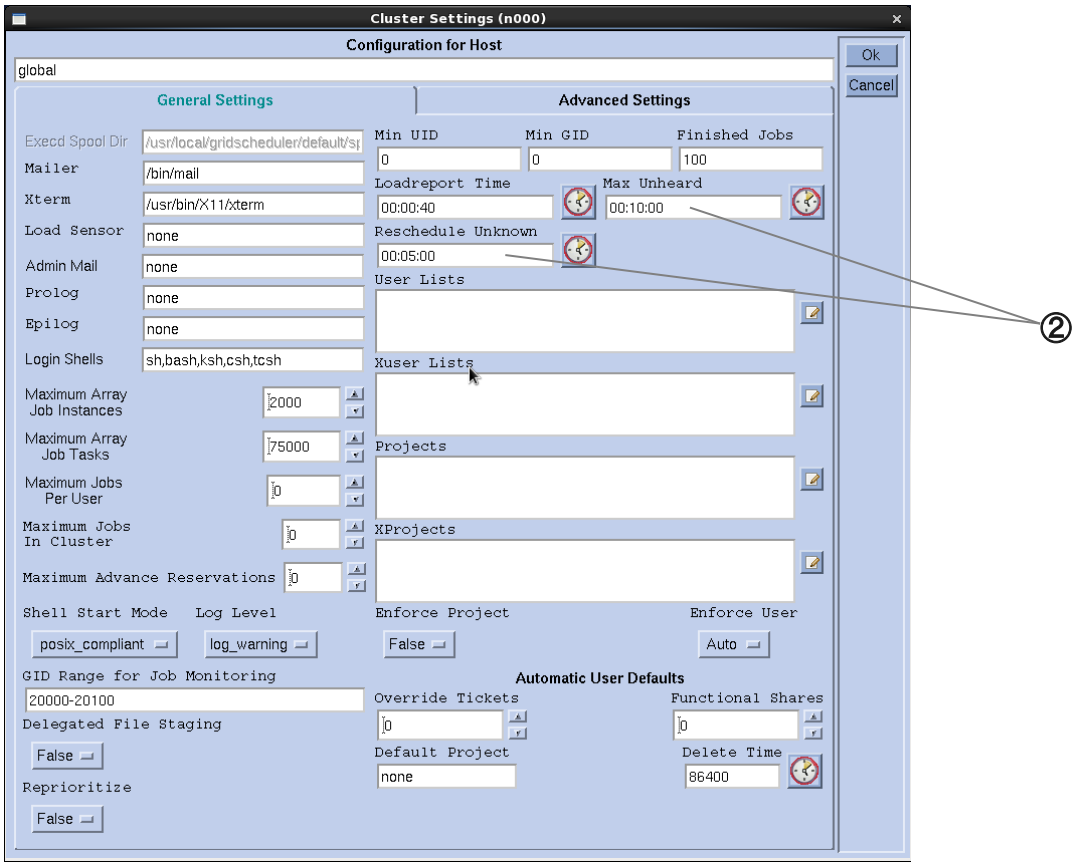


図 5

スケジューラ設定

図 6 は、メインウィンドウの Scheduler Configuration ボタンをクリックした時に出現するウィンドウです。

①目的の設定値を変更します。maxujobs は Max Jobs/User という項目名となっています。

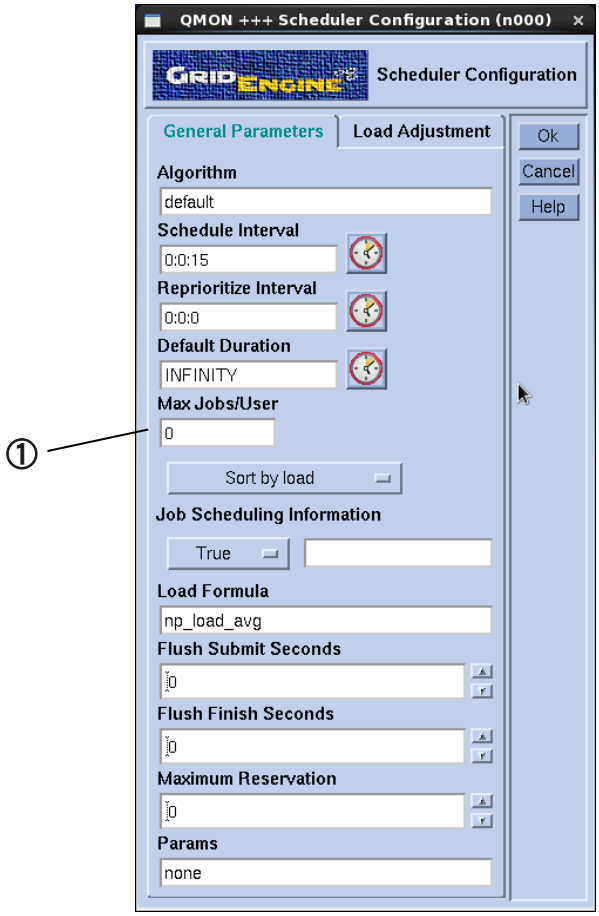


図 6