Sun N1 Grid Engine 6.1 ユーザーズガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–2162–10
2007 年 5 月

原典: Sun N1 Grid Engine 6.1 User's Guide

Part No: 820-0699-10

070524@17466

目次

図目次

表目次

例目次

はじめに

『N1 Grid Engine 6.1 ユーザーズガイド』には次の内容が記載されています。

- 複雑なコンピューティング環境における N1 Grid Engine 6.1 ソフトウェアの主な役割の説明
- 本製品の主要コンポーネントの説明および各コンポーネントの機能の定義
- N1 Grid Engine 6.1 ソフトウェア環境を知る上で重要な用語の意味

対象読者

このマニュアルは、N1 Grid Engine 6.1 ソフトウェアを使用するエンジニアおよび技術担当者を対象にしています。また、N1 Grid Engine 6.1 ソフトウェアを実行するネットワークに接続されたホストシステムの管理を担当している場合も、本書の概念を理解する必要があります。

内容の紹介

第 1 章では、N1 Grid Engine 6.1 ソフトウェアの概念と主要コンポーネントについて説明します。また、ユーザーコマンドの概要や `QMON` グラフィカルユーザーインタフェースも紹介します。

第 2 章では、ユーザー、キュー、ホストおよびジョブ属性など、N1 Grid Engine 6.1 ソフトウェアを実行するネットワーク接続ホストシステムのコンポーネントに関する情報を表示する方法について説明します。

第 3 章では、処理のためにジョブを発行する方法について説明します。

第 4 章では、ジョブとキューを監視および制御する方法について説明します。また、チェックポイントの設定に関する情報も紹介します。

第 5 章では、アカウンティングおよびレポートコンソールの使用方法について説明します。

第 6 章では、 C や Java で DRMAA API を使用して N1 Grid Engine 機能を自動化する方法を説明します。

第 7 章では、一般的な問題とその解決策を紹介します。

付録 A では、レポートデータベースデータモデルの詳細について説明します。

用語集は、製品固有の用語や表現とその定義について説明します。

関連マニュアル

N1 Grid Engine 6.1 ソフトウェアのマニュアルコレクションには、ほかに次のようなマニュアルがあります。

- 『Sun N1 Grid Engine 6.1 インストールガイド』
- 『Sun N1 Grid Engine 6.1 管理ガイド』
- 『Sun N1 Grid Engine 6.1 ご使用にあたって』

マニュアル、サポート、およびトレーニング

Sun の Web サイトでは、次のサービスに関する情報も提供しています。

- マニュアル (`http://jp.sun.com/documentation/`)
- サポート (`http://jp.sun.com/support/`)
- トレーニング (`http://jp.sun.com/training/`)

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1　表記上の規則

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `system%`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	`system%` **`su`** `password:`

表 P–1 表記上の規則 (続き)

字体または記号	意味	例
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、`rm` *filename* と入力します。
『』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

コマンド例のシェルプロンプト

次の表に、C シェル、Bourne シェル、および Korn シェルのデフォルトの UNIX® システムプロンプト、およびスーパーユーザーのプロンプトを紹介します。

表 P–2 シェルプロンプト

シェル	プロンプト
C シェル	`machine_name%`
C シェルのスーパーユーザー	`machine_name#`
Bourne シェルおよび Korn シェル	`$`
Bourne シェルおよび Korn シェルのスーパーユーザー	`#`

◆ ◆ ◆ 第 1 章

1 N1™ Grid Engine 6.1 ソフトウェアの概要

この章では、N1 Grid Engine 6.1 ソフトウェア (Grid Engine システム ともいう) を実行する、ネットワークに接続されたコンピュータホストシステムの背景について説明します。この章の内容は、次のとおりです。

グリッドコンピューティングおよびN1 Grid Engine 製品については、次の YouTube Web サイト にも優れた紹介があります。Introduction to Grid Engine (`http://www.youtube.com/watch?v=0JBsMitNnQ8`)。

グリッドコンピューティングとは

グリッドは、タスクを実行するコンピューティングリソースの集合です。もっとも簡単な形式のグリッドの場合、ユーザーには、強力な分散リソースへの単一のアクセスポイントを提供する大きなシステムのように見えます。この節の後半で説明するより複雑な形式のグリッドは、多くのアクセスポイントをユーザーに提供できます。いずれの場合でも、ユーザーはグリッドを *1* つのコンピュータリソースとして扱います。N1 Grid Engine 6.1 ソフトウェア (*Grid Engine* ソフトウェア) などのリソース管理ソフトウェアは、ユーザーが発行したジョブを受け付けます。ソフトウェアは、リソース管理ポリシーによって、ジョブがグリッド内の適切なシステムで実行されるようにスケジューリングを行います。ユーザーは、ジョブの実行場所を気にせずに、一度に数百万ものジョブを発行できます。

同じグリッドというものは1つもありません。また、あるサイズのグリッドですべての状況に対応することはできません。グリッドには次の3つの主要クラスがあり、単一システムから、数千のプロセッサを使用するスーパーコンピュータクラスの計算ファームまでをカバーしています。

- クラスタグリッドは、もっとも単純なクラスです。クラスタグリッドは、連動して動作するコンピュータホストの集合で構成されています。クラスタグリッドは、単一のプロジェクトまたは部署のユーザーに単一のアクセスポイントを提供します。
- キャンパスグリッドでは、組織内の複数のプロジェクトまたは部署がコンピューティングリソースを共有できます。組織はキャンパスグリッドを使用して、周期的な業務プロセスからレンダリング、データマイニングなどのさまざまなタスクを処理できます。
- グローバルグリッドは、組織の境界を越えて非常に大きな仮想システムを作成するキャンパスグリッドの集合です。ユーザーは、自分自身の組織内で使用できるリソースをはるかに超える計算能力を利用できます。

図 1–1 に3つのグリッドクラスを示します。クラスタグリッドでは、ユーザーのジョブはクラスタ内の1システムのみによって処理されます。ただし、そのユーザーのクラスタグリッドはより複雑なキャンパスグリッドの一部の場合もあります。そしてキャンパスグリッドはもっとも大きいグローバルグリッドの一部の場合があります。このような場合、ユーザーのジョブは世界中のどこにあるメンバー実行ホストでも処理できます。

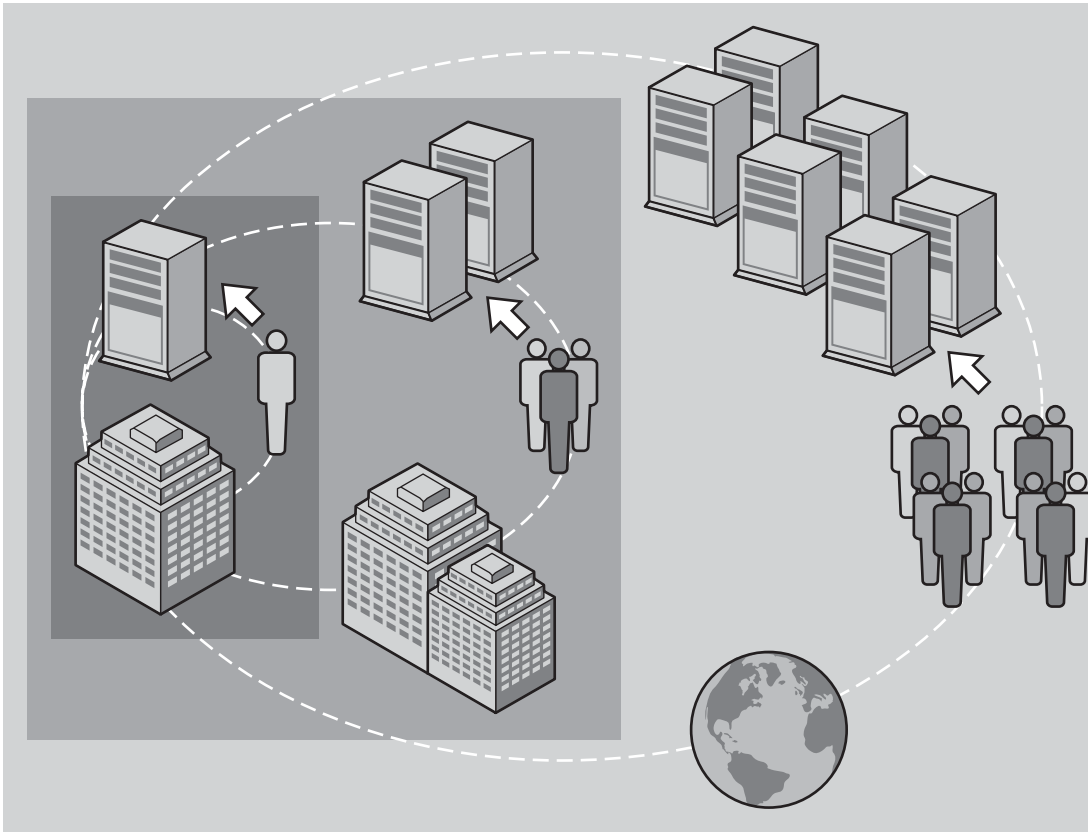


図 1–1　3 つのグリッドクラス

N1 Grid Engine 6.1 ソフトウェアは、キャンパスグリッドに必要なパワーと柔軟性を提供します。本製品を使用するとキャンパスグリッドの作成へのスムーズな移行をより簡単に実現できるので、既存のクラスタグリッドにとって有利です。Grid Engine システムは、キャンパスのすべての既存のクラスタグリッドを統合して、この移行を実行します。さらに、Grid Engine システムはグリッドコンピューティングモデルに初めて移行する 企業にとっても最適です。

Grid Engine ソフトウェアは、組織の技術スタッフと管理スタッフが策定する企業のリソースポリシーを基に、処理能力の提供を調整します。Grid Engine システムは、これらのポリシーを使用してキャンパスグリッド内の使用可能なコンピュータリ

ソースを調べます。システムはこれらのリソースを収集したあと、リソースの割り当てと提供を自動的に行い、キャンパスグリッド間の使用率を最適化します。

キャンパスグリッド内での協調を実現するために、グリッドを使用するプロジェクトの所有者は次の操作を実行する必要があります。

- ポリシー間の折衝
- 固有のプロジェクト要件に対する、手動による優先のためのポリシーの柔軟性の確保
- 自動的なポリシーの監視および実施

Grid Engine ソフトウェア は、コンピュータリソースの利用で競合する多くの部署とプロジェクトのエンタイトルメントを調整できます。

リソースとポリシーの管理による作業負荷の管理

Grid Engine システムは、異機種システムが混在する分散コンピューティング環境向けの高度なリソース管理ツールです。作業負荷管理とは、共有リソースの使用を制御して、生産性、適時性、サービスレベルなどの企業目標を最適な形で達成することです。作業負荷管理は、リソースの管理とポリシーの管理によって実現されます。サイトでは使用量とスループットが最大になるようにシステムが構成されますが、システムはさまざまなレベルの適時性と重要性をサポートしています。たとえば、ジョブの期限は適時性の例です。また、重要性の例にはジョブの優先順位やユーザーの共有などがあります。

Grid Engine ソフトウェアは、複数の共有リソースで構成される UNIX 環境向けの高度なリソース管理とポリシー管理を提供します。Grid Engine システムは、次の主要機能において標準的な負荷管理ツールより優れています。

- Grid Engine ソフトウェア によるサイト固有の管理ポリシーの実行を可能にする画期的な動的スケジューリングおよびリソース管理
- スケジューラに最新のジョブレベルのリソース消費およびシステム負荷情報を提供するためのパフォーマンスデータの動的な収集
- 証明セキュリティープロトコル (CSP) ベースの暗号化による拡張セキュリティー。よりセキュアなこのシステムのメッセージは、平文で転送されるのではなく、秘密鍵で暗号化されます。
- 生産性、適時性、およびサービスレベルなどの企業目標の定義と実行を実現するハイレベルなポリシー管理

Grid Engine ソフトウェア は、高度なコンピュータ処理が必要なタスクをグリッドに発行し、関連作業負荷の透過的な分散を行う方法をユーザーに提供します。ユーザーは、バッチジョブ、対話型ジョブ、および並列ジョブをグリッドに発行できます。本ソフトウェアは、管理者向けにジョブの監視と制御を行う包括的なツールを提供します。

本製品は、チェックポイント設定プログラムもサポートしています。チェックポイント設定ジョブは、ユーザーの介入なしに負荷の要求を基に、ワークステーション間で移行されます。

システムの動作

Grid Engine システムは次のすべてを実行します。

- 外部からのジョブを受け付けます。ジョブとは、ユーザーによるコンピュータリソースの要求です。
- ジョブが実行できるようになるまで、ジョブを保留領域に保管します。
- ジョブを保留領域から実行デバイスに送信します。
- 実行中のジョブを管理します。
- ジョブの完了時にジョブの実行レコードを記録します。

リソースを要求に対応させる

たとえば、大都市にある「マネーセンター」としての大手銀行を想像してください。銀行のロビーでは多数の顧客がサービスを受ける順番を待っています。顧客にはそれぞれ別の用事があります。ある顧客は、口座から少額のお金を引き出そうとしています。このすぐあとに入ってきた顧客は、銀行の投資スペシャリストと会う約束があります。この女性は、複雑な投機を始める前にアドバイスを受けたいと考えています。最初の 2 人の前にいる顧客は、彼女の前の 8 人の顧客同様に高額のローンに申し込みたいと考えています。

それぞれ異なるニーズを持つ顧客は、銀行にさまざまなタイプとレベルのサービスを求めています。この日の銀行には、口座からの単純な引き出しを処理できる行員は多数いるかもしれません。しかし、多数のローン申込者に対応するローン担当者は 1 人または 2 人しかいないということも考えられます。別の日には、この反対の状況になることもありえます。

結果、顧客はいたずらにサービスを待つことになります。顧客の多くは、彼らのニーズが即座に認識され、使用可能なリソースと一致させられさえすれば、即座にサービスを受けられるはずです。

Grid Engine システムが銀行の支店長だったならば、サービスは別のやり方で整理されたでしょう。

- 顧客は、銀行のロビーに入る時点で、名前、所属およびサービスニーズを明らかにするように頼まれます。
- 各顧客の到着時間が記録されます。
- ロビーで顧客が提供した情報を基に、銀行は次の顧客にサービスを提供します。

- ニーズに合ったリソースをすぐに使用できる顧客
- 優先順位がもっとも高い用件を持つ顧客
- 一番長くロビーで待っている顧客

- 「Grid Engine システム銀行」では、1 人の銀行員が同時に複数の顧客に対応できることもあります。Grid Engine システムは、もっとも負荷が少なくもっとも適した銀行員に新しい顧客を割り当てようとします。
- 銀行の支店長として、Grid Engine システムは銀行にサービスポリシーを定義させます。一般的なサービスポリシーは、たとえば次のようになります。
  - 利益が大きいので、企業顧客には優先的なサービスを提供します。
  - これまで良いサービスを受けていない特定の顧客グループに対して、十分サービスが提供されることを確認します。
  - 約束のある顧客がタイムリーな対応を受けられるようにします。
  - 銀行役員から直接要請された特定の顧客を優先します。
- このようなポリシーが、Grid Engine システムという支店長により自動的に実行、監視および調整されます。優先アクセスを持つ顧客は、より早くサービスを受けられます。このような顧客には行員がより多くの便宜を図り、この顧客はその支援をほかの顧客と共有するはずです。支店長である Grid Engine は、顧客が待ち状態になっている場合はそれを認識します。支店長は、サービスレベルを調整してすぐに対応し、銀行のサービスポリシーを遵守します。

ジョブとキュー

Grid Engine システムでは、ジョブは銀行の顧客に相当します。ジョブは、ロビーではなくコンピュータの保留領域で待機しています。ジョブにサービスを提供するキューは、銀行員に相当します。銀行の顧客の場合と同じで、使用可能なメモリー、実行速度、使用可能なソフトウェアライセンスなどのニーズといった各ジョブの要件は、それぞれまったく異なります。特定のキューだけが、それに対応サービスを提供できる場合もあります。

たとえば、Grid Engine ソフトウェア は次のように使用可能なリソースとジョブ要件を調整します。

- Grid Engine システムを通してジョブを発行するユーザーは、ジョブの要件プロファイルを宣言します。さらに、システムはユーザーの ID を検索します。システムは、ユーザーのプロジェクトまたはユーザーグループへの所属も検索します。ユーザーがジョブを発行した時間も保存されます。
- キューが新しいジョブを実行できるようになると、Grid Engine システムはそのキューに適したジョブを判断します。システムは、優先順位がもっとも高いジョブまたは待ち時間がもっとも長いジョブを即座に振り分けます。
- キューでは、多数のジョブを同時に実行できます。Grid Engine システム は、負荷がもっとも少なくもっとも適したキューで新しいジョブを開始しようとします。

使用ポリシー

クラスタの管理者は、サイトに適した条件によってカスタマイズされたハイレベルの使用ポリシーを定義できます。次の 4 つの使用ポリシーが使用できます。

- 緊急度。 このポリシーを使用する場合、各ジョブの優先順位の基準は緊急度の値になります。緊急度の値は、ジョブのリソース要件、ジョブの期限指定、およびジョブの実行までの待機時間を元に計算されます。
- 機能。 このポリシーを使用する場合、管理者は特定のユーザーグループやプロジェクトなどへのユーザーまたはジョブの所属によって特別な取り扱いを行うことができます。
- 共有ベース。 このポリシーを使用する場合、サービスのレベルは、割り当てられた共有、その他のユーザーやユーザーグループの共有、ユーザー全員の過去のリソース使用率、およびシステムにおける現在のユーザーの存在によって異なります。
- 優先。このポリシーでは、クラスタ管理者が手動で介入して、ポリシーの自動的な実行を変更する必要があります。

ポリシー管理は、クラスタ内の共有リソースの使用を自動的に管理して、管理目標を最適な形で達成します。優先順位の高いジョブは優先的に振り分けられます。このようなジョブは、ほかのジョブとリソースを争っている場合に、より大きな CPU 共有を取得できます。Grid Engine ソフトウェア は、すべてのジョブの進行状況を監視し、適宜ポリシーで定義されている目標と照らし合わせて、ジョブの相対的な優先順位を調整します。

チケットの使用によるポリシーの管理

機能、共有ベース、および優先ポリシーは、チケットと呼ばれる Grid Engine システムの概念によって定義されます。 チケットは、株式会社の株式保有数に例えることができます。株式保有数が多いと、その株主の重要度は高くなります。株主 A が株主 B の 2 倍の株式を持っている場合、A の投票も B の 2 倍になります。したがって、株主 A は 2 倍重要だということになります。同様に、ジョブのチケット保有数が多いと、そのジョブの重要性は高くなります。ジョブ A がジョブ B の 2 倍のチケットを持っている場合、ジョブ A はジョブ B の 2 倍リソースを使用できます。

ジョブは、 機能、共有ベースおよび優先ポリシーからチケットを検索できます。チケットの合計数および各チケットポリシーから検索された数は、時間と共に変わります。

管理者は、各チケットポリシーに割り当てられるチケット数を全体として管理します。ジョブのチケット割り当てと同じように、この割り当てでは、チケットポリシー同士の相対的な重要性が判断されます。特定のチケットポリシーに割り当てられたチケットプールを通して、管理者は Grid Engine システムを複数の方法で実行できます。たとえば、システムを共有ベースモードだけで実行することもできます。

または、90% が共有ベースモードで 10% が機能モードというように、モードを組み合わせてシステムを実行することもできます。

緊急度ポリシーによるジョブ優先順位の割り当て

緊急度ポリシーは、ほかの 2 つのジョブ優先順位指定と組み合わせて使用することができます。

- 機能、共有ベース、および優先ポリシーによって割り当てられるチケットの数
- `qsub –p` コマンドによって指定される優先順位

ジョブには、次の 3 つを元に計算される緊急度の値を割り当てることができます。

- ジョブのリソース要件
- 実行前にジョブが待機する時間の長さ
- ジョブが実行を終了する時間、つまりジョブの期限

管理者は、これらの各要素の重要性を別々に重み付けして、ジョブ全体の緊急度の値を割り出すことができます。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の第 5 章「ポリシーとスケジューラの管理」を参照してください。

図 1–2 に、ポリシー間の相関関係を示します。

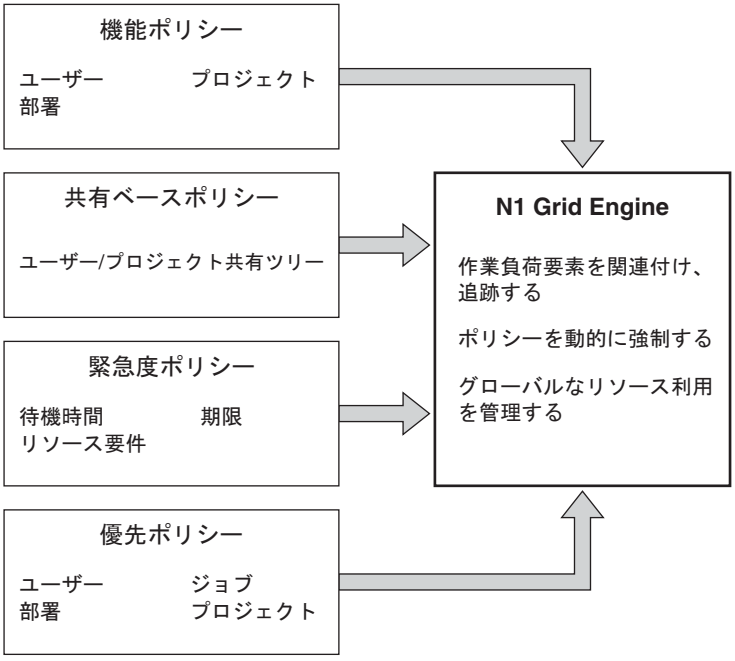


図 1–2　Grid Engine システムのポリシー間の関係

Grid Engine システムコンポーネント

次の節では、もっとも重要な Grid Engine システムコンポーネントの機能について説明します。

ホスト

Grid Engine システムの基本的なホストは、次の 4 種類です。

- マスターホスト
- 実行ホスト
- 管理ホスト
- 発行ホスト

マスターホスト

マスターホストは、クラスタ全体の動作の中心に位置づけられます。マスターホストでは、マスターデーモン `sge_qmaster` と スケジューラ デーモン `sge_schedd` が実行されます。この 2 つのデーモンは、キューやジョブなどのすべての Grid Engine システムコンポーネントを制御します。デーモンは、コンポーネントのステータスやユーザーのアクセス権などに関するテーブルを維持しています。

デフォルトでは、マスターホストが管理ホストでもあり発行ホストでもあります。

実行ホスト

実行ホストは、ジョブの実行権を持つシステムです。したがって、実行ホストにはキューインスタンスが関連付けられています。実行ホストは、実行デーモン `sge_execd` を実行します。

管理ホスト

管理ホストは、Grid Engine システムの管理処理をすべて実行する権限を持つホストです。

発行ホスト

発行ホストでユーザーが発行および制御できるのは、バッチジョブだけです。特に、発行ホストにログインしたユーザーは、`qsub` コマンドでジョブを発行し、`qstat` コマンドでジョブステータスを監視し、Grid Engine システムの OSF/1 Motif グラフィカルユーザーインタフェースである `QMON` を使用できます。QMON については、30 ページの「Grid Engine システムのグラフィカルユーザーインタフェースである QMON」に説明があります。

注 – システムは、複数の種類のホストとして機能できます。

デーモン

3 つのデーモンが Grid Engine システムの機能を提供します。

sge_qmaster – マスターデーモン

クラスタの管理およびスケジューリング処理の中心となる sge_qmaster は、ホスト、キュー、ジョブ、システム負荷、およびユーザーの権限についてのテーブルを維持します。sge_qmaster は、sge_schedd からスケジューリングに関する決定を受け取り、適切な実行ホストの sge_execd に処理を要求します。

sge_schedd – スケジューラデーモン

スケジューリングデーモンは、sge_qmaster によってクラスタのステータスの最新ビューを維持します。スケジューリングデーモンは、スケジューリングに関する次の決定を行います。

- どのジョブがどのキューに割り振られるか
- 共有、優先順位、または期限を維持するためにどのようにジョブを再整理し、優先順位を再設定するか

デーモンはこれらの決定を sge_qmaster に転送し、sge_qmaster は、要求された処理を開始します。

sge_execd – 実行デーモン

実行デーモンは、ホスト上のキューインスタンスおよびこれらのキュー インスタンス内のジョブの実行を担当しています。実行デーモンは、定期的にジョブステータスまたはホストの負荷などの情報を sge_qmaster に転送します。

キュー

キューは、1 台以上のホストで同時に実行できるジョブのクラス用のコンテナです。キューは、ジョブを移行できるかどうかなどの特定のジョブの属性を決定します。実行ジョブはその有効期間中キューに関連付けられています。キューに関連付けられていることで、ジョブで発生する事象の一部が影響を受けます。たとえば、キューが一時停止されると、そのキューに関連付けたすべてのジョブも一時停止されます。

ジョブをキューに直接発行する必要はありません。ジョブの要件プロファイルだけを指定すればすみます。プロファイルには、メモリー、オペレーティングシステム、使用可能なソフトウェアなどの要件が含まれます。Grid Engine ソフトウェア

は、適切なキューと実行負荷が少ない適切なホストにジョブを自動的に割り振ります。指定したキューにジョブを発行すると、ジョブはそのキューと結合されます。結果として、Grid Engine システム デーモンは、負荷が少なくより適したデバイスを選択できなくなります。

キューは 1 台のホストに存在する場合と、複数のホストに及ぶ場合があります。このため、Grid Engine システムのキューはクラスタキュー とも呼ばれます。ユーザーと管理者は、クラスタキューを使用すると、1 つのキュー構成によって実行ホストのクラスタを同時に実行できます。クラスタキューに関連付けられた各ホストは、クラスタキューからそれぞれのキューインスタンスを受け取ります。

クライアントコマンド

コマンド行ユーザーインタフェースは、次の作業を行う補助的なプログラム (コマンド) セットです。

- キューの管理
- ジョブの発行と削除
- ジョブステータスのチェック
- キューとジョブの一時停止と有効化

Grid Engine システムは、次の補助プログラムセットを提供します。

- `qacct` –クラスタログファイルから任意のアカウンティング情報を抽出します。
- `qalter` – 発行済みで保留中となっているジョブの属性を変更します。
- `qconf` – クラスタ構成とキュー構成用のユーザーインタフェースを提供します。
- `qdel` – ジョブまたはジョブのサブセットに信号を送信する方法をユーザー、オペレータ、または管理者に提供します。
- `qhold` – 発行済みのジョブを実行しないようにします。
- `qhost` – 実行ホストのステータス情報を表示します。
- `qlogin` – より負荷が少なく適切なホストを自動的に選択して、`telnet` などのログインセッションを開始します。
- `qmake` – 標準 UNIX の `make` 機能の代わりです。`qmake` は、 `make` の各手順を適切なマシンのクラスタ間で分散する `make` の機能を拡張したものです。
- `qmod` – 所有者がキューを一時停止したり、有効にしたりできるようにします。このキューと関連付けられている現在アクティブなプロセスすべても信号で伝えられます。
- `qmon` – X Windows Motif コマンドインタフェースと監視機能を提供します。
- `qresub` – 実行中または保留中のジョブをコピーして、新しいジョブを作成します。
- `qrls` – `qhold` などで割り当てられた待機状態からジョブを解放します。

- qrsh – 次のようなさまざまな目的に使用できます。
  - Grid Engine システムによる対話型アプリケーションのリモート実行。 qrsh は、標準の UNIX 機能である rsh と互換性があります
  - 実行時に端末 I/O と端末コントロールをサポートするバッチジョブの発行。端末 I/O には標準出力、標準エラーおよび標準入力が含まれます
  - バッチジョブの終了までアクティブな発行クライアントの提供
  - Grid Engine ソフトウェア が制御する並列ジョブのタスクのリモート実行
- qselect – 指定された選択基準に対応するキュー名のリストの印刷。qselect の出力は、通常選択されたキューセットに処理を適用するほかの Grid Engine システムコマンドに送信されます。
- qsh – 負荷の少ないホストの xterm 内で対話型シェルを開く。このシェル内では全種類の対話型ジョブを実行できます。
- qstat – クラスタに関連付けられたすべてのジョブとキューのステータスの提供
- qsub – バッチジョブを Grid Engine システムに発行するためのユーザーインタフェース
- qtcsh – 一般的に使用される UNIX C シェル (csh) の派生コマンド tcsh の、完全な互換性を持つ代用コマンド。 qtcsh は、指定アプリケーションの実行を適切で負荷が少ないホストに Grid Engine ソフトウェア を通して透過的に分散するという拡張機能を持ったコマンドシェルを提供します。

Grid Engine システムのグラフィカルユーザーインタフェースである QMON

ユーザーは、グラフィカルユーザーインタフェース (GUI) ツールである QMON を使用して、ほとんどの Grid Engine システムタスクを実行できます。図 1–3 に、ユーザーと管理者の機能の両方を開始することが多い「QMON Main Control」ウィンドウを示します。「Main Control」ウィンドウの各アイコンは、クリックするとさまざまなタスクを開始できる GUI ボタンです。機能の説明にもなっているボタンの名前を見るには、ボタンの上にポインタを置きます。

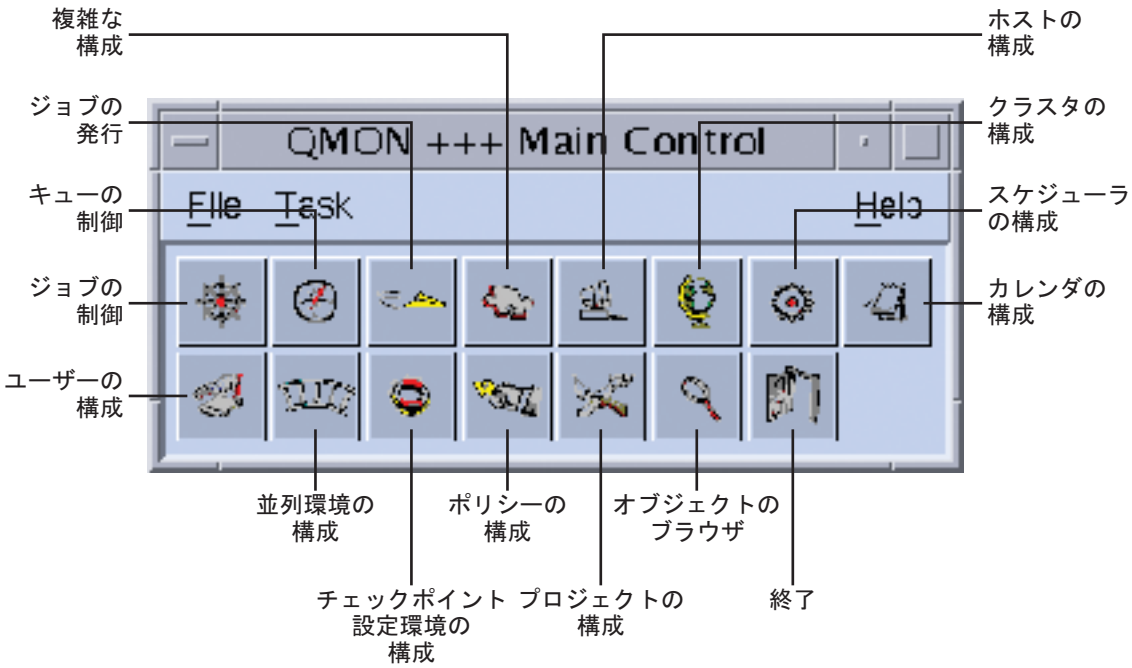


図 1–3 「QMON Main Control」ウィンドウの定義

◆ ◆ ◆ 第 2 章

Grid Engine システムの概要

この章では、ユーザー、キュー、ホストおよびジョブ属性などの Grid Engine システムコンポーネントに関する情報の表示方法について説明します。また、本ソフトウェアの使用を開始するにあたって役立つ基本概念や用語も紹介します。本製品の詳細な内容説明については、第 1 章を参照してください。

この章には、次の作業の実行方法も記載されています。

「QMON Main Control」ウィンドウ

Grid Engine システムの特長は、グラフィカルユーザーインタフェース (GUI) コマンドツールである「QMON Main Control」ウィンドウです。ユーザーは、「 QMON Main Control」ウィンドウで、ジョブの発行、ジョブの制御、および重要情報の収集など、ほとんどの Grid Engine システムの機能を実行できます。

「 QMON Main Control」ウィンドウの起動

「 QMON Main Control」ウィンドウを起動するには、コマンド行から次のコマンドを入力します。

```
% qmon
```

メッセージウィンドウが表示されてから、「QMON Main Control」ウィンドウが表示されます。

アイコンの意味は、図 1–3 を参照してください。アイコンボタンの名前は、ボタン上にポインタを置くと画面に表示されます。名前は、ボタンの機能を表しています。

このガイドの手順の多くでは、「 QMON Main Control」ウィンドウを使用する必要があります。

QMON のカスタマイズ

QMON の使用感の大部分は、特別に設計されたリソースファイルに定義されます。適切なデフォルトが *sge-root*/qmon/Qmon に編集されています。なおここには、 サンプルリソースファイルも含まれています。

クラスタ管理では、次のいずれかを行うことができます。

- /usr/lib/X11/app-defaults/Qmon などの標準的な場所へのサイト固有のデフォルトのインストール
- 標準の .Xdefaults ファイルまたは .Xresources ファイルへの QMON 固有のリソース定義の保存
- XAPPLRESDIR などの標準検索パスによって参照される場所へのサイト固有の Qmon ファイルの保存

実際の環境でこのどれかが当てはまるかどうかは、管理者にお問い合わせください。

さらに、ユーザーは個人的な環境設定も行うことができます。ユーザーは、Qmon ファイルも変更できます。Qmon ファイルは、ホームディレクトリや個人用の XAPPLRESDIR 検索パスが指す場所へ移動できます。またユーザーは、必要なリソース定義を個人用の .Xdefaults または .Xresources ファイルに保存することもできます。さらに、個人用の Qmon リソースファイルを xrdb コマンドによってインストールすることもできます。xrdb コマンドは、動作中に使用できます。また、 .xinitrc リソースファイルなどの X11 環境の起動時に xrdb を使用することもできます。

カスタマイズ可能な設定の詳細は、サンプル Qmon ファイルのコメント行を参照してください。

「Job Customize」ダイアログボックスおよび「Queue Customize」ダイアログボックスを使用して、qmon をカスタマイズすることもできます。これらのダイアログボックスは、95 ページの「「Job Control」ディスプレイのカスタマイズ」と111 ページの「クラスタキューおよびキューインスタンスのフィルタリング」に示されています。両方のダイアログボックスでは、「Save」ボタンを使用してフィルタリング定義と表示定義をホームディレクトリの .qmon_preferences ファイルに保存できます。QMON の再起動時にはこのファイルが読み込まれ、QMON は以前に定義された動作を再度有効にすることができます。

ユーザーとユーザーカテゴリ

Grid Engine システムのユーザーは 4 つのカテゴリに分類されます。各カテゴリユーザーは、それぞれの Grid Engine システムコマンドセットにアクセスできます。

- 管理者 – 管理者は、Grid Engine システムのすべての操作を実行できます。デフォルトでは、すべての管理ホストのスーパーユーザーは管理者特権を持っています。
- オペレータ – オペレータは、管理者と同じコマンドの多くを実行できますが、たとえばキューの追加、削除、または変更などの構成変更は実行できません。
- 所有者 – キューの所有者は、所有するキューを一時停止したり有効にしたりすることができます。キューの所有者は、所有するキュー内のジョブを一時停止したり有効にしたりすることもできます。キューの所有者は、その他の管理権は持ちません。
- ユーザー – 36 ページの「ユーザーのアクセス権」で説明するように、特定のアクセス権を持ちます。ユーザーに、クラスタ管理またはキュー管理はありません。

表 2–1 に、複数のユーザーカテゴリで使用可能なコマンド機能を示します。

表 2–1 ユーザーカテゴリおよび関連コマンド機能

コマンド	管理者	オペレータ	所有者	ユーザー
qacct	完全 (フル)	完全 (フル)	自分のジョブのみ	自分のジョブのみ
qalter	完全 (フル)	完全 (フル)	自分のジョブのみ	自分のジョブのみ
qconf	完全 (フル)	システム設定の変更以外	構成の表示のみとアクセス権	構成の表示のみとアクセス権
qdel	完全 (フル)	完全 (フル)	自分のジョブのみ	自分のジョブのみ
qhold	完全 (フル)	完全 (フル)	自分のジョブのみ	自分のジョブのみ

表 2–1 ユーザーカテゴリおよび関連コマンド機能 (続き)

コマンド	管理者	オペレータ	所有者	ユーザー
qhost	完全 (フル)	完全 (フル)	完全 (フル)	完全 (フル)
qlogin	完全 (フル)	完全 (フル)	完全 (フル)	完全 (フル)
qmod	完全 (フル)	完全 (フル)	自分のジョブとキューのみ	自分のジョブのみ
qmon	完全 (フル)	システム設定の変更以外	構成変更以外	構成変更以外
qrexec	完全 (フル)	完全 (フル)	完全 (フル)	完全 (フル)
qselect	完全 (フル)	完全 (フル)	完全 (フル)	完全 (フル)
qsh	完全 (フル)	完全 (フル)	完全 (フル)	完全 (フル)
qstat	完全 (フル)	完全 (フル)	完全 (フル)	完全 (フル)
qsub	完全 (フル)	完全 (フル)	完全 (フル)	完全 (フル)

ユーザーのアクセス権

管理者は、キューおよび並列環境インタフェースなどの機能へのアクセスを制限することができます。アクセスは、特定のユーザーまたはユーザーグループに対して制限されます。

注 – Grid Engine ソフトウェアは、クラスタ管理によって設定されたアクセス制限を自動的に考慮します。次の節は、自分のアクセス権を知りたい場合にだけお読みください。

アクセス権を制限するために、管理者はアクセスリスト (ACL) を作成および管理します。ACL には、ユーザー名と UNIX グループ名が含まれています。この ACL は、キューまたは並列環境インタフェース構成内のアクセス許可リストまたはアクセス禁止リストに追加されます。詳細は、queue_conf(5) または sge_pe(5) のマニュアルページを参照してください。

アクセス許可リスト内の ACL に属しているユーザーは、キューや並列環境インタフェースにアクセスできます。アクセス禁止リストの ACL のメンバーであるユーザーは、該当リソースにアクセスできません。

ACL は、ユーザーがアクセスできるプロジェクト、つまりユーザーの作業の上位にあるプロジェクトを定義するためにも使用されます。管理者は、プロジェクトごとにクラスタリソースへのアクセスを制限することもできます。

「User Configuration」ダイアログボックスは、「QMON Main Control」ウィンドウの「User Configuration」ボタンをクリックすると開きます。このダイアログボックスを使用すると、アクセス権を持つ ACL に対して照会を行うことができます。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の第 4 章「ユーザーアクセスの管理」を参照してください。

プロジェクトアクセスは、「QMON Main Control」ウィンドウの「Project Configuration」アイコンをクリックすることで表示できます。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「プロジェクトの定義」を参照してください。

コマンド行から次のコマンドを使用して、現在構成されている ACL のリストを取得できます。

```
% qconf -sul
```

次のコマンドを使用すると、1 つ以上のアクセスリストのエントリを一覧表示できます。

```
% qconf -su acl-name[,...]
```

ACL は、ユーザーアカウント名と UNIX グループ名で構成されます。UNIX グループ名の前には @ マークが付いています。このようにして、アカウントが属する ACL を決定できます。

注 – newgrp コマンドで一次 UNIX グループを切り替える権限がある場合は、アクセス権が変更される場合があります。

アクセスできるキューまたは並列環境インタフェース、およびアクセスが拒否されるキューまたは並列環境インタフェースを確認することができます。38 ページの「キューとキュープロパティーの表示」および『Sun N1 Grid Engine 6.1 管理ガイド』の「QMON を使用した並列環境の構成」で説明されているとおり、キューまたは並列環境インタフェース構成を照会してください。

アクセス許可リストの名前は user_lists です。アクセス禁止リストの名前は xuser_lists です。お使いのユーザーアカウントまたは一次 UNIX グループがアクセス許可リストに関連付けられている場合は、該当リソースにアクセスできます。アクセス禁止リストに関連付けられている場合は、キューまたは並列環境インタフェースにはアクセスできません。両方のリストが空の場合は、有効なアカウントを持つユーザーは全員、該当リソースにアクセスできます。

コマンド行から次のコマンドを使用して、プロジェクト構成を制御できます。

```
% qconf -sprjl
% qconf -sprj project-name
```

上のコマンドは、定義されたプロジェクトのリストと特定のプロジェクト構成のリストをそれぞれ表示します。プロジェクトは ACL を使用して定義します。前の段落の説明どおり、ACL 構成を照会してください。

プロジェクトにアクセスできる場合は、プロジェクトの下位のジョブを発行できます。このジョブは、コマンド行から次のコマンドを使用して発行できます。

```
% qsub -P project-name options
```

クラスタ構成、ホスト構成、およびキュー構成は、ACL と同じようにプロジェクトアクセスを定義します。これらの構成では、定義のために `project_lists` と `xproject_lists` パラメータが使用されます。

管理者、オペレータ、および所有者

Grid Engine システム管理者のリストを表示するには、次のコマンドを使用してください。

```
% qconf -sm
```

オペレータのリストを表示するには、次のコマンドを使用してください。

```
% qconf -so
```

注 – デフォルトでは、管理ホストのスーパーユーザーは管理者とみなされます。

特定のキューの所有者であるユーザーは、38 ページの「キューとキュープロパティーの表示」の説明どおり、キュー構成に含まれています。次のコマンドを入力すると、キュー構成を表示できます。

```
% qconf -sq {cluster-queue | queue-instance | queue-domain}
```

このキュー構成エントリの名前は `owner_list` です。

キューとキュープロパティーの表示

サイトの Grid Engine システムを最大限活用するには、キューの構造をよく知る必要があります。また、お使いの Grid Engine システムのキュープロパティーも知っておかなければなりません。

キューリストの表示

「QMON Queue Control」ダイアログボックスの図と説明は、103 ページの「QMON によるキューの監視と制御」を参照してください。このダイアログボックスは、インストールされているキューとキューの現在の状態の簡単な概要を表示します。

キューのリストを表示するには、コマンド行から次のコマンドを入力してください。

```
% qconf -sql
```

キュープロパティーの表示

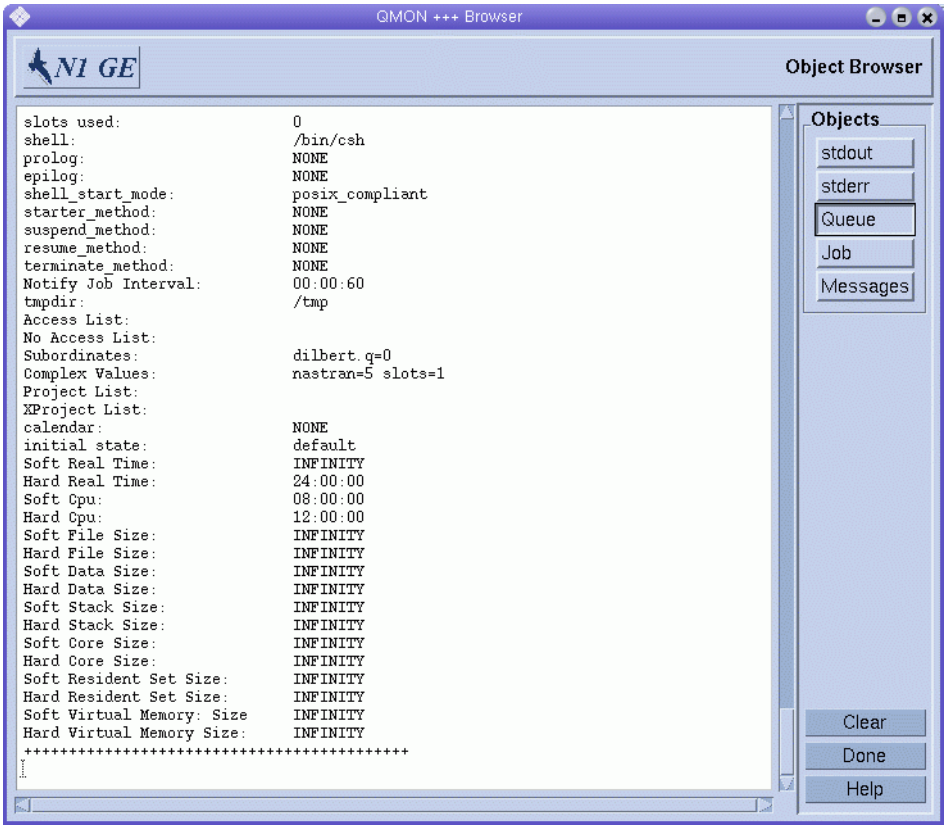
キュープロパティーは、QMON またはコマンド行で表示できます。

▼ QMON によりキュープロパティーを表示する

1. 「QMON **Main Control**」ウィンドウを起動します。
2. 「**Queue Control**」ボタンをクリックします。
   「Cluster Queue」ダイアログボックスが表示されます。
3. キューを選択し、「**Show Detached Settings**」をクリックします。
   「Browser」ダイアログボックスが表示されます。
4. 「**Browser**」ダイアログボックスで「**Queue**」をクリックします。
5. 「**Cluster Queue**」ダイアログボックスで「**Queue Instances**」タブをクリックします。
6. キューインスタンスを選択します。
   「Browser」ダイアログボックスに選択したキューインスタンスのキュープロパティーが一覧表示されます。

例 2–1 キュープロパティー情報

次の図に、表示されるキュープロパティー情報の例を示します。



コマンド行からのキュープロパティーの表示

コマンド行からキュープロパティーを表示するには、次のコマンドを入力してください。

```
% qconf -sq {queue | queue-instance | queue-domain}
```

上の図のような情報が表示されます。

キュープロパティー情報の解釈

各キュープロパティーの詳細は、 queue_conf(5) のマニュアルページを参照してください。

次に重要性の高いパラメータをいくつか示します。

- qname – 要求されたとおりのキュー名。
- hostlist – キューに関連するホストとホストグループのリスト。

- processors – キューがアクセスできるマルチプロセッサシステムのプロセッサ。

注意 – この値は、確実に必要な場合以外は変更しないでください。

- qtype – このキューで実行できるジョブのタイプ。現在、キュータイプはバッチまたは対話型のどちらかです。
- slots – そのキューで同時に実行できるジョブの数。
- owner_list – 38 ページの「管理者、オペレータ、および所有者」で説明されているキューの所有者。
- user_lists – このパラメータにリストされているユーザーアクセスリスト内のユーザーまたはグループ識別子は、キューにアクセスできます。詳細は、36 ページの「ユーザーのアクセス権」を参照してください。
- xuser_lists – このパラメータにリストされているユーザーアクセスリスト内のユーザーまたはグループ識別子は、キューにアクセスできません。 詳細は、36 ページの「ユーザーのアクセス権」を参照してください。
- project_lists – このパラメータにリストされているプロジェクト識別子で発行されたジョブは、キューにアクセスできます。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「プロジェクトの定義」を参照してください。
- xproject_lists – このパラメータにリストされているプロジェクト識別子で発行されたジョブは、キューにアクセスできません。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「プロジェクトの定義」を参照してください。
- complex_values – このキューの指定に従って、特定の複雑なリソース属性に対して機能を割り当てます。詳細は、42 ページの「要求可能な属性」を参照してください。

ホストとホストの機能

「QMON Main Control」ウィンドウの「Host Configuration」ボタンをクリックすると、ユーザーのクラスタ内のホストに関連する機能の概要が表示されます。ただし、管理者特権を持たない場合は、構成を変更することはできません。

ホスト構成ダイアログボックスは、『Sun N1 Grid Engine 6.1 管理ガイド』の第 1 章「ホストとクラスタの構成」で説明されています。これ以降の節では、コマンド行からホスト情報を検出するためのコマンドについて説明します。

マスターホストの名前の検出

マスターホストの場所は、いつでも現在のマスターホストとシャドウマスターホスト間で移動する可能性があります。したがって、マスターホストの場所はユーザーに対して透過的であるべきです。

テキストエディタで *sge-root*/ *cell*/common/act_qmaster ファイルを開きます。

現在のマスターホストの名前はファイルに含まれています。

実行ホストのリストの表示

クラスタ内で実行ホストとして構成されているホストのリストを表示するには、次のコマンドを使用します。

```
% qconf -sel
% qconf -se hostname
% qhost
```

qconf -sel コマンドは、実行ホストとして現在構成されているすべてのホストの名前のリストを表示します。qconf -se コマンドは、指定された実行ホストの詳細情報を表示します。qhost コマンドは、実行ホストのステータス情報と負荷情報を表示します。

qconf によって表示される情報の詳細は、host_conf(5) のマニュアルページを参照してください。出力とその他のオプションの詳細は、qhost(1) のマニュアルページを参照してください。

管理ホストのリストの表示

管理権を持つホストのリストを表示するには、次のコマンドを使用してください。

```
% qconf -sh
```

発行ホストのリストの表示

発行ホストのリストを表示するには、次のコマンドを使用してください。

```
% qconf -ss
```

要求可能な属性

ジョブを発行するときに、ジョブに対して要件プロファイルを指定できます。ユーザーは、実行を成功させるためにジョブに必要なホストやキューの属性または特徴を指定できます。Grid Engine ソフトウェアは、これらのジョブ要件をクラスタのホストとキュー構成に対応付けることによって、ジョブに適したホストを見つけます。

ジョブ要件の指定に使用できる属性は、次のいずれかと関連があります。

- ネットワーク共有ディスクで必要な空間などのクラスタ
- オペレーティングシステムアーキテクチャーなどの個別ホスト
- 使用できる CPU 時間などのキュー

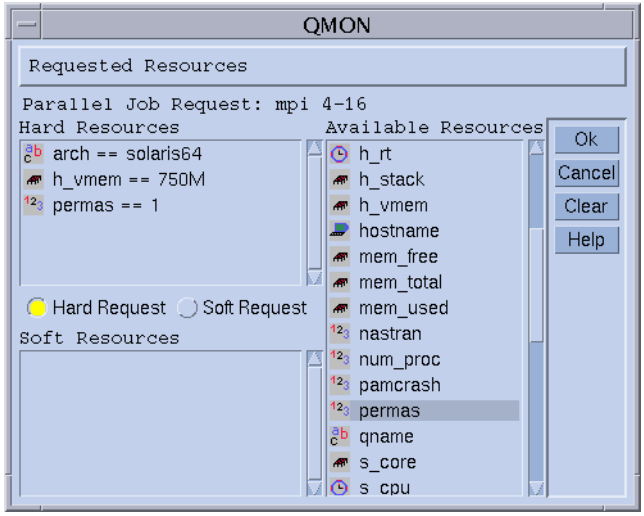
属性は、特定のホスト上だけで使用できるインストールソフトウェアなどのサイトポリシーから派生することもあります。

次のような属性を使用できます。

- キュープロパティーリスト – 38 ページの「キューとキュープロパティーの表示」を参照
- グローバル属性とホスト関連の属性のリスト – 『Sun N1 Grid Engine 6.1 管理ガイド』の「キュー、ホスト、およびグローバルクラスタへのリソース属性の割り当て」を参照
- 管理者定義の属性

管理者は便宜上よく、すべての使用可能な属性のサブセットを要求できるように定義します。

現在要求可能な属性は、次の図の「Requested Resources」ダイアログボックスに表示されています。



「Requested Resources」ダイアログボックスにアクセスするには、「QMON Submit Job」ダイアログボックスを使用してください。要求可能な属性は「Available Resources」の下に一覧表示されます。

要求可能な属性のリストの表示

構成済みのリソース属性のリストを表示するには、コマンド行から次のコマンドを入力してください。

```
% qconf -sc
```

Grid Engine システムコンプレックスには、すべてのリソース属性の定義が含まれています。リソース属性の詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の第 3 章「コンプレックスリソース属性の構成」を参照してください。complex(5) マニュアルページの複雑な書式の説明も参照してください。

qconf -sc コマンドの出力は、例 2–2 のようになります。

例 2–2 コンプレックス属性の表示

```
gimli% qconf -sc
#name               shortcut   type        relop requestable consumable default  urgency
#-------------------------------------------------------------------------------------------
arch                a          RESTRING    ==    YES         NO         NONE     0
calendar            c          STRING      ==    YES         NO         NONE     0
cpu                 cpu        DOUBLE      >=    YES         NO         0        0
h_core              h_core     MEMORY      <=    YES         NO         0        0
h_cpu               h_cpu      TIME        <=    YES         NO         0:0:0    0
h_data              h_data     MEMORY      <=    YES         NO         0        0
h_fsize             h_fsize    MEMORY      <=    YES         NO         0        0
h_rss               h_rss      MEMORY      <=    YES         NO         0        0
h_rt                h_rt       TIME        <=    YES         NO         0:0:0    0
h_stack             h_stack    MEMORY      <=    YES         NO         0        0
h_vmem              h_vmem     MEMORY      <=    YES         NO         0        0
hostname            h          HOST        ==    YES         NO         NONE     0
load_avg            la         DOUBLE      >=    NO          NO         0        0
load_long           ll         DOUBLE      >=    NO          NO         0        0
load_medium         lm         DOUBLE      >=    NO          NO         0        0
load_short          ls         DOUBLE      >=    NO          NO         0        0
mem_free            mf         MEMORY      <=    YES         NO         0        0
mem_total           mt         MEMORY      <=    YES         NO         0        0
mem_used            mu         MEMORY      >=    YES         NO         0        0
min_cpu_interval    mci        TIME        <=    NO          NO         0:0:0    0
np_load_avg         nla        DOUBLE      >=    NO          NO         0        0
np_load_long        nll        DOUBLE      >=    NO          NO         0        0
np_load_medium      nlm        DOUBLE      >=    NO          NO         0        0
np_load_short       nls        DOUBLE      >=    NO          NO         0        0
num_proc            p          INT         ==    YES         NO         0        0
qname               q          STRING      ==    YES         NO         NONE     0
rerun               re         BOOL        ==    NO          NO         0        0
s_core              s_core     MEMORY      <=    YES         NO         0        0
s_cpu               s_cpu      TIME        <=    YES         NO         0:0:0    0
```

例 2–2 コンプレックス属性の表示 *(続き)*

```
s_data              s_data     MEMORY      <=    YES         NO          0        0
s_fsize             s_fsize    MEMORY      <=    YES         NO          0        0
s_rss               s_rss      MEMORY      <=    YES         NO          0        0
s_rt                s_rt       TIME        <=    YES         NO          0:0:0    0
s_stack             s_stack    MEMORY      <=    YES         NO          0        0
s_vmem              s_vmem     MEMORY      <=    YES         NO          0        0
seq_no              seq        INT         ==    NO          NO          0        0
slots               s          INT         <=    YES         YES         1        1000
swap_free           sf         MEMORY      <=    YES         NO          0        0
swap_rate           sr         MEMORY      >=    YES         NO          0        0
swap_rsvd           srsv       MEMORY      >=    YES         NO          0        0
swap_total          st         MEMORY      <=    YES         NO          0        0
swap_used           su         MEMORY      >=    YES         NO          0        0
tmpdir              tmp        STRING      ==    NO          NO          NONE     0
virtual_free        vf         MEMORY      <=    YES         NO          0        0
virtual_total       vt         MEMORY      <=    YES         NO          0        0
virtual_used        vu         MEMORY      >=    YES         NO          0        0
# >#< starts a comment but comments are not saved across edits --------
```

`name` の列は、`qconf -sq` コマンドによって表示される最初の列と同じです。`shortcut` の列には、最初の列のフルネームの省略形が入ります。省略形は管理者によって定義されます。`qsub` コマンドの要求オプションでは、フルネームまたはショートカットのどちらでも指定できます。

`requestable` の列には、リソース属性が `qsub` コマンドで使用できるかどうかが示されています。管理者は、たとえば、クラスタのユーザーが特定のマシンまたは自分のジョブのキューを直接要求できないようにすることもできます。管理者は、エントリ `qname`、`hostname`、またはこの両方を要求不可にすることができます。キューまたはホストを要求不可能にすると、実行可能なユーザー要求を通常複数のキューで満足させることができ、Grid Engine システムの負荷均衡機能が実行されます。

`relop` の列には、キューまたはホストがユーザー要求を満たせるかどうかを計算するための関係演算子が定義されています。次のような比較が行われます。

```
User_Request     relop     Queue/Host/... -Property
```

比較結果が偽の場合、ユーザーのジョブをそのキューまたはホストで実行することはできません。たとえば、キュー `q1` を 100 秒のソフト CPU 時間制限で構成したとします。キュー `q2` は、1000 秒のソフト CPU 時間制限があるように構成します。ユーザープロセス制限の説明は、 `queue_conf`(5) および `setrlimit`(2) のマニュアルページを参照してください。

`consumable` と `default` の列は、管理者の消費可能リソースの宣言方法に影響します。『Sun N1 Grid Engine 6.1 管理ガイド』の「消費可能リソース」を参照してください。

ユーザーは、ほかの属性とまったく同じように消費可能リソースを要求します。ただし、Grid Engine システムがリソースの内部帳簿機能をつける点が異なります。

ユーザーが次の要求を発行したとします。

```
% qsub -l s_cpu=0:5:0 nastran.sh
```

`s_cpu=0:5:0` 要求は、5 分以上のソフト制限 CPU 時間を許可するキューを求めます。したがって、5 分以上のソフト CPU 実行時間制限を提供するキューだけが適切に設定され、ジョブを実行できます。構文の詳細は、`qsub`(1) のマニュアルページを参照してください。

注 – Grid Engine ソフトウェアは、複数のキューまたはホストがジョブを実行できる場合だけ、スケジューリングプロセスの作業負荷情報を考慮します。

◆ ◆ ◆ 第 3 章

ジョブの発行

この章では、ジョブの発行の内容説明と、処理を行うためにジョブを発行する手順について説明します。まず、単純なジョブの実行例から始めて、より複雑なジョブの実行手順に進みます。

この章には、次の作業の実行方法が記載されています。

単純なジョブの発行

この節の情報と手順を読んで、ジョブの発行を行うための基本的な手順に慣れてください。

注 – 非特権ユーザーアカウントで N1 Grid Engine 6.1 ソフトウェアをインストールした場合は、ジョブを実行できるユーザーとしてログインする必要があります。詳細は、『Sun N1 Grid Engine 6.1 インストールガイド』の「インストールアカウント」を参照してください。

▼ コマンド行からの単純なジョブの発行方法

すべての Grid Engine システムコマンドを実行する前に、まず実行可能な検索パスやその他の環境条件を適切に設定する必要があります。

1 コマンド行から次のコマンドのいずれかを入力します。

- コマンドインタプリタとして `csh` または `tcsh` を使用している場合は、次のように入力します。

  ```
  % source sge-root/cell/common/settings.csh
  ```

  sge-root は、Grid Engine システムのルートディレクトリの場所を指定します。このディレクトリは、インストール手順の最初に指定されています。
- コマンドインタプリタとして `sh`、`ksh`、または `bash` を使用している場合は、次のように入力します。

  ```
  # . sge-root/cell/common/settings.sh
  ```

注 – これらのコマンドは、`.login`、`.cshrc`、または `.profile`ファイルの中で適当なものに追加できます。これらのコマンドを設定すると、今後開始するすべての対話セッションに適切な設定を保証できます。

---

2 次のコマンドを入力して、クラスタに単純なジョブスクリプトを発行します。

```
% qsub simple.sh
```

コマンドは、`simple.sh` がスクリプトファイルの名前で、このファイルがユーザーの現在の作業ディレクトリにあるとみなします。

次のジョブを */sge-root*`/examples/jobs/simple.sh` ファイル内で見つけることができます。

```
#!/bin/sh
#
#
# (c) 2004 Sun Microsystems, Inc. Use is subject to license terms.

#  SGE バッチスクリプトの簡単な例

# Bourne シェルに次のジョブを要求する。
#$ -S /bin/sh

#
# 日時を印刷する
date
# 20 秒スリープする
```

```
sleep 20
# 日時を再び印刷する
date
```

ジョブが正しく発行されると、qsub コマンドは次の例のようなメッセージで応答します。

```
your job 1 (“simple.sh”) has been submitted
```

3 次のコマンドを入力して、ジョブのステータス情報を検索します。

```
% qstat
```

Grid Engine システムが現在認識しているすべてのジョブの情報が示されたステータスレポートを受け取ることができるはずです。ステータスレポートは、ジョブごとに次の項目を一覧表示します。

- 発行の確認に含まれる一意の番号であるジョブ ID
- ジョブスクリプトの名前
- ジョブの所有者
- 実行中を表す r などの状態インジケータ
- 発行時または開始時
- ジョブが実行されるキューの名前

qstat を実行しても何も出力されなかった場合、システムが認識しているジョブはありません。たとえば、ジョブはすでに終了している可能性もあります。

stdout および stderr リダイレクトファイルをチェックすると、終了ジョブの出力を制御できます。デフォルトでは、これらのファイルはジョブを実行したホスト上のジョブ所有者のホームディレクトリに生成されます。ファイルの名前は、stdout ファイルの場合は .o、stderr ファイルの場合は .e の拡張子が付くジョブスクリプトのファイル名のあとに一意のジョブ ID が続く構成となります。ジョブの stdout および stderr ファイルは、それぞれ simple.sh.o1 と simple.sh.e1 という名前で見つけることができます。これらの名前は、そのジョブが、新たにインストールされた Grid Engine システムで初めて実行されるジョブの場合に使用されます。

▼ QMON により単純なジョブを発行する

グラフィカルユーザーインタフェースの QMON を使用すると、ジョブの発行と制御および Grid Engine システムの概要の取得をより便利に行うことができます。ほかにも機能はありますが、QMON は、ジョブの発行と監視を行うためのジョブ発行ダイアログボックスと「Job Control」ダイアログボックスも提供しています。

1 次のコマンドを入力して QMON GUI を起動します。

```
% qmon
```

起動中にメッセージウィンドウが表示され、「QMON Main Control」ウィンドウが表示されます。

図 3–1 「QMON Main Control」ウィンドウ

2 「**Job Control**」ボタンをクリックしたあと、「**Submit Jobs**」ボタンをクリックします。

ヒント – 「Job Control」などのボタン名は、ボタン上にマウスポインタを置くと表示されます。

「Submit Job」ダイアログボックスと「Job Control」ダイアログボックスが、次の図のように表示されます。

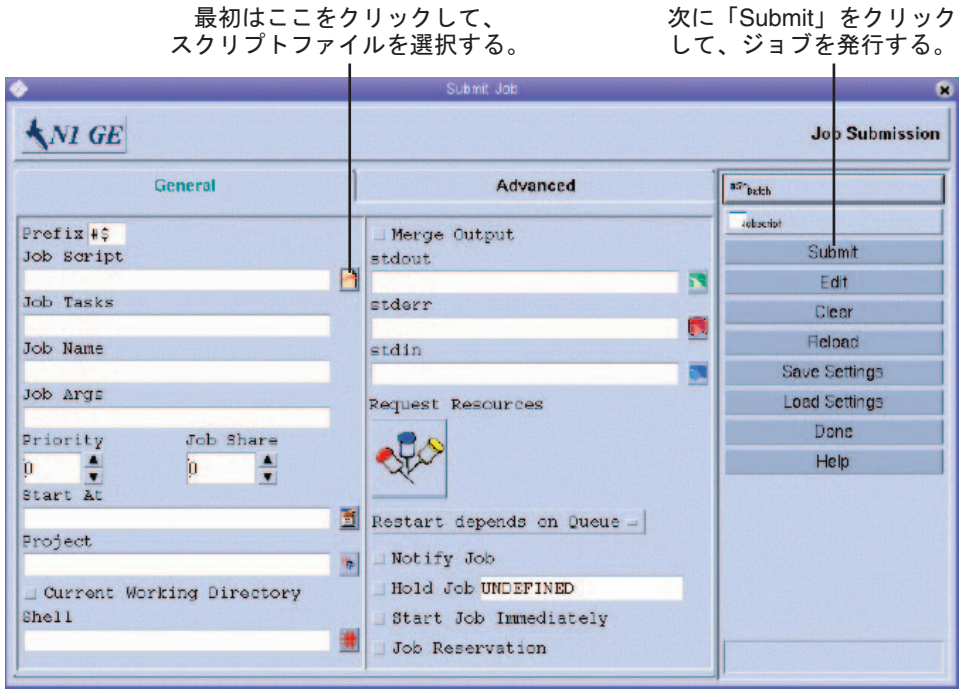


図 3–2 「Submit Job」ダイアログボックス

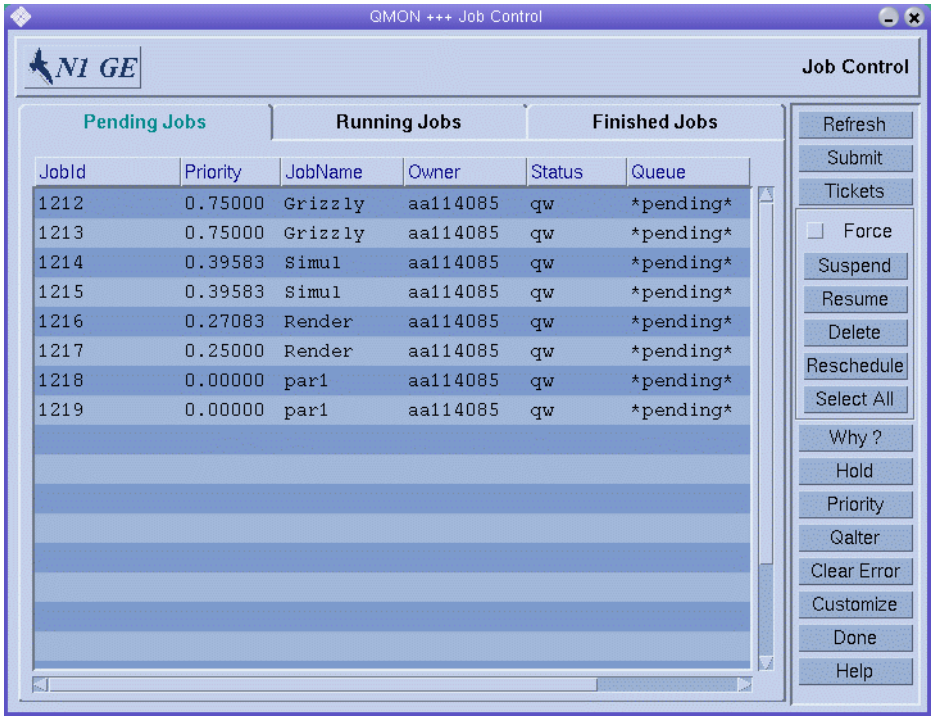


図 3–3 「Job Control」ダイアログボックス

3 「**Submit Job**」ダイアログボックスで「**Job Script**」フィールドの右にあるアイコンをクリックします。

「Select a File」ダイアログボックスが表示されます。

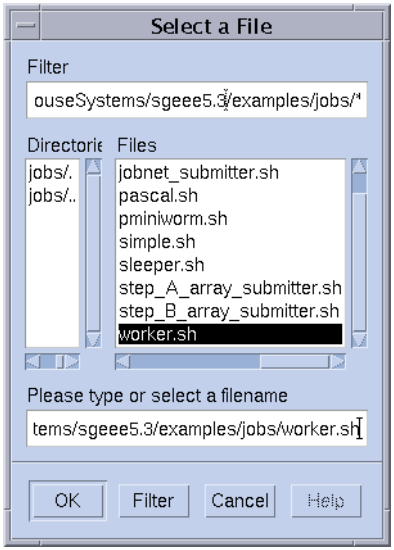


図 3–4 「Select a File」ダイアログボックス

4 スクリプトファイルを選択します。
たとえば、コマンド行の例で使用した `simple.sh` ファイルを選択します。

5 「**OK**」をクリックして、「**Select a File**」ダイアログボックスを閉じます。

6 「**Submit Job**」ダイアログボックスで「**Submit**」をクリックします。
数秒後には、「Job Control」ダイアログボックスでジョブを監視できるはずです。最初に「Pending Jobs」タブで自分のジョブを確認します。ジョブの実行が開始されると、ジョブはすぐ「Running Jobs」タブに移ります。

バッチジョブの発行

次の節では、Grid Engine システムでより複雑なジョブを発行する方法について説明します。

シェルスクリプトについて

バッチジョブとも呼ばれるシェルスクリプトは、ファイル内でアセンブルされる一連のコマンド行命令です。スクリプトファイルは、 `chmod` コマンドによって実行可能になります。スクリプトを呼び出すと、コマンドインタプリタが起動されます。各命令は、スクリプトを実行するユーザーが手で入力しているかのように解釈されます。一般的なコマンドインタプリタには、`csh`、`tcsh`、 `sh`、または `ksh` があります。

シェルスクリプト内からは、任意のコマンド、アプリケーション、およびその他のシェルスクリプトを呼び出すことができます。

コマンドインタプリタは、ログインシェルとして呼び出すことができます。このためには、ジョブを実行している特定のホストおよびキューに対して有効な Grid Engine システムの構成の `login_shells` リストに該当コマンドインタプリタの名前を含める必要があります。

注 – Grid Engine システムの構成は、クラスタ内のさまざまなホストやキューによって異なる場合があります。有効な構成は、`qconf` コマンドの `-sconf` オプションと `-sq` オプションで表示できます。詳細は、`qconf`(1) のマニュアルページを参照してください。

コマンドインタプリタをログインシェルとして呼び出すと、ジョブの環境はスクリプトにログインして実行した場合と同じになります。たとえば、`csh` を使用すると、`/etc/login` などのシステムのデフォルト起動リソースファイル以外に `.login` と `.cshrc` が実行されます。一方、`csh` を `login-shell` として呼び出さなかった場合は、`.cshrc` だけが実行されます。`login-shell` として呼び出す場合と呼び出さない場合の違いについては、お使いのコマンドインタプリタのマニュアルページを参照してください。

シェルスクリプトの例

例 3–1 に単純なシェルスクリプトを示します。このスクリプトは、まずアプリケーション `flow` を Fortran77 ソースからコンパイルしたあと、このアプリケーションを実行します。

例 3–1　単純なシェルスクリプト

```
#!/bin/csh
# N1 Grid Engine 6.1 にある FORTRAN プログラムのサンプルを
#  コンパイルし、実行するスクリプトファイルの例
cd TEST
# プログラム "flow.f" をコンパイルし、実行可能な "flow" を
# ファイル名として命名する必要がある。
f77 flow.f -o flow
```

ローカルシステムのユーザーズガイドには、シェルスクリプトの作成とカスタマイズに関する詳細情報が記載されています。`sh`、`ksh`、`csh` または `tcsh` のマニュアルページも参照してください。以降の節では、Grid Engine システム用のバッチスクリプトの作成時に、特に考慮すべき事柄を重点的に説明します。

一般的に、コマンドプロンプトから手動で実行できるすべてのシェルスクリプトは、Grid Engine システムに発行できます。 このシェルスクリプトは、端末接続や対話形式のユーザー介入を必要としてはなりません。ただし、自動的にリダイレクトされる標準エラーと標準出力デバイスは例外です。したがって、例 3–1 では Grid Engine システムへの発行準備が整っており、スクリプトは指定の処理を行います。

通常のシェルスクリプトの拡張

通常のシェルスクリプトを拡張すると、Grid Engine システムの制御下で実行されるスクリプトの動作に影響が出る場合があります。次の節では、これらの拡張機能について説明します。

コマンドインタプリタの選択方法

図 3–5 に示すように、発行時には、ジョブスクリプトファイルの処理に使用するコマンドインタプリタを指定できます。何も指定しなかった場合、コマンドインタプリタの選択方法は構成変数 `shell_start_mode` によって決められます。

- `shell_start_mode` が `unix_behavior` に設定されると、スクリプトファイルの 1 行目でコマンドインタプリタが指定されます。スクリプトファイルの 1 行目は、`#!` で始めてください。1 行目が `#!` で始まっていない場合、デフォルトでは Bourne シェル `sh` が使用されます。
- `shell_start_mode` のその他のすべての設定では、デフォルトのコマンドインタプリタは、ジョブが開始されるキューの `shell` パラメータによって決まります。38 ページの「キューとキュープロパティーの表示」と`queue_conf`(5) のマニュアルページを参照してください。

出力のリダイレクト

バッチジョブには端末接続がないため、標準出力と標準エラー出力はファイルにリダイレクトしなければなりません。Grid Engine システムでは、出力のリダイレクト先となるファイルの場所を定義できます。出力ファイルが指定されていない場合は、デフォルトが使用されます。

ファイルの標準的な場所は、ジョブが実行される現在の作業ディレクトリです。デフォルトの標準出力ファイル名は *job-name*`.o` *job-id* で、デフォルトの標準エラーの出力先は *job-name*>.e*job-id* にリダイレクトされます。*job-name* はスクリプトファイル名から作成するか、ユーザーが定義できます。たとえば、`submit`(1) のマニュアルページの `-N` オプションを参照してください。*job-id* は、Grid Engine システムによってジョブに割り当てられる一意の識別子です。

配列ジョブのタスクの場合 、タスクの識別子はドットで区切られて、これらのファイル名に追加されます。リダイレクト後の標準出力先パスは、*job-name*.o*job-id*.*task-id*> と *job-name*.e *job-id*.*task-id* になります。詳細は、72 ページの「配列ジョブの発行」を参照してください。

標準の場所が適切でない場合は、 図 3–6 のように、QMON で出力先を指定できます。また、qsub コマンドの -e や -o オプションを使用して、出力先を指定することもできます。標準出力と標準エラー出力は、1 つのファイルにマージすることもできます。リダイレクトは、実行ホストごとに指定できます。その場合、ジョブが実行されるホストによって、出力するリダイレクトファイルの場所は異なります。一意のカスタマイズリダイレクトファイルパスを作成するには、qsub の -e および -o オプションと一緒に使用できるダミー環境変数を使用します。次にこれらの変数の一覧を示します。

- HOME – 実行マシン上のホームディレクトリ
- USER – ジョブ所有者のユーザー ID
- JOB_ID – 現在のジョブ ID
- JOB_NAME – 現在のジョブ名。 -N オプションを参照してください。
- HOSTNAME – 実行ホストの名前
- TASK_ID – 配列ジョブタスクのインデックス番号

ジョブを実行すると、これらの変数に実際の値が入り、その値でリダイレクトパスが作成されます。

詳細は、qsub(1) のマニュアルページを参照してください。

有効なコメント

シェルスクリプトでは、ハッシュ記号 (#) が最初にある行はコメントとして扱われます。 ただし、Grid Engine システムは特別なコメント行を認識し、それらの行を特別な方法で使用します。そのような特別なコメントスクリプト行は、qsub コマンドのコマンド行引数リストの一部として扱われます。これらの特別なコメント行内で指定される qsub オプションも「QMON Submit Job」ダイアログボックスで解釈されます。スクリプトファイルを選択すると、対応するパラメータがプリセットされます。

デフォルトでは、特別なコメント行は #$ という接頭辞で識別されます。接頭辞は、qsub -C コマンドで再定義できます。

このような特別なコメントの使用は、発行引数のスクリプト組み込みと呼ばれます。次に、スクリプト組み込みコマンド行オプションを利用したスクリプトファイルの例を示します。

例 3–2 スクリプト組み込みコマンド行オプションの使用

```
#!/bin/csh

# Grid Engine のデフォルトのシェルでなければ
# csh を強制

#$ -S /bin/csh
```

例 3–2 スクリプト組み込みコマンド行オプションの使用 *(続き)*

```
# これは N1 Grid Engine 6.1 の元でサンプルの FORTRAN プログラムを
# コンパイルし、実行するスクリプトファイルの例です。
# Grid Engine に、ジョブの開始時と終了時に
# メールを送信させるように
# します。

#$ -M EmailAddress
#$ -m b,e

# 標準出力と標準エラーのためのファイルを
# 命名します。

#$ -o flow.out -j y

# ファイルがあるディレクトリに変更します。

cd TEST

# プログラム "flow.f" をコンパイルし、
# 実行可能な "flow" と命名する必要があります。

f77 flow.f -o flow

# コンパイルを完了すると、そのプログラムを実行できます。

flow
```

環境変数

ジョブが実行されると、いくつかの変数がジョブの 環境にプリセットされます。

- ARC – ジョブが実行されているノードのアーキテクチャー名。この名前は sge_execd バイナリにコンパイルされます。
- SGE_ROOT – Grid Engine システムのルートディレクトリ。起動する前に sge_execd で設定されたディレクトリ、またはデフォルトの /usr/SGE ディレクトリです。
- SGE_BINARY_PATH – Grid Engine システムのバイナリがインストールされているディレクトリ。
- SGE_CELL – ジョブが実行されるセル。
- SGE_JOB_SPOOL_DIR – ジョブの実行中にジョブ関連データを保存するために sge_shepherd が使用するディレクトリ。
- SGE_O_HOME – ジョブの発行元になったホスト上にある、ジョブ所有者のホームディレクトリへのパス。
- SGE_O_HOST – ジョブの発行元になったホスト。

- SGE_O_LOGNAME – ジョブの発行元になったホストでのジョブ所有者のログイン名。
- SGE_O_MAIL – ジョブ発行コマンドのコンテキスト内での MAIL 環境変数の内容。
- SGE_O_PATH – ジョブ発行コマンドのコンテキスト内での PATH 環境変数の内容。
- SGE_O_SHELL – ジョブ発行コマンドのコンテキスト内での SHELL 環境変数の内容。
- SGE_O_TZ – ジョブ発行コマンドのコンテキスト内での TZ 環境変数の内容。
- SGE_O_WORKDIR – ジョブ発行コマンドの作業ディレクトリ。
- SGE_CKPT_ENV – チェックポイント設定ジョブが実行されるチェックポイント設定環境。チェックポイント設定環境は、qsub -ckpt コマンドで選択されます。
- SGE_CKPT_DIR – チェックポイント設定インタフェースの ckpt_dir パス。チェックポイント設定ジョブに対してのみ設定されます。詳細は、checkpoint(5) のマニュアルページを参照してください。
- SGE_STDERR_PATH – ジョブの標準エラーストリームを出力するファイルのパス名。このファイルは、通常プロローグ、エピローグ、並列環境開始スクリプトおよび終了スクリプト、またはチェックポイント設定スクリプトからのエラーメッセージ出力を拡張するために使用されます。
- SGE_STDOUT_PATH – ジョブの標準出力ストリームを出力するファイルのパス名。このファイルは、通常プロローグ、エピローグ、並列環境開始スクリプトおよび終了スクリプト、またはチェックポイント設定スクリプトからのメッセージ出力を拡張するために使用されます。
- SGE_TASK_ID – このタスクで表される配列ジョブ内のタスク識別子。
- ENVIRONMENT – 常に BATCH に設定されます。この変数は、スクリプトがバッチモードで実行されることを示します。
- HOME – passwd ファイルからのユーザーのホームディレクトリのパス。
- HOSTNAME – ジョブを実行しているノードのホスト名。
- JOB_ID – ジョブの発行時に sge_qmaster デーモンによって割り当てられる一意の識別子。ジョブ ID は、1 から 9,999,999 の 10 進法の整数です。
- JOB_NAME – qsub コマンドで指定されたファイル名、ピリオド、およびジョブ ID の数字で構成されるジョブ名。このデフォルトは、qsub -N で無効にできます。
- LOGNAME – passwd ファイルから得られるユーザーのログイン名。
- NHOSTS – 並列ジョブで使用されるホストの数。
- NQUEUES – ジョブに割り当てられたキューの数。直列ジョブでは、この数は常に 1 です。
- NSLOTS – 並列ジョブで使用されるキュースロットの数。
- PATH – /usr/local/bin:/usr/ucb:/bin:/usr/bin のデフォルトシェル検索パス。
- PE – ジョブが実行される並列環境。この変数は、並列ジョブでのみ使用されます。

- PE_HOSTFILE – Grid Engine システムによって並列ジョブに割り当てられた仮想並列マシンの定義を含むファイルのパス。 この変数は、並列ジョブでのみ使用されます。このファイルの形式については、「sge_pe」の $pe_hostfile パラメータの説明を参照してください。
- QUEUE – ジョブが実行されているキューの名前。
- REQUEST – ジョブの要求名。この名前は、ジョブスクリプトファイル名か、qsub -N コマンドによって明示的にジョブに割り当てられた名前です。
- RESTARTED – チェックポイント設定ジョブが再開されたかどうかを示します。値が 1 に設定されている場合、ジョブは 1 回以上割り込まれています。よって、ジョブは再開されていることになります。
- SHELL – passwd ファイルから得られるユーザーのログインシェル。

注 – SHELL は、ジョブに使用されているシェルとは限りません。

- TMPDIR – ジョブの一時作業ディレクトリへの絶対パス。
- TMP – TMPDIR と同じです。この変数は、NQS との互換性を確保するために提供されています。
- TZ – 設定されている場合、sge_execd からインポートされるタイムゾーン変数。
- USER – passwd ファイルから得られるユーザーのログイン名。

拡張ジョブと高度なジョブの発行

拡張ジョブと高度なジョブは、より複雑なジョブ発行形式です。これらのジョブを発行しようとする前に、プロセスに関する重要な内容説明を理解する必要があります。次の節では、これらのジョブプロセスについて説明します。

QMON による拡張ジョブの発行

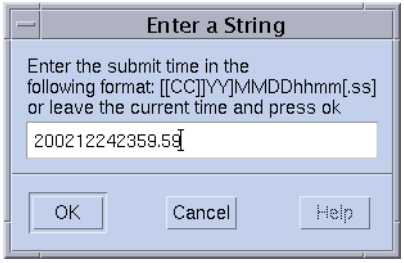
「Submit Job」ダイアログボックスの「General」タブでは、拡張ジョブに対して次のパラメータを設定できます。「General」タブは、 図 3–2 に示されています。

- Prefix – スクリプト組み込み発行オプションで使用する接頭辞。詳細は、56 ページの「有効なコメント」を参照してください。
- Job Script – 使用されるジョブスクリプト。ファイル選択ボックスを開くには、「Job Script」フィールドの右のアイコンをクリックします。ファイル選択ボックスは、図 3–4 に示されています。
- Job Tasks – 配列ジョブを発行するためのタスク ID の範囲。詳細は、72 ページの「配列ジョブの発行」を参照してください。

- Job Name – ジョブの名前。デフォルトは、ジョブスクリプトの選択後に設定されます。
- Job Args – ジョブスクリプトの引数。
- Priority – ジョブの初期状態での優先順位を設定するカウントボックス。この優先順位は、1 人のユーザーのジョブ間でのジョブの順番です。優先順位によって、同時に 1 人のユーザーの複数のジョブがシステム内に存在する場合に、どのようにしてそのユーザーのジョブから選択を行うかがスケジューラに伝えられます。

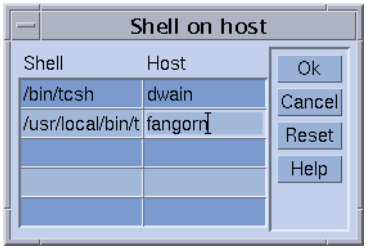
注 – ユーザーが自分のジョブの優先順位を設定できるように、管理者はスケジューラ構成の `weight_priority` パラメータで優先順位を有効にしなければなりません。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の第 5 章「ポリシーとスケジューラの管理」を参照してください。

- Job Share – その他のジョブに関連するジョブのチケット共有の定義。ジョブの共有は、共有ツリーポリシーと機能ポリシーにだけ影響します。
- Start At – ジョブが実行可能とみなされる時点。正しい書式の時間を入力するダイアログボックスを開くには、「Start At」フィールドの右のアイコンをクリックします。

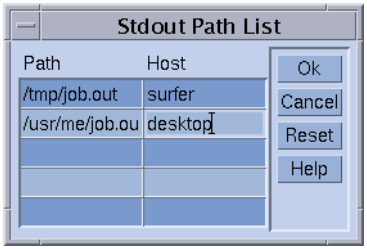


- Project – ジョブを下位に持つプロジェクト。「Project」フィールドの右にあるアイコンをクリックして、使用可能なプロジェクトの中からプロジェクトを選択します。

- Current Working Directory – 現在の作業ディレクトリでジョブを実行するかどうかを示すフラグ。このフラグは、発行ホストと潜在実行ホスト間の同一のディレクトリ階層に対してのみ使用してください。
- Shell – ジョブスクリプトの実行に使用されるコマンドインタプリタ。詳細は、55 ページの「コマンドインタプリタの選択方法」を参照してください。ジョブのコマンドインタプリタ指定を入力するダイアログボックスを開くには、「Shell」フィールドの右のアイコンをクリックします。



- Merge Output – ジョブの標準出力と標準エラー出力を標準出力ストリームにマージするかどうかを示すフラグ。
- stdout – 使用される標準出力先。詳細は、55 ページの「出力のリダイレクト」を参照してください。指定がない場合は、デフォルトが使用されます。代わりの出力先を入力するダイアログボックスを開くには、「`stdout`」フィールドの右のアイコンをクリックします。



- stderr – 使用される標準エラー出力。標準出力と同じようなものです。
- stdin – 使用される標準入力ファイル。標準出力先と同じようなものです。
- Request Resources – ジョブのリソース要件を定義するには、このボタンをクリックします。ジョブにリソースが必要な場合、ボタンの色が変わります。
- Restart depends on Queue – システムクラッシュなどのイベントで中止されたあと、ジョブを再開できるかどうかを定義するには、このボタンをクリックします。このボタンはまた、再開動作をキューによって決めるか、ジョブの要求によって決めるかも制御します。
- Notify Job – ジョブが一時停止またはキャンセルされる場合に、`SIGUSR1` または `SIGUSR2` 信号によってジョブを通知するかどうかを示すフラグ。

- Hold Job – ユーザーの待機またはジョブ依存関係がジョブに割り当てられることを示すフラグ。ジョブになんらかの種類の待機が割り当てられている間は、ジョブを実行することはできません。詳細は、89 ページの「ジョブの監視と制御」を参照してください。「Hold Job」フィールドでは、特定の範囲の配列ジョブのタスクだけに待機を限定することができます。配列ジョブについては、72 ページの「配列ジョブの発行」を参照してください。
- Start Job Immediately – 可能であればすぐにジョブを強制的に開始し、不可能な場合は拒否するフラグ。このフラグが選択されている場合、ジョブはキューには入れられません。
- Job Reservation – リソースをこのジョブ用に予約する必要があることを示すフラグ。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「リソース予約およびバックフィリング」を参照してください。

「Submit Job」ダイアログボックスの右にあるボタンを使用すると、次のようなさまざまな操作を開始できます。

- **Submit** – 現在指定されているジョブを発行します。
- **Edit** – `vi`、または `EDITOR` 環境変数によって定義されたエディタを使用して、選択したスクリプトファイルを X 端末で編集します。
- **Clear** – 指定されたリソース要求など、「Submit Job」ダイアログボックス内のすべての設定を消去します。
- **Reload** – 指定したスクリプトファイルを再読込みし、スクリプト組み込みオプションの構文解析やデフォルト設定の構文解析を行い、これらの設定への中間的な手動による変更を廃棄します。詳細は、56 ページの「有効なコメント」と 68 ページの「デフォルト要求ファイル」を参照してください。この操作は、「Clear」操作の後に前のスクリプトファイルを指定するのと同じです。このオプションが有効なのは、スクリプトファイルがすでに選択されている場合だけです。
- **Save Settings** – 現在の設定をファイルに保存します。ファイル選択ボックスを使用して、ファイルを選択します。保存されたファイルは、あとで読み込みをしたり、デフォルト要求として使用したりできます。詳細は、**Load Settings** と 68 ページの「デフォルト要求ファイル」を参照してください。
- **Load Settings** – 以前「Save Settings」ボタンで保存した設定を読み込みます。読み込んだ設定は、現在の設定を上書きします。**Save Settings** を参照してください。
- **Done** – 「Submit Job」ダイアログボックスを閉じます。

拡張ジョブの例

図 3–5 に、大部分のパラメータが設定された「Submit Job」ダイアログボックスを示します。

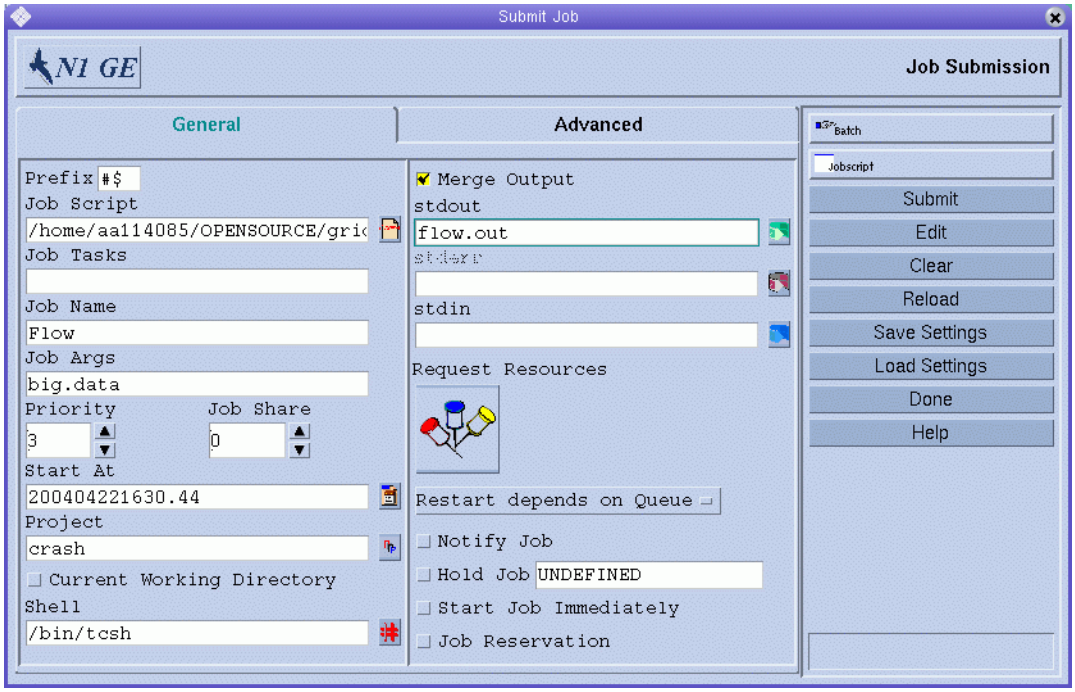


図3–5　拡張ジョブの発行例

この例で設定されているジョブのパラメータは、次のとおりです。

- ジョブは、スクリプトファイル `flow.sh` を持っています。このファイルは、`QMON` の作業ディレクトリになければなりません。
- ジョブの名前は `Flow` です。
- スクリプトファイルは、1 つの引数 `big.data` を取ります。
- ジョブは、優先順位 3 で開始されます。
- ジョブは、2004 年 4 月 22 日の午前 4 時 30 分 44 秒より前には実行できません。
- プロジェクト定義は、ジョブがプロジェクト `crash` の下位にあることを示しています。
- ジョブは、発行作業ディレクトリで実行されます。
- ジョブは、`tcsh` コマンドインタプリタを使用します。
- 標準出力と標準エラー出力は `flow.out` ファイルにマージされます。このファイルは、現在の作業ディレクトリに作成されます。

コマンド行からの拡張ジョブの発行

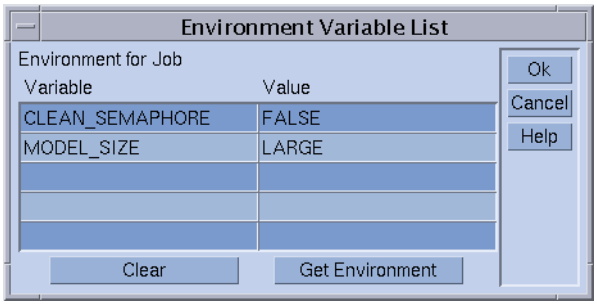
コマンド行から 図 3–5 に示されている拡張ジョブ要求を発行するには、次のコマンドを入力します。

```
% qsub -N Flow -p -111 -P devel -a 200404221630.44 -cwd \
    -S /bin/tcsh -o flow.out -j y flow.sh big.data
```

QMON による高度なジョブの発行

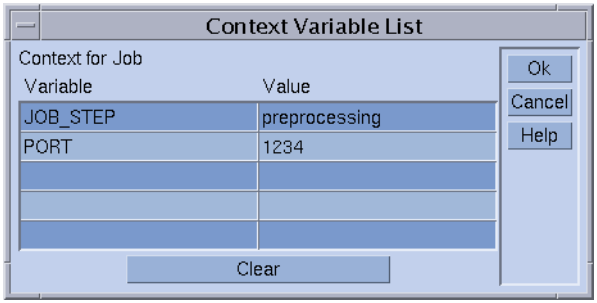
「Submit Job」ダイアログボックスの「Advanced」タブでは、次の追加パラメータを定義できます。

- Parallel Environment – 使用される並列環境インタフェース
- Environment – ジョブの実行前にジョブに対して設定される一連の環境変数。エクスポートする環境変数を定義するためのダイアログボックスを開くには、「Environment」フィールドの右のアイコンをクリックします。

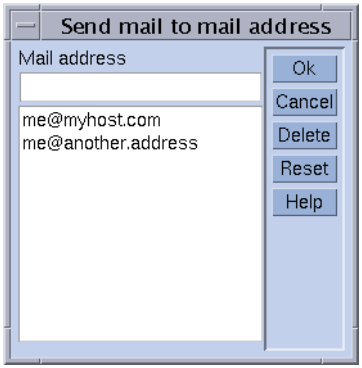


  環境変数は QMON の実行環境から得ることができます。また、独自の環境変数を定義することもできます。

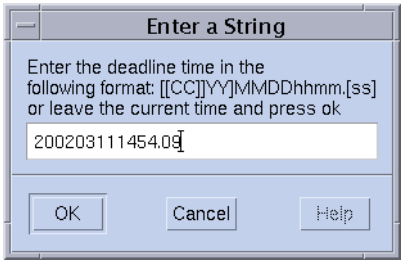
- Context – ジョブ関連情報の保存または転送に使用できる名前と値のペアのリスト。この情報には、クラスタ内のどこからでもアクセスできます。コンテキスト変数は、コマンド行から qsub、 qrsh、qsh、qlogin、および qalter の -ac、-dc、および -sc オプションを使用して変更できます。コンテキスト変数の検索は、 qstat -j コマンドで行うことができます。



- Checkpoint Object – ジョブのチェックポイントの設定が必要で適切な場合に使用されるチェックポイントの設定環境。詳細は、112 ページの「ジョブチェックポイント設定の使用」を参照してください。
- Account – ジョブと関連付けられるアカウント文字列。アカウント文字列は、ジョブのために維持されるアカウンティングレコードに追加されます。アカウンティングレコードは、あとでアカウンティング分析に使用できます。
- Verify Mode – ジョブの一貫性チェックモードを決定する確認フラグ。ジョブ要求の一貫性をチェックするために、Grid Engine システムは空で読み込み解除されたクラスタを想定します。システムは、ジョブを実行できるキューを 1 つ以上見つけようとします。次のようなチェックモードを使用できます。
  - **Skip** – 一貫性はチェックされません。
  - **Warning** – 不一致は報告されますが、ジョブは受け付けられます。警告モードは、ジョブの発行後にクラスタ構成を変更しなければならない場合に適しています
  - **Error** – 不一致が報告されます。不一致がある場合、ジョブは拒否されます。
  - **Just verify** – ジョブは発行されません。ジョブがクラスタ内の各ホストとキューに適しているかどうかについての詳しいレポートが生成されます。
- Mail – ユーザーに電子メールで通知するイベント。イベントの開始、終了、中止、および一時停止が、現在ジョブに対して定義されています。
- Mail To – これらの通知を送信する電子メールアドレスのリスト。メーリングリストを定義するダイアログボックスを開くには、「Mail To」フィールドの右のアイコンをクリックします。



- Hard Queue List、Soft Queue List – ジョブの実行の必須選択項目として必要なキュー名のリスト。「Hard Queue List」と「Soft Queue List」は、対応するリソース要件と同様に処理されます。
- Master Queue List – 並列ジョブのマスターキューとして実行可能なキュー名のリスト。 並列ジョブはマスターキューで開始されます。ジョブが並列タスクを生成するその他のすべてのキューは、スレーブキューと呼ばれます。
- Job Dependencies – 発行したジョブを開始する前に終了しなければならないジョブの ID のリスト。新たに作成されるジョブは、ジョブの完了によって異なります。
- Deadline – 期限付きジョブの開始期限。開始期限は、期限付きジョブが指定期限前に終了されるために最高の優先順位を持つ時点を定義します。開始期限を決定するには、最高の優先順位での期限付きジョブの実行時間を指定の期限から引いて差を求めます。期限を設定するためのダイアログボックスを開くには、「Deadline」フィールドの右のアイコンをクリックします。



注 – 一部のユーザーは、期限付きジョブを発行できません。自分が期限付きジョブを発行できるかどうかは、システム管理者にお問い合わせください。期限付きジョブに与えられる最高の優先順位については、クラスタ管理者にお問い合わせください。

高度なジョブの例

図 3–6 に、高度なジョブの発行例を示します。

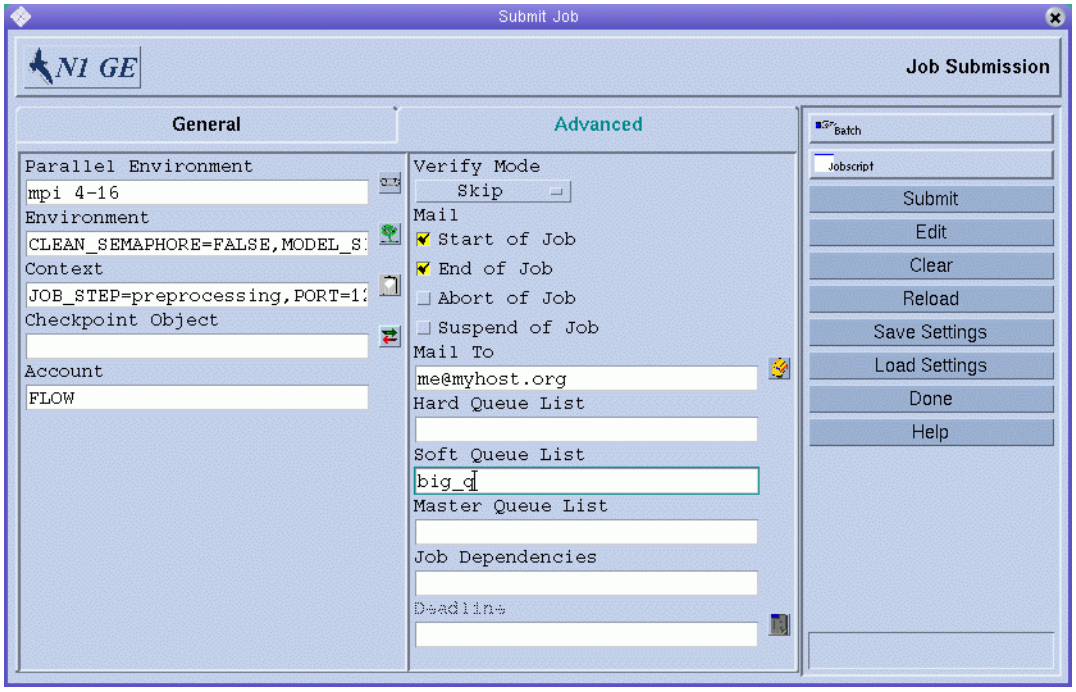


図 3–6　高度なジョブの発行例

62 ページの「拡張ジョブの例」で定義されているジョブは、59 ページの「QMON による拡張ジョブの発行」のジョブ定義と比較すると、さらに次のような特徴を備えています。

- 並列環境 `mpi` を使用する必要があります。4 つ以上のプロセスを作成する必要があり、使用可能な場合、プロセスを 16 個まで使用できます。
- ジョブに対して 2 つの環境変数が設定され、エクスポートされます。
- 2 つのコンテキスト変数が設定されます。
- アカウント文字列 `FLOW` が、ジョブアカウンティングレコードに追加されます。
- ジョブの開始と終了直後に、`me@myhost.org` にメールが送信されます。
- ジョブは、できるだけキュー `big_q` 内で実行するようにします。

コマンド行からの高度なジョブの発行

コマンド行から図 3–6 に示されている高度なジョブ要求を発行するには、次のコマンドを入力します。

```
% qsub -N Flow -p -111 -P devel -a 200012240000.00 -cwd \
 -S /bin/tcsh -o flow.out -j y -pe mpi 4-16 \
 -v SHARED_MEM=TRUE,MODEL_SIZE=LARGE \
 -ac JOB_STEP=preprocessing,PORT=1234 \
 -A FLOW -w w -m s,e -q big_q\
 -M me@myhost.com,me@other.address \
 flow.sh big.data
```

デフォルト要求ファイル

前のコマンドは、特に同じような要求を頻繁に発行する必要がある場合に、高度なジョブ要求は複雑で扱いにくくなる可能性があることを示しています。スクリプトファイルに qsub オプションを組み込むことによって、またはデフォルト要求ファイルを使用することによって、このようなコマンドを入力するという、面倒でエラーを引き起こしやすい作業を回避できます。詳細は、56 ページの「有効なコメント」を参照してください。

注 – -binary yes|no オプションで y 引数を指定すると、qrsh を使用してスクリプトラッパーなしで実行可能ジョブを発行できます。詳細は、qsub のマニュアルページを参照してください。

クラスタ管理では、すべての Grid Engine システムユーザーに対してデフォルト要求ファイルを設定できます。一方ユーザーは、ホームディレクトリに保存される個人用デフォルト要求ファイルを作成できます。ユーザーはまた、作業ディレクトリに保存されるアプリケーション固有のデフォルト要求ファイルを作成することもできます。

デフォルト要求ファイルには、デフォルトでは 1 行以上のジョブに適用される qsub オプションが含まれます。グローバルクラスタデフォルト要求ファイルは、*sge-root*/*cell*/common/sge_request に保存されます。個人用一般デフォルト要求ファイルは、$HOME/.sge_request に保存されます。アプリケーション固有のデフォルト要求ファイルは、$cwd/.sge_request に保存されます。

これらのファイルを複数使用できる場合、これらのファイルは次の優先順位で 1 つのデフォルト要求にマージされます。

1. アプリケーション固有のデフォルト要求ファイル
2. 個人用一般デフォルト要求ファイル
3. グローバルデフォルト要求ファイル

スクリプト組み込みと qsub コマンド行は、デフォルト要求ファイルより優先されます。よって、スクリプト組み込みはデフォルト要求ファイル設定を無効にします。また、qsub コマンド行オプションは、これらの設定を無効にします。

前の設定を破棄するには、デフォルト要求ファイル、組み込みスクリプトコマンド、または qsub コマンド行で qsub -clear コマンドを実行してください。

次に個人用デフォルト要求ファイルの例を示します。

```
-A myproject -cwd -M me@myhost.com -m b e
-r y -j y -S /bin/ksh
```

無効にされない限り、このユーザーのすべてのジョブで次のことが言えます。

- アカウント文字列は、myproject です。
- ジョブは、現在の作業ディレクトリ内で実行されます。
- メールの通知は、ジョブの最初と最後に me@myhost.com に送信されます。
- 標準出力と標準エラー出力はマージされます。
- ksh がコマンドインタプリタとして使用されます。

リソース要件の定義

ここまでの例では、発行オプションはジョブが実行されるホストに対するリソース要件を表していませんでした。Grid Engine システムは、このような場合のジョブはどのホストで実行してもよいとみなします。しかし、実際には、ほとんどのジョブで正常終了のために、実行ホストに関する前提条件が満たす必要があります。この前提条件には、十分な使用可能なメモリー、インストールが必要なソフトウェア、または特定のオペレーティングシステムアーキテクチャーなどがあります。また、クラスタ管理は通常、クラスタ内のマシンの使用に制限を加えます。たとえば、ジョブが消費できる CPU 時間はしばしば制限されます。

Grid Engine システムでは、クラスタの装置や使用ポリシーの正確な知識を持たなくても、ジョブに適したホストを見つけることができます。ユーザーはジョブの要件を指定し、適切で負荷が少ないホストの検出作業を Grid Engine システムに管理させます。

リソース要件は、42 ページの「要求可能な属性」で説明されている要求可能な属性によって指定します。 QMON を使用すると、ジョブの要件を指定するのに便利です。「Requested Resources」ダイアログボックスには、現在実行可能な「Available Resource」リストの属性だけが表示されます。「Requested Resources」ダイアログボックスを開くには、「Submit Job」ダイアログボックスの「Request Resources」をクリックします。例については、図 3–7 を参照してください。

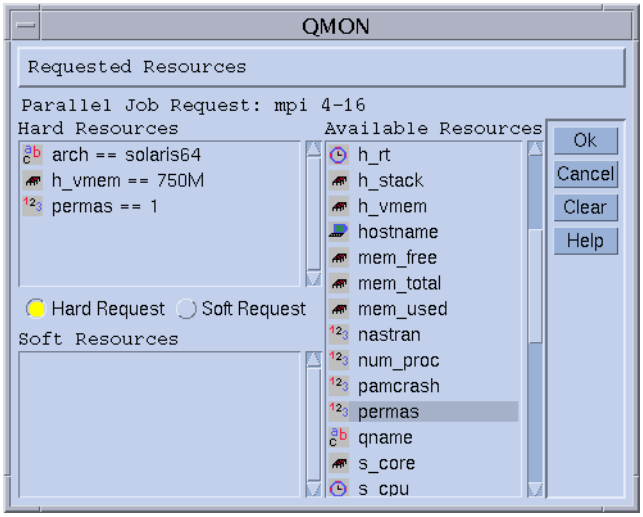


図 3–7 「Requested Resources」ダイアログボックス

属性をダブルクリックすると、その属性はジョブの「Hard Resources」または「Soft Resources」に追加されます。True に設定される BOOLEAN 属性以外の属性に値を入力するためのダイアログボックスが表示されます。詳細は、71 ページの「Grid Engine システムによるリソースの割り当て方法」を参照してください。

図 3–7 に、有効な `permas` ライセンスを持ち 750M バイト以上のメモリーを備えた `solaris64` ホストを必要とするジョブのリソースプロファイルを示します。この指定を満たすキューが複数見つかった場合は、定義されているソフトリソース要件が考慮されます。ハードとソフト要件を満たすキューが見つからなかった場合は、ハード要件を満たすキューが適切なものとみなされます。

注 – 複数のキューが 1 つのジョブに適している場合のみ、スケジューラ構成の `queue_sort_method` パラメータによってジョブの開始場所が決まります。詳細は、`sched_conf` (5) のマニュアルページを参照してください。

整数である属性 `permas` は、グローバルリソース属性の管理者向け拡張属性です。文字列である属性 `arch` は、ホストリソース属性です。メモリーの属性 `h_vmem` は、キューリソース属性です。

`qsub` コマンド行からも、これと同じリソース要件プロファイルを次のように発行できます。

```
% qsub -l arch=solaris64,h_vmem=750M,permas=1 \
    permas.sh
```

最初の `-l` オプションの前の暗黙的な `-hard` スイッチはスキップされます。

750M バイトを表す `750M` は、Grid Engine システムの量を表す構文の例です。メモリー消費が必要な属性に対しては、10 進数の整数、10 進数の浮動小数、8 進数の整数、および 16 進数の整数を指定できます。これらの数には次の乗数を加える必要があります。

- k – 値に 1000 を掛ける。
- K – 値に 1024 を掛ける。
- m – 値に 1000 × 1000 を掛ける。
- M – 値に 1024 × 1024 を掛ける。

8 進数の定数は、最初が 0 で次が 0 から 7 の数字でのみ指定されます。16 進数を指定する場合は、数の前に 0x を付けてください。また、0 から 9、a から f、A から F を使用します。乗数が使用されない場合、値はバイトとしてカウントするとみなされます。10 進数の浮動小数を使用する場合、結果は整数値に切り上げられます。

時間制限を課す属性の時間値は、時間単位、分単位、秒単位、またはこれらを組み合わせて指定することができます。時間、分、秒は、コロンで区切られた 10 進数で指定されます。`3:5:11` という時間は 11111 秒に変換されます。時間、分または秒の指定が 0 の場合、コロンが残っていれば指定をしなくても構いません。よって、`:5:` という値は 5 分と解釈されます。 図 3–7 に示されている「Requested Resources」ダイアログボックスの形式は拡張形式で、`QMON` 内でのみ有効です。

Grid Engine システムによるリソースの割り当て方法

前の節で示されているように、Grid Engine ソフトウェアがどのようにリソース要求を処理し、割り当てているのかを理解することは大切です。Grid Engine ソフトウェアのリソース割り当てアルゴリズムの概略は、次のようになります。

1. すべてのデフォルト要求ファイルを読み込み、構文解析します。詳細は、68 ページの「デフォルト要求ファイル」を参照してください。
2. 組み込みオプションのスクリプトファイルを処理します。詳細は、56 ページの「有効なコメント」を参照してください。
3. ジョブの発行時には、スクリプトファイル内の位置に関わらず、すべてのスクリプト組み込みオプションを読み取ります。
4. コマンド行からすべての要求を読み取り、構文解析します。

すべての `qsub` 要求が収集されるとすぐに、ハード要件とソフト要求が別々に処理されます。まず、ハード要求が処理されます。要求は、次の優先順位で評価されます。

1. スクリプトまたはデフォルト要求ファイルの左から右
2. スクリプトまたはデフォルト要求ファイルの上から下
3. コマンド行の左から右

言い換えると、コマンド行を使用して組み込みフラグを無効にすることができます。

ハードとして必要なリソースが割り当てられます。要求が有効でない場合、発行は拒否されます。1 つ以上の要求を発行時に満足できない場合、ジョブはスプールされ、後で実行するように再スケジューリングされます。たとえば、要求されたキューがビジーの場合、要求は満たされません。すべてのハード要求が満たされると、要求は割り当てられジョブを実行できます。

ソフトとして必要なリソースがチェックされます。これらの要求の一部またはすべてを満足できない場合もジョブは実行できます。ハード要求を満たす複数のキューがソフトリソースリストの一部を提供している場合、Grid Engine ソフトウェアはもっとも多くのソフト要求を提供しているキューを選択します。

ジョブが開始され、割り当てられたリソースが使用されます。

引数リストオプション、組み込みオプション、ハードおよびソフト要求が互いにどのような影響し合うのかを実際に試すとよいでしょう。`hostname` または `date` などの UNIX コマンドを実行する小さなテストスクリプトファイルで実験することができます。

ジョブの依存関係

大部分の場合、複雑なタスクを作成するのに一番便利なのは、タスクをサブタスクに分割することです。これらの場合、依存サブタスクを開始する前にその他のサブタスクの完了を待つ必要があります。たとえば、依存タスクが読み取って処理する出力ファイルが前のタスクによって作成されることなどがあります。

Grid Engine システムは、ジョブ依存関係機能で相互に依存するタスクをサポートしています。ユーザーは、1 つまたは複数のほかのジョブの完了に依存するジョブを設定できます。この機能は、`qsub -hold_jid` コマンドによって実行されます。発行ジョブが依存するジョブのリストを指定することもできます。ジョブのリストには、配列ジョブのサブセットも含まれます。依存関係リスト内のすべてのジョブが終了しなければ、発行ジョブは実行できません。

配列ジョブの発行

Grid Engine システムの 配列ジョブ機能の理想的な適用は、ジョブスクリプト内に含まれる操作のセットをパラメータ化して繰り返し実行することです。この典型例は、レンダリングなどで、デジタルコンテンツ作成業界でよくみられます。アニメーションのコンピュータ処理はフレームに分割されます。同じレンダリング処理が、各フレームに対して別々に行われます。

配列ジョブ機能によって、このような適用での発行、監視、および制御が便利になります。Grid Engine システムは、1 つのジョブに結合された別々のタスクの配列としてコンピュータ処理を行い、配列ジョブを効率的に実行します。配列ジョブのタスクは、配列インデックス番号によって参照されます。すべてのタスクのインデック

スは、配列ジョブ全体のインデックス範囲に及びます。インデックス範囲は、配列ジョブの発行時に qsub コマンドによって定義されます。

ユーザーは、 配列ジョブの監視および制御を行うことができます。たとえば、全体として、またはそれぞれのタスクやタスクのサブセットごとに、配列ジョブを一時停止、再開、取り消しすることができます。タスクを参照するために、対応するインデックス番号が ジョブ ID の前に付けられています。タスクは、一般的なジョブとほぼ同じように実行されます。タスクは、環境変数 SGE_TASK_ID を使用して、独自のタスク インデックス番号を検索し、そのこのタスク 識別子に対する入力データセットにアクセスすることができます。

QMON による配列ジョブの発行

次の情報も考慮しながら、49 ページの「QMON により単純なジョブを発行する」の手順に従ってください。

QMON による配列ジョブの発行は、 49 ページの「QMON により単純なジョブを発行する」に説明されている単純なジョブの発行方法とほぼ同じです。 唯一の違いは、図 3–5 の「Job Tasks」タスク入力ウィンドウにタスク範囲指定があることです。タスク範囲の指定では、qsub -t コマンドと同じ構文を使用します。配列インデックスの構文については、 qsub(1) のマニュアルページを参照してください。

一般的なジョブの監視と制御および特定の配列ジョブについては、89 ページの「ジョブの監視と制御」と 99 ページの「コマンド行からのジョブの監視と制御」を参照してください。qstat(1)、qhold(1)、 qrls(1)、qmod(1)、および qdel(1) のマニュアルページを参照してください。

配列ジョブは、通常のジョブで使用可能な Grid Engine システムの全機能にフルアクセスできます。特に、配列ジョブは同時に並列ジョブとなることもできます。配列ジョブも、その他のジョブと相互依存関係を持つことができます。

注 – 配列タスクは、その他のジョブまたはその他の配列タスクとの相互依存関係を持つことはできません。

コマンド行からの配列ジョブの発行

コマンド行から配列ジョブを発行するには、適切な引数を指定して qsub コマンドを入力します。

次に、配列ジョブの発行方法の例を示します。

```
% qsub -l h_cpu=0:45:0 -t 2-10:2 render.sh data.in
```

-t オプションは、タスクインデックスの範囲を指定します。この場合、2-10:2 は、2 が最小インデックス番号で、10 が最大インデックス番号という範囲を指定します。:2 という指定により、1 つ飛ばしのインデックスだけが指定されます。よって、配

列ジョブはタスクインデックス 2、4、6、8、10 の 5 つのタスクによって構成されます。各タスクは、-l オプションで 45 分のハード CPU 時間制限を要求します。タスクが Grid Engine システムによって振り分けられ、開始されると、各タスクはジョブスクリプト render.sh を実行します。タスクは SGE_TASK_ID を使用してインデックス番号を探し、その番号を使用してデータファイル data.in 内の入力データレコードを見つけることができます。

対話型ジョブの発行

ユーザーが直接入力を行い、ジョブの結果に影響を与える場合は、 バッチジョブの代わりに対話型ジョブを発行すると便利です。このような場面は、X ウィンドウアプリケーションや、さらに処理を進めるためにユーザーが結果をすぐに解釈しなければいけないタスクなどで一般的に見られます。

対話型ジョブは、次の 3 つの方法で作成できます。

- qlogin – Grid Engine ソフトウェアが選択したホスト上で開始される telnet のようなセッション。
- qrsh – 標準的な UNIX rsh 機能と同じ。コマンドは、Grid Engine システムが選択したホスト上でリモート実行されます。コマンドが指定されていない場合は、リモート rlogin セッションがリモートホストで開始されます。
- qsh – ジョブを実行しているマシンから表示される xterm。表示は、ユーザーの指定または DISPLAY 環境変数の設定に従って設定されます。DISPLAY 変数を設定せず、表示先が定義されていない場合、Grid Engine システムは xterm をジョブが発行された元のホストの X サーバーの 0.0 画面に送ります。

注 – 正しい機能のために、すべての機能で Grid Engine システムのクラスタパラメータを正しく構成してください。また、正しい xterm 実行パスを qsh に対して定義してください。この種のジョブでは、対話型キューを使用しなければなりません。クラスタで対話型ジョブを実行できる体制が整っているかどうかについては、システム管理者にお問い合わせください。

対話型ジョブのデフォルト処理は、バッチジョブの処理とは異なります。発行時に実行できなかった場合、対話型ジョブはキューには入れられません。ジョブがキューに入れられないということは、ジョブの発行時に対話型ジョブを振り分ける適切なリソースが十分なかったということです。クラスタが現在ビジー状態になっている場合は、ユーザーに通知が行われます。

このデフォルト動作は、qsh、 qlogin、および qrsh の -now no オプションで変更できます。このオプションを使用すると、対話型ジョブはバッチジョブのようにキューに入れられます。また、 -now yes オプションを使用して、qsub で発行されたバッチジョブを対話型ジョブのように処理することもできます。この場合のバッチジョブはすぐに実行のために振り分けられるか、拒否されます。

注 – 対話型ジョブは、INTERACTIVE タイプのキュー内でのみ実行されます。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「キューの構成」を参照してください。

次の節では、qlogin と qsh 機能の使用方法について説明します。qrsh コマンドは、78 ページの「透過的なリモート実行」のより広い文脈で説明します。

QMON による対話型ジョブの発行

QMON から発行できる対話型ジョブは、Grid Engine システムが選択したホストで xterm を発行するジョブだけです。

「Interactive」アイコンが表示されるまで、「Submit Job」ダイアログボックスの右側の「Submit」ボタンの上にあるボタンをクリックします。これにより、対話型ジョブを発行するための「Submit Job」ダイアログボックスの準備が整えられます。図 3–8 と図 3–9 を参照してください。

ダイアログボックスの選択オプションの意味と使用方法は、53 ページの「バッチジョブの発行」のバッチジョブの説明と同じです。違いは、対話型ジョブには適用されない入力フィールドがグレー表示になっていることです。

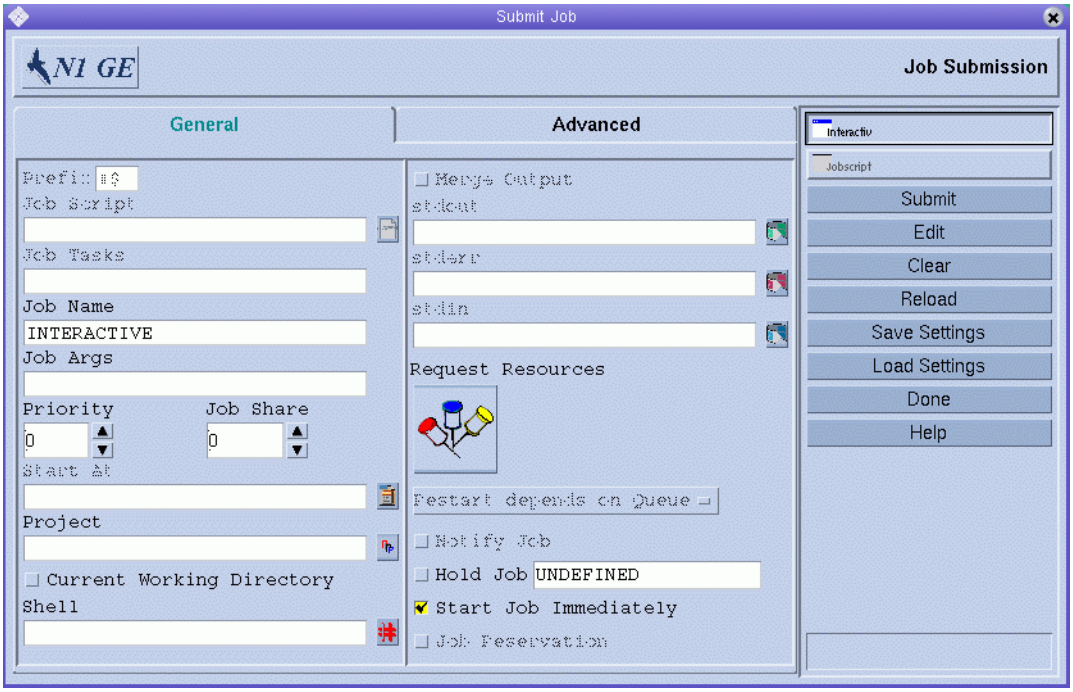


図 3–8 「Interactive Submit Job」ダイアログボックス、「General」タブ

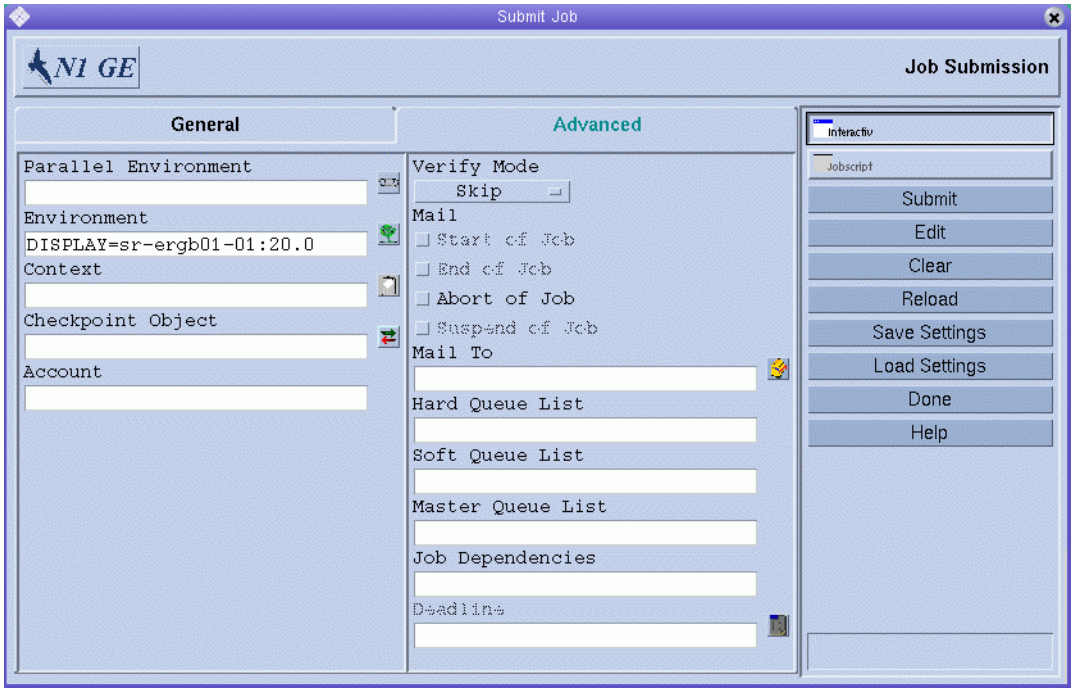


図 3–9 「Interactive Submit Job」ダイアログボックス、「Advanced」タブ

qsh による対話型ジョブの発行

qsh は、qsub と非常に似ています。qsh は、いくつかの qsub オプション、および呼び出される xterm の表示を指示する追加オプション -display をサポートしています。詳細は、 qsub(1) のマニュアルページを参照してください。

qsh で対話型ジョブを発行するには、次のようなコマンドを入力します。

```
% qsh -l arch=solaris64
```

このコマンドは、使用可能な任意の Sun Solaris 64 ビットオペレーティングシステムのホストで xterm を開始します。

qlogin による対話型ジョブの発行

端末または端末エミュレーションで qlogin コマンドを使用して、Grid Engine システムの制御下で対話セッションを開始します。

qlogin で対話型ジョブを発行するには、次のようなコマンドを入力します。

```
% qlogin -l star-cd=1,h_cpu=6:0:0
```

このコマンドは、負荷の少ないホストを特定します。ホストには、有効な Star-CD ライセンスがあります。また、ホストには、最低 6 時間のハード CPU 時間制限を持つキューが 1 つ以上あります。

注 – Grid Engine システムによって使用されるように設定されているリモートログイン機能によっては、ログインプロンプトでユーザー名、パスワード、またはこの両方を入力しなければならない場合があります。

透過的なリモート実行

Grid Engine システムは、特定のコンピュータ処理タスクの透過的なリモート実行をサポートする密接な関連を持つ機能のセットを提供します。この機能の中心的なツールは、78 ページの「qrsh によるリモート実行」に説明がある `qrsh` コマンドです。`qtcsh` と `qmake` という 2 つの上位機能が `qrsh` の上にあります。これらの 2 つのコマンドによって、Grid Engine システムは暗黙的なコンピュータ処理タスクを透過的に分散し、標準的な UNIX 機能である `make` と `csh` を拡張できます。`qtcsh` の説明は、80 ページの「qtcsh による透過的なジョブの分配」にあります。`qmake` の説明は、82 ページの「qmake による並列 Makefile 処理」にあります。

qrsh によるリモート実行

`qrsh` は、標準 `rsh` 機能を基に構築されています。`rsh` の関係については、*sge-root*`/3rd_party` の情報を参照してください。`qrsh` は、次のようなさまざまな目的に使用できます。

- 標準的な UNIX 機能である `rsh` と比較できる Grid Engine システムを使用する対話型アプリケーションのリモート実行。`rsh` は、HP-UX システムでは `remsh` とも呼ばれます。
- Grid Engine システムを使用する対話型ログインセッション機能の提供。標準の UNIX 機能である `rlogin` に似ています。Grid Engine システムが UNIX `telnet` 機能を表現するために、`qlogin` は今でも必要とされます。
- 端末 I/O (標準出力、標準エラー、および標準入力) および端末制御をサポートするバッチジョブの発行の許可。
- シェルスクリプトに組み込まれていないスタンドアロンプログラムの発行方法の提供

注 – `qrsh` で `-b n` オプションを使用することで、スクリプトを発行することもできます。詳細は、`qrsh` のマニュアルページを参照してください。

- バッチジョブが保留中または実行中のときにアクティブのままで、ジョブが終了または取り消されたときだけ停止される発行クライアントの提供
- 並列ジョブによって割り当てられた分散リソースのフレームワーク内での Grid Engine システム制御によるジョブタスクのリモート実行。『Sun N1 Grid Engine 6.1 管理ガイド』の「並列環境と Grid Engine ソフトウェアの密統合」を参照してください。

これらの機能によって、qrsh は qtcsh および qmake 機能を実装する主な使用可能インフラストラクチャーとなっています。qrsh は、Grid Engine システムを MPI や PVM などの 並列環境と密接に統合するためにも使用されます。

qrsh による透過的なリモート実行の呼び出し

qrsh コマンドを入力し、次の構文に従ってオプションと引数を追加します。

```
% qrsh     [options] program|shell-script [arguments] \
    [> stdout] [>&2 stderr] [< stdin]
```

qrsh は、qsub のほとんどすべてのオプションに対応しています。 qrsh は、次のオプションを提供します。

- -now yes|no – -now yes は、ただちにジョブのスケジューリングを行うように指定します。適切なリソースを使用できない場合、ジョブは拒否されます。-now yes はデフォルトです。-now no は、発行時にジョブを開始できない場合に、バッチジョブのようにジョブをキューに入れるように指定します。
- -inherit – qrsh は、ジョブタスクを開始するスケジューリングプロセスを実行しません。qrsh は、そのジョブが、指定されたリモート実行ホスト上に適切な割り当てリソースを持っている並列ジョブに組み込まれているとみなします。この形式の qrsh は一般的に、qmake または密接な並列環境の統合で使用されます。デフォルトでは、外部ジョブ リソースは継承されません。
- -binary yes|no – n オプションを指定すると、qrsh を使用してスクリプトジョブを発行できます。
- -noshell – このオプションでは、ユーザーのログインシェルで qrsh に対して指定されているコマンド行は開始されません。このコマンドは、包括するシェルなしで実行されます。このオプションは、シェルの開始およびシェルリソースファイルの調達などのオーバーヘッドを避け、実行速度を速めるために使用してください。
- -nostdin – 入力ストリーム STDIN を制御します。このオプションを設定すると、qrsh が -n オプションを rsh コマンドに渡します。入力ストリームの抑制は、たとえば make プロセスなどで、qrsh を使用して複数のタスクを並列実行している場合に特に役立ちます。入力を受けるプロセスは未定義です。
- -verbose – このオプションは、スケジューリングプロセスの出力を表します。-verbose は主にデバッグのために使用されるので、デフォルトではオフになっています。

qtcsh による透過的なジョブの分配

qtcsh は、一般的に使用されるよく知られた UNIX C 派生シェル tcsh の完全な互換性を持つ代用コマンドです。 qtcsh は、tcsh を基に構築されています。tcsh の関係については、*sge-root*/3rd_party の情報を参照してください。qtcsh は、Grid Engine システムを使用する適切で負荷の少ないホストに指定アプリケーションを透過的に分散する拡張機能をコマンドシェルに提供します。.qtask 構成ファイルには、リモート実行するアプリケーションと、実行ホストの選択に適用される要件が定義されています。

ユーザーに対して透過的なこれらのアプリケーションは、qrsh 機能を使用して Grid Engine システムに発行されます。qrsh は、標準出力、エラー出力と標準入力処理、およびリモート実行アプリケーションへの端末制御接続を提供します。アプリケーションをリモートで実行する場合と同一ホストでシェルとして実行する場合の明確な違いは、3 つあります。

- リモートホストの方がより強力で、負荷が少なく、インストールが必要なハードウェアおよびソフトウェアリソースを備えている場合もあります。このようなリモートホストは、アプリケーションをまったく実行できない場合もあるローカルホストより、はるかに実行に適しています。
- ジョブのリモート起動と Grid Engine システムを通したジョブ処理によって、若干の遅れが生じます。
- 管理者は、対話型ジョブ (qrsh)、よって qtcsh によるリソースの使用を制限することができます。qrsh によってアプリケーションを開始するのための適切なリソースを十分使用できない場合、またはすべての適切なシステムに負荷がかかりすぎている場合は、暗黙的な qrsh の発行は失敗し、「Not enough resources ... try later」などの該当エラーメッセージが返されます。

標準的な使用以外に、qtcsh は、サン以外のコードやツールとの統合用に適したプラットフォームでもあります。qtcsh のシングルアプリケーション実行形式は、qtcsh -c *app-name* です。統合環境内部でこの書式の qtcsh を使用すると、ほぼ永久的に変更の必要がない持続的なインタフェースが表示できます。すべての必要なアプリケーション、ツール、統合、サイト、およびユーザー固有の 構成までもが、適切に定義された .qtask ファイルには含まれています。さらに、すべての種類のシェルスクリプト、C プログラム、および Java アプリケーションでも使用できることもこのインタフェースの長所です。

qtcsh の使用

qtcsh の呼び出しは、 tcsh の呼び出しとまったく同じです。qtcsh は、 .qtask ファイルのサポートを提供し、特殊化されたシェル内蔵モードのセットを提供することで、tcsh を拡張します。

.qtask ファイルは次のように定義されます。ファイルの各行は次のような書式を持ちます。

```
% [!]app-name qrsh-options
```

最初に付けられるオプションのエクスクラメーションマーク (!) は、グローバルクラスタ .qtask ファイルと qtcsh ユーザーの個人用 .qtask ファイルの定義が相反する場合の優先順位を定義します。グローバルクラスタファイルにエクスクラメーションマークがない場合は、ユーザーファイルの定義がグローバルクラスタファイルの定義より優先されます。グローバルクラスタファイルにエクスクラメーションマークがある場合は、グローバルクラスタファイルの定義が優先されます。

app-name は、qtcsh のコマンド行で入力された場合に、リモート実行向けに Grid Engine システムに発行されるアプリケーション名を指定します。

qrsh-options は、使用される qrsh 機能のオプションを指定します。これらの オプションは、アプリケーションのリソース要件を定義します。

アプリケーション名は、アプリケーションが .qtask ファイルで定義されるのとまったく同じようにコマンド行に表示されなければなりません。アプリケーション名の前にパス名が追加されている場合は、ローカルバイナリがアドレス指定されます。リモート実行は行われません。

csh エイリアスは、アプリケーション名との比較が行われる前に拡張されます。リモート実行向けのアプリケーションは、qtcsh コマンド行のどこででも実行できますが、特に標準入出力先のリダイレクトの前後に実行されます。

よって、次の例は有効で意味のある構文となります。

```
# .qtask file
netscape -v DISPLAY=myhost:0
grep -l h=filesurfer
```

この .qtask ファイルを前提とすると、次の qtcsh コマンド行は、

```
netscape
~/mybin/netscape
cat very_big_file | grep pattern | sort | uniq
```

暗黙的に次のような結果になります。

```
qrsh -v DISPLAY=myhost:0 netscape
~/mybin/netscape
cat very_big_file | qrsh -l h=filesurfer grep pattern | sort | uniq
```

qtcsh は複数のモードで実行でき、オンまたはオフの状態であるスイッチに影響を受けます。

- コマンドのローカルまたはリモート実行。デフォルトは、リモートです。
- 即時実行またはバッチリモート実行。デフォルトは、即時実行です。
- 冗長出力または非冗長出力。デフォルトは、非冗長出力です。

これらのモードの設定は、開始時に qtcsh のオプション引数を使用して、または実行時にシェル内蔵コマンド qrshmode を使用して変更できます。詳細は、 qtcsh(1) のマニュアルページを参照してください。

qmake による並列 Makefile 処理

qmake は、標準的な UNIX make 機能の代わりとなる機能です。qmake は、適切なマシンのクラスタ上で make の各手順の分散を可能にすることにより、make の機能を拡張したものです。qmake は、よく使用される GNU-make 機能の gmake を基に構築されています。gmake の関係については、 *sge-root*/3rd_party の情報を参照してください。

分散 make プロセスを最後まで実行できるように、 qmake はまず、必要なリソースを並列ジョブと同じような方法で割り当てます。qmake は、次にスケジューリング機能と対話せずに、このリソースのセットを管理します。リソースが使用可能になると、qmake は -inherit オプションを設定した qrsh 機能を使用して、 make の手順を分散させます。

qrsh は、標準出力、エラー出力と標準入力の処理、およびリモートで実行されている make 手順への端末制御接続を提供します。このため、 make 手続きのローカル実行と qmake の使用との間には、明確な違いが 3 つあるだけです。

- それぞれの make 手順が特定の時間を持ち、処理する make 手順がそれぞれ十分にある場合、make プロセスの並列化により処理速度を大幅に上げることができます。
- リモートで起動される make 手順では、qrsh とリモート実行が原因で少しのオーバーヘッドが生じます。
- qmake の make 手順の分散を利用するためには、並列化の最低レベルを指定する必要があります。つまり、ユーザーは同時に実行可能な make 手順数を指定しなければなりません。さらに、ユーザーは、有効なソフトウェアライセンス、マシンアーキテクチャー、メモリーまたは CPU 時間要件などの、make 手順に必要なリソースの特徴を指定できます。

make がもっとも一般的に使用されるのは、複雑なソフトウェアパッケージのコンパイルです。しかし、qmake はコンパイルにはそれほど使用されないこともあります。プログラムファイルは、適切なプログラミングの結果、かなり小さくなることがよくあります。したがって、1 つのプログラムファイルのコンパイル (1 つの make 手順) は、大部分の場合、数秒しかかかりません。さらに通常、コンパイルでは大量のファイルアクセスが行われます。ネスト化されたインクルードファイルは、この問題の原因となりえます。ファイルサーバーがボトルネックとなることがあるため、複数の make 手順に対してコンパイルを並行して実行した場合、ファイルのアクセス速度を速めることができない場合があります。ボトルネックは、すべてのファイルアクセスを事実上直列化するからです。したがって、満足のいく形ではコンパイルプロセスを高速化できない場合があります。

qmake は、その他の使用方法においてより効果を発揮できます。たとえば、makefile によって相互依存関係や複雑な分析処理の作業フローを管理することができます。この環境での make の各手順は一般的に、無視できないリソースとコンピュータ処理時間要件を持つシミュレーションやデータ分析処理です。この場合は、かなりの高速化を達成できます。

qmake の使用

qmake のコマンド行構文は、qrsh の構文と同じようにみえます。

```
% qmake [-pe pe-name pe-range][options] \
 -- [gnu-make-options][target]
```

注 – --inherit オプションは、この節の後半に説明がある qmake によってもサポートされています。

-pe オプションの使用と gmake -j オプションとの関係に特に注意してください。両方のオプションは、達成する並列化の量を表すために使用できます。違いは、 gmake は -j を使用して、使用される並列環境などを指定できないことです。したがって、qmake は、make と呼ばれる並列 make 用のデフォルト環境が構成されていることを前提にします。さらに、gmake の -j では、範囲の指定はできず、1 つの数字しか設定できません。qmake は、-j によって指定された数字を 1-*n* の範囲として解釈します。これと対照的に、-pe ではこれらのパラメータすべてを詳細に指定できます。結果として、次の 2 つのコマンド行例の意味は同じになります。

```
% qmake -- -j 10
% qmake -pe make 1-10 --
```

次の コマンド行は、-j オプションを使用して表現することはできません。

```
% qmake -pe make 5-10,16 --
% qmake -pe mpi 1-99999 --
```

構文以外に、qmake は 2 つの呼び出しモードをサポートしています。-inherit オプションを使用せずにコマンド行から対話方式で呼び出すモードと -inherit オプションによってバッチジョブ内で呼び出すモードがあります。これらの 2 つのモードは、次のような別の処理シーケンスを開始します。

- 対話型 – qmake がコマンド行で呼び出されると、make プロセスは qrsh によって、暗黙的に Grid Engine システムに発行されます。プロセスは、qmake コマンド行で指定されたリソース要件を考慮します。次に Grid Engine システムは、並列 make ジョブに関連付けられた並列ジョブを実行するためのマスターマシンを選択します。Grid Engine システムは、ここで make 手続きを開始します。make プロセスがアーキテクチャーに依存している場合もあるので、手続きはここで開始しなければなりません。必要なアーキテクチャーは、qmake コマンド行に指定されています。マスターマシンの qmake プロセスが、それぞれの make 手順の実行をジョブに割り当てられているその他のホストに委託します。手順は、並列環境ホストファイルによって qmake に渡されます。
- バッチ – この場合、qmake は -inherit オプションを持つバッチスクリプト内にあります。-inherit オプションがない場合は、最初の例で説明されているように、新しいジョブが生成されます。これにより、qmake は、qmake が組み込まれているジョブに割り当てられているリソースを利用することになります。qmake は、qrsh -inherit を直接使用して、 make 手順を開始します。qmake をバッチモードで呼び出すと、リソース要件の指定、-pe オプションおよび -j オプションは無視されます。

注 – 1 つの CPU ジョブでも次のように並列環境が必要です。

```
qmake -pe make 1 --
```

並列実行が必要ない場合は、Grid Engine システムオプションと -- のない gmake コマンド行構文で、qmake を呼び出してください。この qmake コマンドは、gmake のように動作します。

詳細は、qmake(1) のマニュアルページを参照してください。

ジョブのスケジューリング方法

Grid Engine ソフトウェアのポリシー管理では、管理目標を最適な形で達成するために、クラスタ内の共有リソースの使用を自動的に制御します。優先順位の高いジョブは優先的に振り分けられ、リソースにより多くアクセスできます。クラスタ管理では、ハイレベルの使用ポリシーを定義できます。次のポリシーが使用可能です。

- 機能 – 特定のユーザーグループ、プロジェクトなどへの所属によって、特別な処理を行うことができます。
- 共有ベース – サービスのレベルは、割り当てられた共有エンタイトルメント、その他のユーザーやユーザーグループの共有、ユーザー全員の過去のリソース使用率、およびシステムにおける現在のユーザーの存在によって異なります。
- 緊急度 – 緊急度の高いジョブが優先的に処理されます。ジョブの緊急度は、リソース要件、ジョブの待機時間、ジョブが期限付きで発行されたかどうかを基に決められます。
- 優先 – クラスタ管理者が手動で介入することで、自動ポリシー実行が変更されます。

Grid Engine ソフトウェアは、共有ベースのポリシー、機能ポリシー、または両方を日常的にに使用するように設定できます。これらのポリシーはどんな割合で組み合わせることもできます。1 つのポリシーの重みを 0 にして 2 番目のポリシーだけを使用することも、両方のポリシーに同じ重みを与えることもできます。

ルーチンポリシーに加えて、ジョブに開始期限を付けて発行することもできます。64 ページの「QMON による高度なジョブの発行」の期限発行パラメータの説明を参照してください。期限ジョブは、ルーチンスケジューリングに干渉します。管理者は、共有ベーススケジューリングと機能スケジューリングを一時的に無効にすることもできます。無効は、個別ジョブ、あるユーザーに関連するすべてのジョブ、部門、またはプロジェクトに対して適用できます。

ジョブの優先順位

すべてのジョブ間で調整を行う 4 つのポリシー以外に、Grid Engine ソフトウェアではユーザーが自分のジョブ間の優先順位を設定できる場合もあります。複数のジョブを発行したユーザーは、たとえば、ジョブ 3 が一番重要で、ジョブ 1 と 2 の重要度は同じだが、ジョブ 3 よりは重要ではないなどと指定することができます。

ジョブの優先順位は、`QMON` Submit Job パラメータ Priority または `qsub -p` オプションを使用して設定します。1024 (最低) から 1023 (最高) の優先順位範囲を指定できます。優先順位によって、同時に 1 人のユーザーの複数のジョブがシステム内に存在する場合に、どのようにしてそのユーザーのジョブから選択を行うかがスケジューラに

伝えられます。特定のジョブに割り当てられる相対的な重要度は、ユーザーのジョブに対して指定された最高と最低の優先順位、および特定のジョブの優先順位によって異なります。

チケットポリシー

機能ポリシー、共有ベースのポリシーおよび優先ポリシーはすべて、チケットを使用して実行されます。各チケットポリシーには、マルチマシン Grid Engine システムに入力されるジョブに割り当てられるチケットの割り当て元であるチケットプールがあります。有効な各ルーチンチケットポリシーは、複数のチケットをすべての新しいジョブに割り当てます。チケットポリシーは、スケジューリング間隔で実行中のジョブにチケットを再割り当てすることもできます。各チケットポリシーがチケットを割り当てるために使用する条件を、この節で説明します。

チケットは、3 つのポリシーに重み付けを行います。たとえば、機能ポリシーにチケットが割り当てられていない場合、機能ポリシーは使用されません。同じ数のチケットが機能チケットプールと共有ベースのチケットプールに割り当てられている場合は、両方のポリシーがジョブの重要度の決定において等しい重さを持ちます。

Grid Engine 管理者が、システム構成時にチケットをルーチンチケットポリシーに割り当てます。管理者とオペレータは、いつでもチケットの割り当てを変更できます。無効を示すために追加チケットを一時的にシステムに投入することもできます。チケットポリシーは、チケットの割り当てによって組み合わせられます。チケットが複数のチケットポリシーに割り当てられている場合、ジョブは有効な各チケットポリシーから一部のチケットを受け取ります。

Grid Engine システムは、有効な各チケットポリシーでのジョブの重要度を示すために、システムに入力されるジョブにチケットを与えます。実行中の各ジョブは、チケットを優先ポリシーなどから受け取り、ジョブがリソースの適当な共有以上を受け取っている場合などには、チケットを失います。また各スケジューリング間隔で同じ数のチケットを持ち続けます。ジョブが持っているチケットの数は、Grid Engine システムが各スケジューリング間隔でジョブに与えようとするリソース共有を表します。

ジョブが持っているチケットの数は、 `QMON` または `qstat -ext` で表示できます。89 ページの「`QMON` を使用したジョブの監視と制御」を参照してください。`qstat` コマンドは、たとえば `qsub -p` などによって、優先順位を表示することもできます。詳細は、 `qstat`(1) のマニュアルページを参照してください。

キューの選択

ジョブをすぐに開始できない場合、Grid Engine システムは不定のキューを要求するジョブを振り分けません。このようなジョブは、 ジョブのスケジューリングを適宜試行する `sge_qmaster` でスプールとしてマークされます。このようなジョブは使用可能になった次の適切なキューに振り分けられます。

スプーリングジョブとは反対に、名前を指定して特定のキューに発行されたジョブは、ジョブを開始できるかスプールする必要があるかに関わらず、そのキューに直接送られます。したがって、名前を指定されたジョブ要求に対してのみ、Grid Engine システムのキューを情報工学で言われるバッチキューとみなすことができます。特定の要求なしで発行されたジョブは、キューに入るために、`sge_qmaster` のスプーリングメカニズムを使用するので、より抽象的で柔軟なキュー概念が採用されます。

ジョブのスケジューリングを行い、複数の空きキューがリソース要求を満たす場合、ジョブは通常一番負荷が少ないホストに属する適切なキューに振り分けられます。スケジューラ 構成エントリ `queue_sort_method` を `seq_no` に設定すると、クラスタ管理でこの負荷に依存したスキームを固定順のアルゴリズムに変更できます。キュー構成エントリ `seq_no` は、キュー間の優先順位を定義し、一番高い優先順位をシーケンス番号が一番小さいキューに割り当てます。

◆ ◆ ◆ 第 4 章

ジョブとキューの監視と制御

ジョブの発行後は、ジョブを監視および制御をする必要があります。この章には、ジョブとキューの監視と制御に関する内容説明およびこれらのタスクの実行手順が記載されています。また、チェックポイントの設定に関する情報も紹介します。

この章では、次の作業の手順を紹介します。

ジョブの監視と制御

発行したジョブは、次の 3 つの方法で監視および制御することができます。

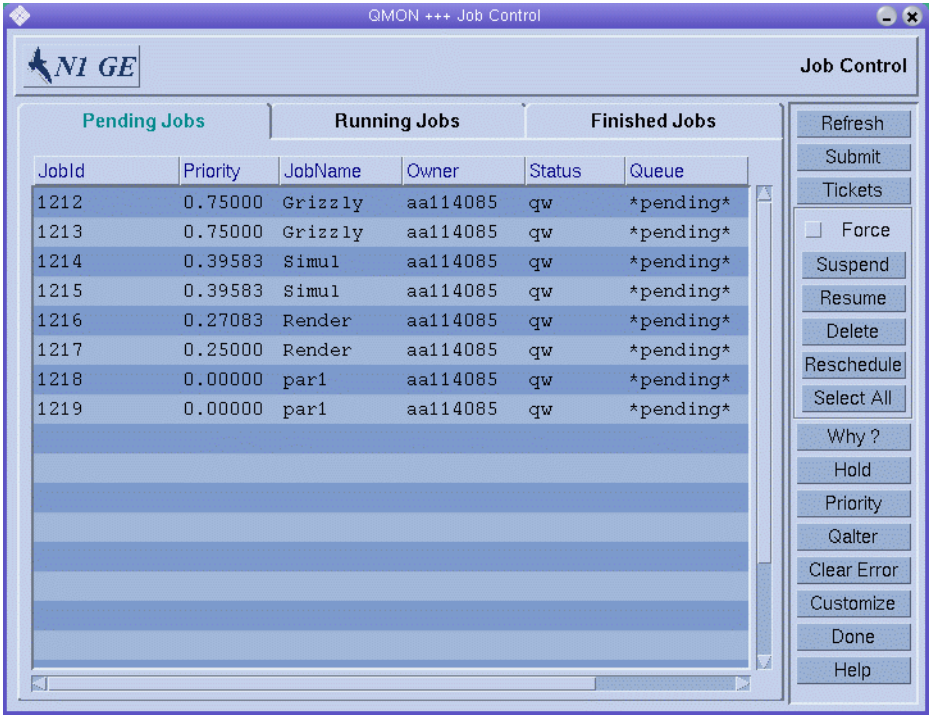
- QMON の使用
- コマンド行の qstat、qdel および qmod コマンドの使用
- 電子メールの使用

これらの各方法については、次の節で説明します。

QMON を使用したジョブの監視と制御

「QMON Job Control」ダイアログボックスを使用して、ジョブを制御できます。

発行したジョブの監視と制御を行うには、「QMON Main Control」ウィンドウで「Job Control」ボタンをクリックします。「Job Control」ダイアログボックスが表示されます。



「Job Control」ダイアログボックスには、実行中のジョブのタブ (「Running Jobs,」)、適切なリソースへ振り分けられるのを待っている保留ジョブのタブ (「Pending Jobs」)、および最近終了したジョブのタブ (「Finished Jobs」) の 3 つのタブがあります。

「Submit」ボタンは、「Submit Job」ダイアログボックスへのリンクを提供します。

「Job Control」ダイアログボックスを使用すると、システムが認識している実行中のジョブ、保留中のジョブおよび終了ジョブをすべて監視できます。また、このダイアログボックスはジョブの管理にも使用できます。ジョブの優先順位を変更することができ、ジョブの一時停止、再開、および取り消しも実行できます。

デフォルト形式の「Job Control」ダイアログボックスでは、実行中および保留中の各ジョブに対して次の列が表示されます。

- 「JobId」
- 「Priority」
- 「JobName」
- 「Owner」

- 「Status」
- 「Queue」

形式をカスタマイズすると、デフォルトの表示を変更できます。詳細は、95 ページの「「Job Control」ディスプレイのカスタマイズ」を参照してください。

「Job Control」ディスプレイの再表示

表示される情報を最新の状態に保つため、`QMON` はポーリング方式でジョブのステータスを `sge_qmaster` から取り出します。「Job Control」ディスプレイの再表示を強制的に行うには、「Refresh」をクリックしてください。

ジョブの選択

ジョブは、次のマウス操作やキーの組み合わせで選択できます。

- 複数の非連続的なジョブを選択するには、Control キーを押しながら 2 つ以上のジョブをクリックします。
- 連続した範囲のジョブを選択するには、Shift キーを押しながら範囲の最初のジョブをクリックして、範囲の最後のジョブをクリックします。
- ジョブの選択と選択解除を切り替えるには、 Control キーを押しながらそのジョブをクリックします。

フィルタを使用して、表示するジョブを選択することもできます。詳細は、96 ページの「ジョブリストのフィルタリング」を参照してください。

ジョブの管理

ダイアログボックスの右側のボタンを使用して、選択したジョブを次のように管理できます。

- 一時停止
- 再開（一時停止の解除）
- 削除
- 待機
- 解放
- 優先順位の再設定
- 再スケジューリング
- `qalter` による変更

ジョブの所有者または Grid Engine 管理者およびオペレータだけが、ジョブの一時停止、再開、削除、待機、およびジョブの優先順位の変更とジョブの変更を実行できます。38 ページの「管理者、オペレータ、および所有者」を参照してください。一時停止または再開できるのは実行中のジョブだけです。再スケジューリングしたり、待機させたり、優先順位やその他の属性を変更できるのは、保留中のジョブだけです。

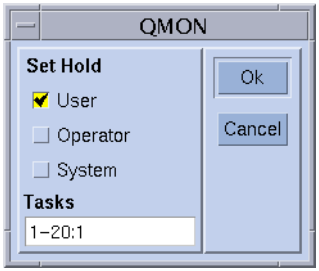
ジョブを一時停止すると、UNIX の killコマンドによってジョブの処理グループに SIGSTOP 信号が送信されます。SIGSTOP はジョブを停止し、CPU 時間を消費しなくなります。ジョブを再開すると、 SIGCONT 信号が送信され、これによってジョブの一時停止が解除されます。信号処理プロセスについては、お使いのシステムの kill(1) のマニュアルページを参照してください。

注 – ジョブの一時停止、再開、および削除は、強制的に実行できます。言い換えれば、そのジョブを制御する sge_execd に通知せずに、sge_qmaster を使用して、これらの処理を登録できます。強制実行は、たとえばネットワーク問題が原因で対応する sge_execd を実行できない場合に有効です。この操作を行うには、「Force」オプションを選択してください。

現在実行中のジョブを再スケジューリングするには、「Reschedule」をクリックします。

ジョブの待機

ジョブを待機させるには、保留中のジョブを選択し、「Hold」をクリックします。「Set Hold」ダイアログボックスが表示されます。



「Set Hold」ダイアログボックスでは、ユーザー、オペレータおよびシステムを待機させたり、待機状態をリセットしたりすることができます。ユーザーの待機を設定およびリセットできるのは、ジョブの所有者および Grid Engine 管理者とオペレータです。オペレータの待機を設定およびリセットできるのは、管理者とオペレータです。システムの待機を設定およびリセットできるのは、管理者だけです。ジョブになんらかの種類の待機が割り当てられている間は、ジョブを実行することはできません。qalter、qhold、および qrls コマンドを使用して、待機を設定したり解除したりすることもできます。

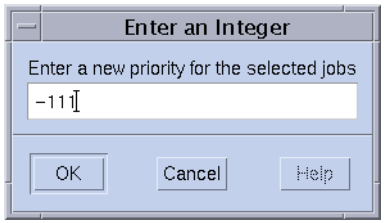
配列ジョブのタスクの待機

「Set Hold」ダイアログボックスの「Tasks」フィールドは、配列ジョブに適用されます。配列ジョブの特定のサブタスクを待機状態にするには、このボタンを使用します。「Tasks」フィールドのテキスト書式に注意してください。このフィールドで指

定するタスク ID 範囲は、数字 1 つの場合、*n-m* という書式の単純な範囲の場合、または指定された間隔で数字を使用する範囲の場合があります。たとえば、`2-10:2` と指定されたタスク ID 範囲には、タスク ID インデックス 2、4、6、8 および 10 が含まれます。この範囲は 5 つの同質のタスクの合計を表し、環境変数 `SGE_TASK_ID` には 5 つのインデックス番号のいずれかが含まれます。ジョブの待機の詳細は、`qsub`(1) のマニュアルページを参照してください。

ジョブの優先順位の変更

「Job Control」ダイアログボックスの「Priority」をクリックすると、次のダイアログボックスが表示されます。



このダイアログボックスでは、選択した保留中または実行中のジョブの新しい優先順位を入力できます。優先順位は、1 人のユーザーのジョブ間でのジョブの順番です。優先順位によって、同時に複数のジョブがシステム内に存在する場合に、どのようにして 1 人のユーザーのジョブから選択を行うかがスケジューラに伝えられます。

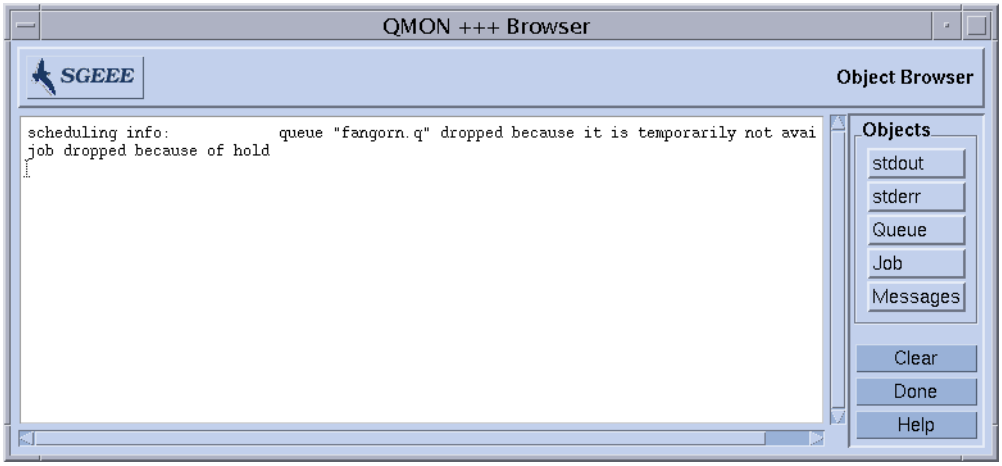
保留中のジョブを選択して「Qalter」をクリックすると、「Submit Job」ウィンドウが表示されます。ダイアログボックスのすべてのエントリは、ジョブの発行時に定義されたジョブの属性に従って設定されます。変更できないエントリはグレー表示されます。その他のエントリは編集できます。変更は、「Submit Job」ダイアログボックスの「Qalter」をクリックすると、Grid Engine システムに登録されます。「Qalter」ボタンは「Submit」ボタンの代わりとして使用できます。

ジョブの一貫性の確認

「Submit Job」ダイアログボックスの「Verify」フラグは、Qalter モードで使用される場合、特別な意味を持ちます。保留中のジョブの一貫性をチェックしたり、ジョブがスケジューリングされていない理由を調べることができます。「Verify」フラグに対して設定する一貫性チェックモードを選択したあと、「Qalter」をクリックしてください。選択したチェックモードによって、一貫性のない場合は警告が表示されます。詳細は、64 ページの「`QMON` による高度なジョブの発行」および `qalter`(1) のマニュアルページの `-w` オプションを参照してください。

「Why?」ボタンを使用した保留中のジョブ情報の取得

ジョブが保留中である理由をチェックするもう 1 つの方法は、ジョブを選択して「Job Control」ダイアログボックスの「Why?」をクリックすることです。これにより、「Object Browser」ダイアログボックスが開きます。このダイアログボックスには、スケジューラが最新のパス内のジョブの割り振りを行わない理由のリストが表示されます。このようなメッセージが表示された「Browser」ウィンドウの例を次の図に示します。



「Why?」 ボタンが意味のある出力を行うのは、スケジューラ構成パラメータ `schedd_job_info` が `true` に設定されている場合だけです。`sched_conf`(5) のマニュアルページを参照してください。表示されたスケジューラ情報は、最後のスケジューリング間隔に関するものです。この情報は、ジョブがスケジューリングされなかった理由を調べるまで正しくない可能性があります。

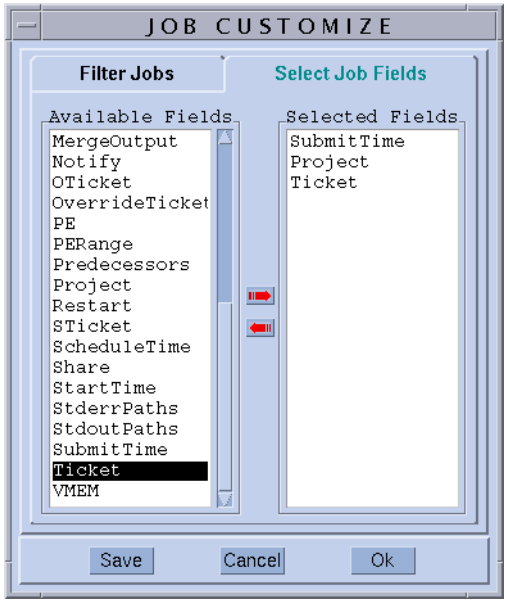
エラー状態のクリア

ジョブに依存する問題が原因で失敗した保留中のジョブのエラー状態を削除するには、「Clear Error」をクリックします。たとえば、ジョブが指定されたジョブ出力ファイルへの十分な書き込み権を持ってない場合などにエラーが発生します。

エラー状態は、保留中のジョブリスト内に赤いフォントで表示されます。ジョブは、たとえば `qalter` などでエラー状態を修正したあとに削除してください。ジョブの中止時には電子メールを送信するようにジョブが要求している場合、前記のようなエラー状態は電子メールで自動的に報告されます。たとえば、`qsub -m a` コマンドなどでジョブが中止されている場合があります。

「Job Control」ディスプレイのカスタマイズ

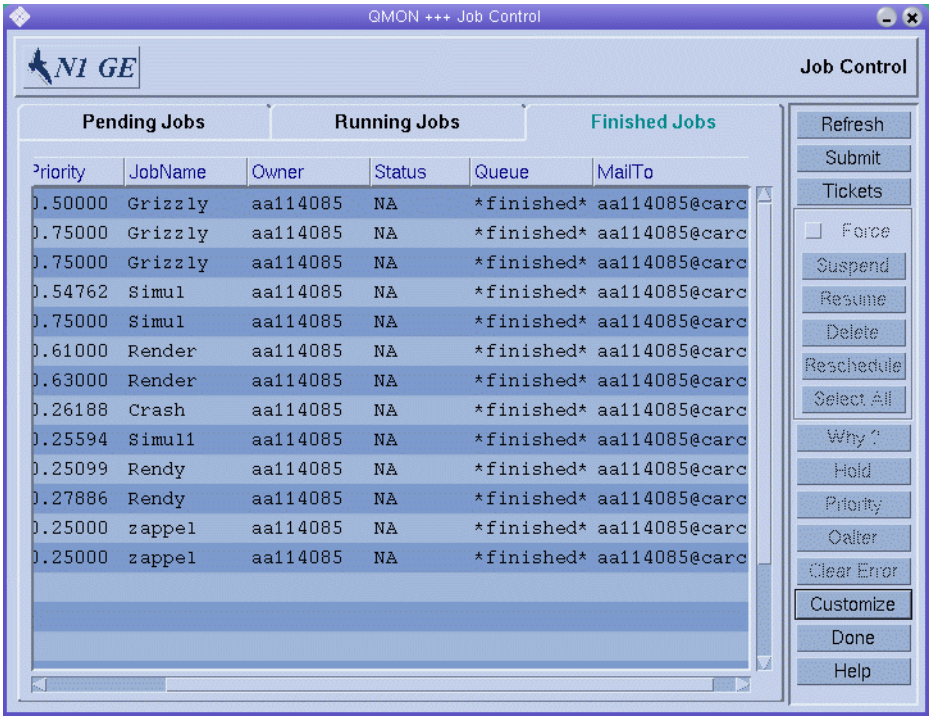
デフォルトの「Job Control」ディスプレイをカスタマイズするには、「Customize」をクリックします。「Job Customize」ダイアログボックスが表示されます。「Select Job Fields」タブをクリックします。次の図は、「Select Job Fields」タブの表示例です。



「Job Customize」ダイアログボックスを使用して、表示する情報を設定します。

「Job Customize」ダイアログボックスでは、ジョブオブジェクトのエントリをより多く選択して、表示できます。また、必要なジョブのフィルタリングも可能です。上の図の例では、「Projects」、「Tickets」、および「Submit Time」という追加フィールドが選択されています。

次の図には、「Finished Jobs」リストにカスタマイズが適用されたあとの拡張された外観を表示しています。



カスタマイズ設定を `.qmon_preferences` ファイルに保存するには、「Customize Job」ダイアログボックスの「Save」ボタンを使用します。このファイルはユーザーのホームディレクトリに保存されています。カスタマイズ設定を保存すると、「Job Control」ダイアログボックスの見た目が再定義されます。

ジョブリストのフィルタリング

次の例のフィルタリング機能では、アーキテクチャー `solaris64` での実行に適した `aa114085` が所有するジョブだけを選択します。

「Job Control」ダイアログボックスの「Running Jobs」タブのフィルタリング結果を次の図に示します。

上の図の「Job Control」ダイアログボックスは、QMON によって配列ジョブがどのように表示されるかの例でもあります。

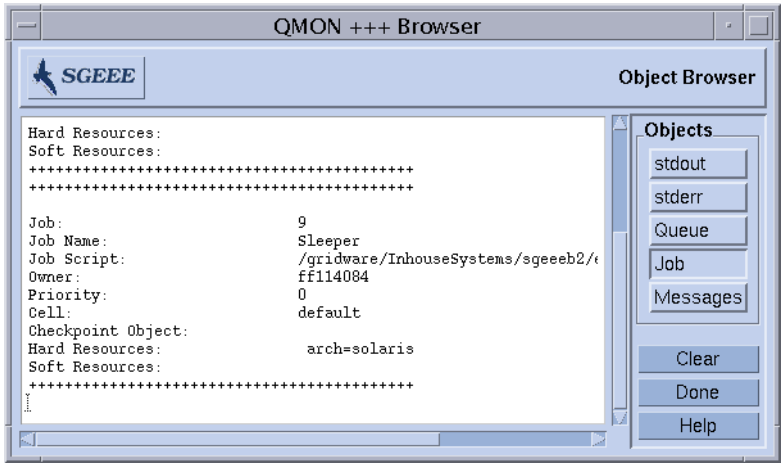
QMON の Object Browser によるジョブの追加情報の取得

89 ページの「QMON を使用したジョブの監視と制御」 に説明されているとおり、QMON Object Browser を使用すると、「Job Control」ダイアログボックスをカスタマイズせずに、ジョブの追加情報をすばやく取り出すことができます。

ジョブに関する情報を表示する Object Browser は、次の 2 つの方法で開けます。

- 「QMON Main Control」ウィンドウの「Browser」ボタンをクリックしたあと、「Browser」ダイアログボックスの「Job」をクリックします。
- 「Job Control」ダイアログボックスのジョブにポインタを移動します。

次の「Browser」ウィンドウに、表示されるジョブ情報の例を示します。



コマンド行からのジョブの監視と制御

この節では、コマンド qstat、 qdel、および qmod によるコマンド行からのジョブの監視、削除、および変更方法について説明します。

qstat によるジョブの監視

ジョブを監視するには、次の節で詳述されている情報を参考に、次のコマンドのいずれかを入力します。

```
qstat
qstat -f
qstat -ext
```

オプションのない qstat では、発行したジョブだけの概要が提供されます。qstat -f では、さらに現在構成されているキューに関する情報も含まれます。qstat -ext を指定すると、ジョブの使用率の最新情報やジョブに割り当てられたチケットなどの詳細も含まれます。

最初の書式では、ヘッダ行が列の意味を示します。ほとんどの列の目的は、それ自体で説明できなければなりません。ただし、「state」列には次の意味の 1 文字のコードが含まれます。実行中は r、一時停止は s 、キュー内は q、待機は w で表されます。qstat の出力形式の詳細は、qstat(1) のマニュアルページを参照してください。

2 番目の書式は 2 つのセクションに分けられます。1 番目のセクションは、すべての使用可能なキューのステータスを表示します。PENDING JOBS というタイトルの 2 番目のセクションは、 sge_qmaster ジョブスプール空間のステータスを表示します。キューセクションの 1 行目では、リストされているキューに対する列の意味が定義されています。キューは水平線で区切られています。キュー内でジョブが実行され

る場合、1 番目の書式の qstat コマンドと同じ書式で、関連するキューの下にそのジョブ名が表示されます。2 番目の出力セクションの保留中のジョブは、qstat の 1 番目の書式と同じように表示されます。

キュー説明の列は、次の情報を提供します。

- qtype – キューのタイプ。キューのタイプは B (バッチ) または I (対話型) のどちらかです。
- used/free – キュー内の使用されているジョブスロットと空きジョブスロットの数。
- states – キューの状態。キューの状態については、 qstat(1) のマニュアルページを参照してください。

qstat(1) のマニュアルページにはqstat の出力形式の詳しい説明が記載されています。

3 番目の形式では、使用率およびジョブに割り当てられたチケットの値が次の列に表示されます。

- cpu/mem/io – 現在蓄積されている CPU、メモリー、および I/O の使用量。
- tckts/ovrts/otckt/ftckt/stckt – これらの値は次のとおりです。

  tckts – ジョブに割り当てられたチケットの合計

  ovrts – qalter -ot によって割り当てられた優先チケット

  otckt – 優先ポリシーによって割り当てられたチケット

  ftckt – 機能ポリシーによって割り当てられたチケット

  stckt – 共有ベースポリシーによって割り当てられたチケット

さらに、期限の開始時が適宜「deadline」列に表示されます。「share」列には、クラスタ内のすべてのジョブによって発生する使用量を基準にした各ジョブの現在のリソース配分が示されます。詳細は、qstat(1) のマニュアルページを参照してください。

qstat コマンドのさまざまな追加オプションは、機能性を拡張します。発行したジョブのリソース要件を表示するには、-r オプションを使用します。さらに、出力を特定のユーザーまたはキューに限定することもできます。69 ページの「リソース要件の定義」で説明されているとおり、-l オプションを使用して qsub コマンドのリソース要件を指定できます。リソース要件を使用すると、qstat によって指定されたリソース要件に一致するキューおよびそれらのキューで実行されているジョブだけが表示されます。

注 – qstat コマンドは、管理者およびユーザーが有用なオプションを含めたファイルを定義できるように拡張されました。sge_qstat(5) のマニュアルページを参照してください。クラスタ全体の sge_qstat ファイルは、$xxQS_NAME_Sxx_ROOT/$xxQS_NAME_Sxx_CELL/common/sge_qstat に置くことができます。ユーザー専用のファイルは、$HOME/.sge_qstat で処理されます。ホームディレクトリ要求ファイルの優先順位がもっとも高く、次がクラスタグローバルファイルです。コマンド行を使用すると、ファイルに指定したフラグよりも優先されます。

例 4–1 と 例 4–2 に、qstat とqstat -f コマンドの出力例を示します。

例 4–1 qstat -f の出力例

```
queuename                qtype  used/free   load_avg   arch         states

dq                       BIP    0/1         99.99      sun4         au

durin.q                  BIP    2/2         0.36       sun4
  231     0    hydra             craig       r          07/13/96     20:27:15    MASTER
  232     0    compile           penny       r          07/13/96     20:30:40    MASTER

dwain.q                  BIP    3/3         0.36       sun4
  230     0    blackhole         don         r          07/13/96     20:26:10    MASTER
  233     0    mac               elaine      r          07/13/96     20:30:40    MASTER
  234     0    golf              shannon     r          07/13/96     20:31:44    MASTER

fq                       BIP    0/3         0.36       sun4

############################################################################

- PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS -

############################################################################

  236     5    word              elaine      qw         07/13/96     20:32:07

  235     0    andrun            penny       qw         07/13/96     20:31:43
```

例 4–2 qstat の出力例

```
job-ID   prior   name         user      state   submit/start at     queue      function
231      0       hydra        craig     r       07/13/96            durin.q    MASTER
                                                20:27:15
232      0       compile      penny     r       07/13/96            durin.q    MASTER
                                                20:30:40
230      0       blackhole    don       r       07/13/96            dwain.q    MASTER
```

例 4–2 qstat の出力例　(続き)

```
                                        20:26:10
233      0       mac          elaine    r       07/13/96            dwain.q    MASTER
                                        20:30:40
234      0       golf         shannon   r       07/13/96            dwain.q    MASTER
                                        20:31:44
236      5       word         elaine    qw      07/13/96
                                        20:32:07
235      0       andrun       penny     qw      07/13/96 20:31:43
```

qdel および qmod によるジョブの制御

コマンド行からジョブを制御するには、次のいずれかのコマンドに適切な引数を指定して実行します。

```
% qdel arguments
% qmod arguments
```

ジョブが実行中なのかスプール状態なのかに関わらず、ジョブを取り消すには qdel コマンドを使用します。すでに実行中のジョブを一時停止および再開 (一時停止を解除) するには、qmod コマンドを使用します。

両方のコマンドでジョブ識別番号が必要です。ジョブ識別番号は、qsub コマンドを正常に実行すると返されます。番号を忘れた場合は、qstat で取り出すことができます。99 ページの「qstat によるジョブの監視」を参照してください。

次のリストに、qdel および qmod コマンドの例をいくつか示します。

```
% qdel job-id
% qdel -f job-id1, job-id2
% qmod -s job-id
% qmod -us -f job-id1, job-id2
% qmod -s job-id.task-id-range
```

ジョブを削除、一時停止、または再開するには、ジョブの所有者、Grid Engine 管理者、またはオペレータでなければなりません。38 ページの「管理者、オペレータ、および所有者」を参照してください。

どちらのコマンドでも、-f (force) オプションを使用すると、sge_execd に影響することなくジョブステータスの変更を sge_qmaster に登録することができます。強制実行オプションは、たとえばネットワーク問題などが原因で sge_execd を実行できない場合などに使用してください。 -f オプションは、管理者だけが使用できます。ただし、 qdel でクラスタ構成 qmaster_params エントリの ENABLE_FORCED_QDEL フラグが設定されている場合は、自分自身のジョブは強制削除することができます。詳細は、sge_conf(5) のマニュアルページを参照してください。

電子メールによるジョブの監視

コマンド行から、適切な引数を指定して次のコマンドを入力します。

```
% qsub arguments
```

qsub -m コマンドは、特定のイベントが発生するとジョブを発行したユーザーまたは -M フラグで指定されたアドレスに電子メールを送信するように要求します。フラグの説明については、qsub(1) のマニュアルページを参照してください。-m オプションの引数は、イベントを指定します。次の 引数が使用可能です。

- b – ジョブの開始時に電子メールを送信します。
- e – ジョブの終了時に電子メールを送信します。
- a – ジョブが再スケジューリングされた、または qdel コマンドなどによって中止されたときに、電子メールを送信します。
- s – ジョブの一時停止時に電子メールを送信します。
- n – 電子メールを送信しません。デフォルトは n になります。

-m オプションでこれらのオプションのいくつかを指定するには、1 つ以上の文字引数で構成された文字列を使用してください。たとえば、 -m be と指定すると、電子メールはジョブの開始時と終了時に送信されます。

「Submit Job」ダイアログボックスを使用して、これらのメールイベントを設定することもできます。64 ページの「QMON による高度なジョブの発行」を参照してください。

キューの監視と制御

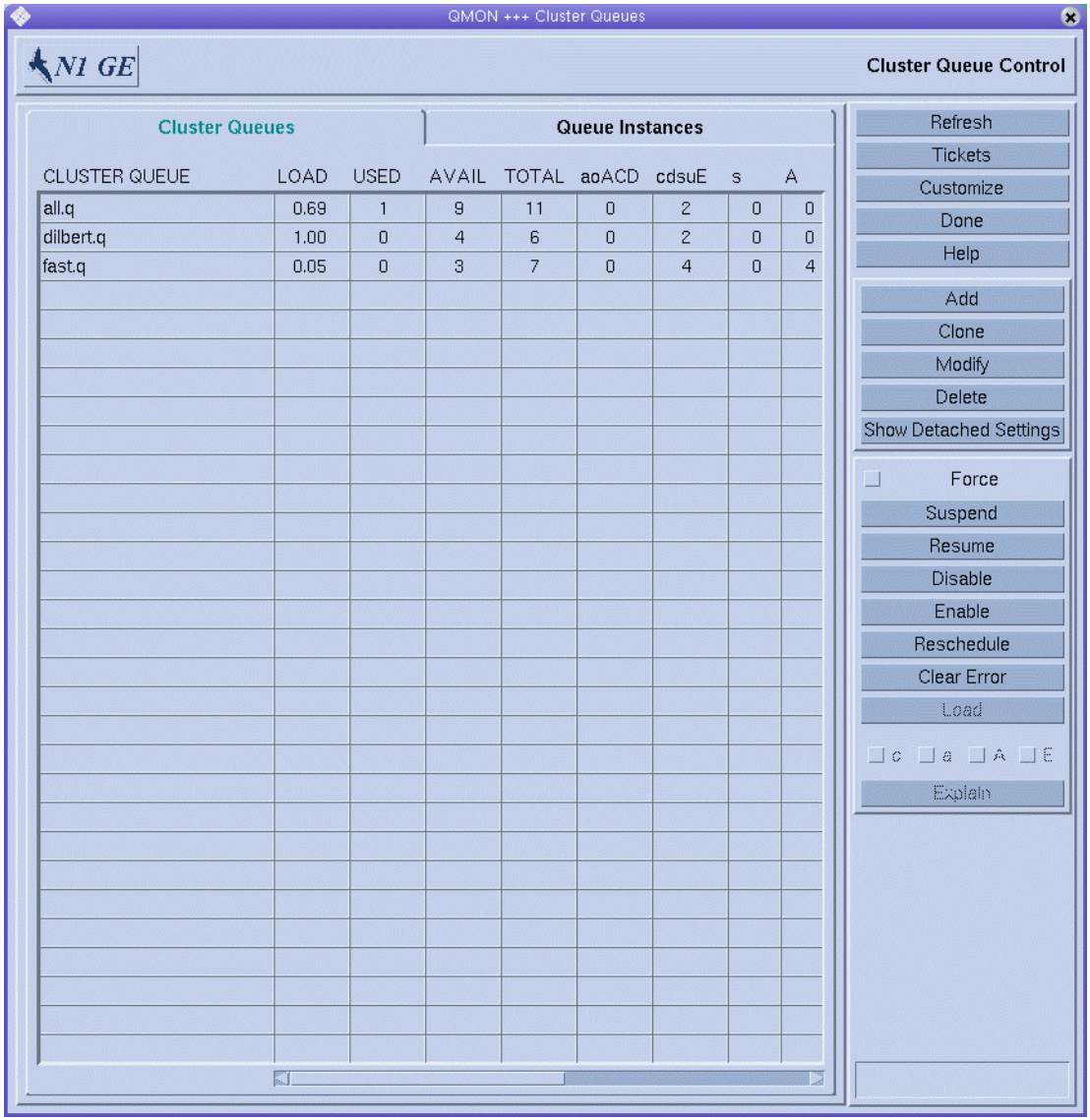
38 ページの「キューとキュープロパティーの表示」の説明どおり、キューの所有者はキューの一時停止や再開および無効化や有効化を行う権限を持っています。所有者は、重要な作業に特定のマシンが必要で、これらのマシンが背景で実行されているジョブの影響を強く受ける場合に、キューを一時停止したり無効にしたりする場合があります。

キューは 2 つの方法で制御できます。

- 「QMON Queue Control」ダイアログボックスの使用
- qmod コマンドの使用

QMON によるキューの監視と制御

「QMON Main Control」ウィンドウで「Queue Control」ボタンをクリックします。「Cluster Queues」ダイアログボックスが表示されます。



クラスタキューの監視と制御

「Cluster Queue」タブには、クラスタに対して定義したすべてのクラスタキューの簡単な概要が示されます。また「Cluster Queue」タブでは、クラスタキューを一時停止および再開させたり、クラスタキューを有効または無効にしたり、クラスタキューを構成したりすることもできます。

「Cluster Queue」ダイアログボックスに表示される情報は、定期的に更新されます。更新を強制的に行うには、「Refresh」をクリックしてください。クラスタキュー名をクリックして、キューを選択します。

「Delete」、「Suspend」、「Resume」、「Disable」、または「Enable」をクリックして、対応する操作を選択したクラスタキューに対して実行します。一時停止/再開および無効化/有効化操作では、対応する `sge_execd` へ通知を行う必要があります。通知できない場合は、「Force」をクリックして、`sge_qmaster` 内部ステータスを強制的に変更できます。たとえば、ホストが休止している場合は通知を行えません。

一時停止/再開および無効化/有効化操作は、クラスタキュー所有者権限、Grid Engine 管理者権限またはオペレータ権限が必要です。詳細は、38 ページの「管理者、オペレータ、および所有者」を参照してください。

一時停止したクラスタキューにさらにジョブを追加することはできません。89 ページの「QMON を使用したジョブの監視と制御」の説明どおり、一時停止したキュー内ですでに実行されていたジョブも一時停止されます。キューが再開されるとすぐ、クラスタキューとジョブの一時停止も解除されます。

注 – 一時停止したクラスタキュー内のジョブが明示的に一時停止されている場合は、キューが再開されても、そのジョブは再開されません。明示的に一時停止したジョブは、明示的に再開する必要があります。

無効にされたクラスタキューは閉じられています。ただし、これらのキュー内の実行中のジョブは続行できます。クラスタキューは通常、キューを「排出」するために無効にされます。クラスタキューを有効にしたあとは、ジョブを再度実行できます。現在実行中のジョブに対する処理は行われません。

エラー状態は、キューリスト内に赤いフォントを使用して表示されます。キューからエラー状態を削除するには、「Clear Error」をクリックします。

選択されたクラスタキュー内の現在実行中のすべてのジョブを再スケジューリングするには、「Reschedule」をクリックします。

クラスタキューとキューインスタンスを構成するには、「Cluster Queue」ダイアログボックスの「Add」または「Modify」をクリックします。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「QMON を使用したキューの構成」を参照してください。

ダイアログボックスを閉じるには、「Done」をクリックしてください。

クラスタキューのステータス

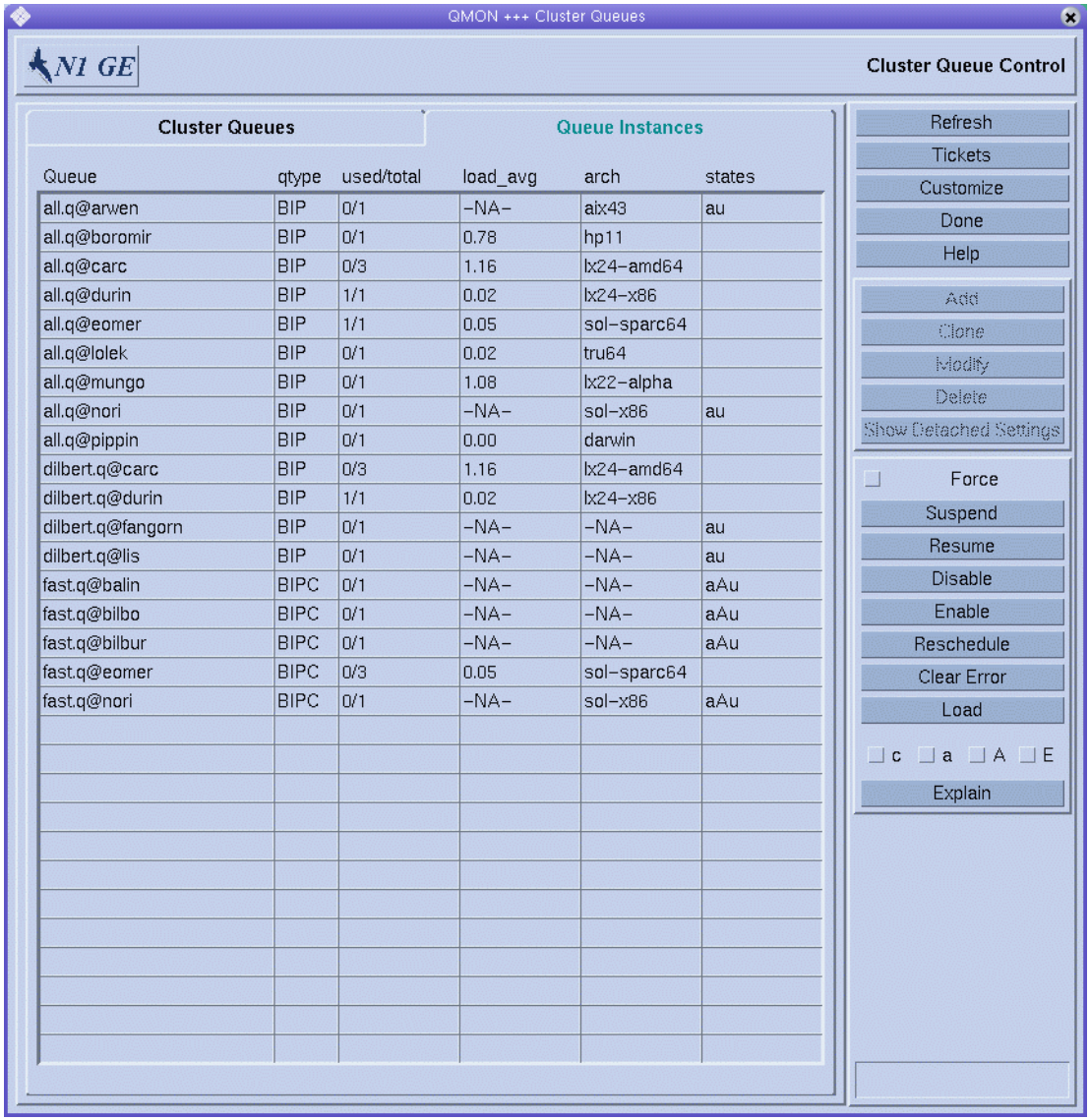
クラスタキューテーブルの各行は、1 つのクラスタキューを表します。各クラスタキューについて、テーブルは次の情報を一覧表示します。

- Cluster Queue – クラスタキューの名前。
- Load – すべてのクラスタキューホストの標準負荷平均の平均。負荷値を持つホストだけが考慮されます。
- Used – 現在使用済みのジョブスロットの数。
- Avail – 現在使用可能なジョブスロットの数。
- Total – ジョブスロットの合計数。
- aoACD – 次の状態のうち 1 つ以上の状態を持つキューインスタンスの数。
  - a – 負荷しきい値アラーム
  - o – 親のない状態
  - A – 一時停止しきい値アラーム
  - C – カレンダにより一時停止
  - D – カレンダにより無効化
- cdsuE – 次の状態のうち 1 つ以上の状態を持つキューインスタンスの数。
  - c – あいまいな構成
  - d – 無効化
  - s – 一時停止
  - u – 不明
  - E – エラー
- s – 一時停止状態のキューインスタンスの数。
- A – 1 つ以上の一時停止しきい値を現在超過しているキューインスタンスの数。ジョブはこれ以上追加できません。
- S – 別のキューとの従属関係によって一時停止されているキューインスタンスの数。
- C – Grid Engine システムのカレンダによって自動的に一時停止されているキューインスタンスの数。
- u – 状態が不明なキューインスタンスの数。
- a – 1 つ以上の負荷しきい値を現在超過しているキューインスタンスの数。
- d – 無効状態のキューインスタンスの数。
- Ｄ – Grid Engine システムのカレンダによって自動的に無効になったキューインスタンスの数。
- c – 構成があいまいなキューインスタンスの数。
- o – 親がない状態のキューインスタンスの数。
- E – エラー状態のキューインスタンスの数。

クラスタキューおよびそれらの状態については、qstat(1) のマニュアルページを参照してください。

キューインスタンスの監視と制御

「Queue Instances」タブでは、選択したクラスタキューに関連するすべてのキューインスタンスの簡単な概要が示されます。また「Queue Instance」タブでは、キューインスタンスを一時停止、再開、無効化、および有効化することもできます。



Queue	qtype	used/total	load_avg	arch	states
all.q@arwen	BIP	0/1	-NA-	aix43	au
all.q@boromir	BIP	0/1	0.78	hp11	
all.q@carc	BIP	0/3	1.16	lx24-amd64	
all.q@durin	BIP	1/1	0.02	lx24-x86	
all.q@eomer	BIP	1/1	0.05	sol-sparc64	
all.q@lolek	BIP	0/1	0.02	tru64	
all.q@mungo	BIP	0/1	1.08	lx22-alpha	
all.q@nori	BIP	0/1	-NA-	sol-x86	au
all.q@pippin	BIP	0/1	0.00	darwin	
dilbert.q@carc	BIP	0/3	1.16	lx24-amd64	
dilbert.q@durin	BIP	1/1	0.02	lx24-x86	
dilbert.q@fangorn	BIP	0/1	-NA-	-NA-	au
dilbert.q@lis	BIP	0/1	-NA-	-NA-	au
fast.q@balin	BIPC	0/1	-NA-	-NA-	aAu
fast.q@bilbo	BIPC	0/1	-NA-	-NA-	aAu
fast.q@bilbur	BIPC	0/1	-NA-	-NA-	aAu
fast.q@eomer	BIPC	0/3	0.05	sol-sparc64	
fast.q@nori	BIPC	0/1	-NA-	sol-x86	aAu

クラスタキュー名をクリックして、キューインスタンスを選択します。

「Suspend」、「Resume」、「Disable」、または「Enable」をクリックして、対応する操作を選択したキューインスタンスに対して実行します。一時停止/再開および無効化/有効化操作では、対応する `sge_execd` へ通知を行う必要があります。たとえばホストに到達不可能で、通知ができない場合は、「Force」をクリックして `sge_qmaster` 内部ステータスを強制的に変更できます。

一時停止/再開および無効化/有効化操作は、キュー所有者権限、Grid Engine 管理者権限またはオペレータ権限が必要です。38 ページの「管理者、オペレータ、および所有者」を参照してください。

一時停止したキューインスタンスにさらにジョブを追加することはできません。89 ページの「QMON を使用したジョブの監視と制御」の説明どおり、一時停止したキューインスタンス内ですでに実行中となっていたジョブも一時停止されます。キューインスタンスが再開されるとすぐ、キューインスタンスとジョブの一時停止も解除されます。

注 – 一時停止したキューインスタンス内のジョブが明示的に一時停止されている場合は、キューインスタンスが再開されても、そのジョブは再開されません。明示的に一時停止したジョブは、明示的に再開する必要があります。

無効になったキューインスタンスは閉じられます。ただし、キューインスタンス内の実行中のジョブは続行できます。キューインスタンスは通常、キューインスタンスを「排出する」ために無効にされます。キューインスタンスを有効にしたあとは、ジョブを再実行できます。現在実行中のジョブに対する処理は行われません。

キューインスタンスのステータス

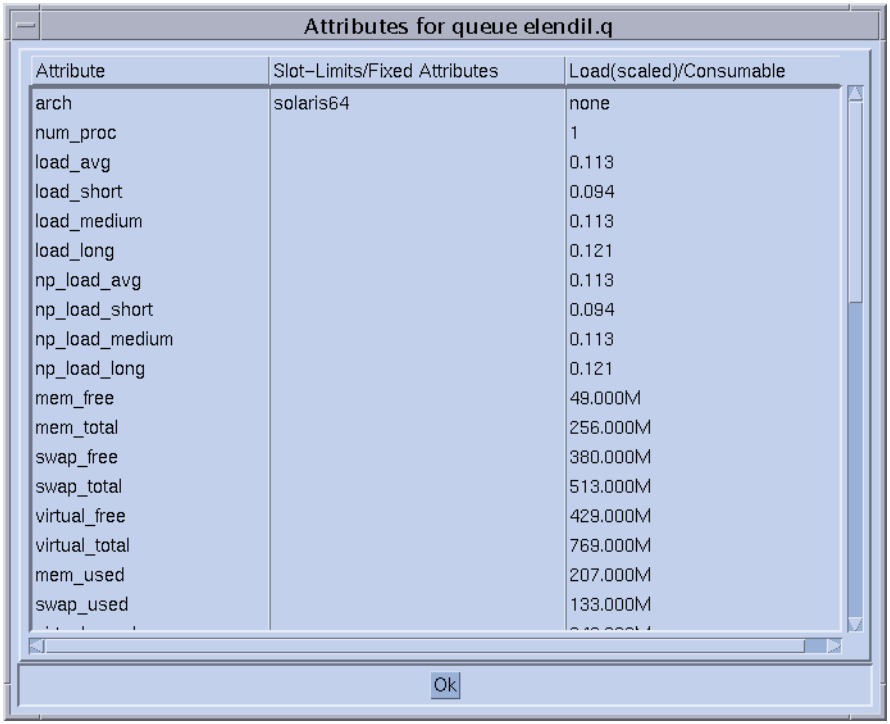
キューインスタンステーブルの各行は、1 つのキューインスタンスを表します。各キューインスタンスについて、テーブルは次の情報を一覧表示します。

- Queue – キューインスタンスの名前
- qtype – キューインスタンスのタイプ。B (バッチ)、I (対話型)、または P (並行処理) のいずれかです。
- used/total – 使用済みジョブスロットの数とジョブスロットの総数
- load_avg – キューインスタンスホストの負荷平均
- arch – キューインスタンスホストのアーキテクチャー
- states – キューインスタンスの状態

キュー状態のリストについては、106 ページの「クラスタキューのステータス」を参照してください。キューインスタンスおよびそれらの状態については、qstat(1) のマニュアルページを参照してください。

キューインスタンス属性の表示

キューインスタンスの現在の属性情報、負荷情報、およびリソース消費情報を取り出すには、キューインスタンスを選択して「Load」をクリックします。この情報には、キューインスタンスのホストとなっているマシンの情報も暗黙に含まれています。次の図のウィンドウが表示されます。



「Attribute」列では、ホストまたはグローバルクラスタから継承される属性を含む、キューインスタンスに付けられるすべての属性が一覧表示されます。

「Slot-Limits/Fixed Attributes」列では、キューインスタンススロット制限単位または固定リソース属性として定義される属性の値が表示されます。

「Load(scaled)/Consumable」列では、報告された負荷パラメータや拡張された負荷パラメータに関する情報が表示されます。この列ではまた、消費可能なリソース機能を基準にした使用可能なリソース機能に関する情報も表示されます。『Sun N1 Grid Engine 6.1 管理ガイド』の「負荷パラメータ」 と『Sun N1 Grid Engine 6.1 管理ガイド』の「消費可能リソース」を参照してください。

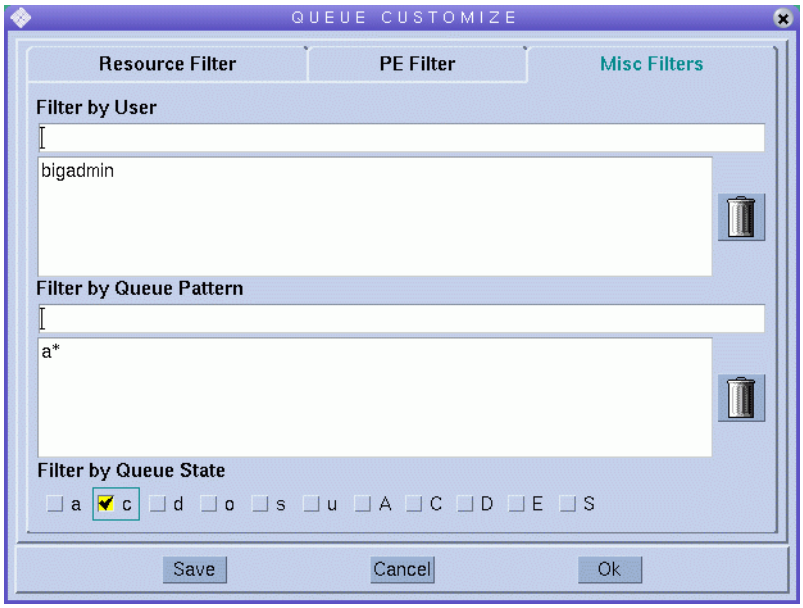
負荷属性が消費可能なリソースとして構成されている場合、負荷レポートと消費可能な容量は、互いに無効にしあう恐れがあります。ジョブ割り振りアルゴリズムで使用される両方の最小値が表示されます。

注 – 表示されている負荷および消費可能な値では現在、27 ページの「実行ホスト」の説明どおり負荷調整の修正は考慮されていません。

クラスタキューおよびキューインスタンスのフィルタリング

「Customize」ボタンを使用すると、表示するクラスタキューとキューインスタンスをフィルタリングできます。

次の 図では、現在の構成があいまいなキューインスタンスだけがフィルタリングで選択されています。



「Queue Customize」ダイアログボックスの「Save」をクリックして、ホームディレクトリの `.qmon_preferences` ファイルに設定を保存し、あとで `QMON` を呼び出したときの標準的な再起動で使用できるようにします。

qmod によるキューの制御

`qmod` コマンドを使用して、キューを一時停止および再開することができます。`qmod` を使用して、キューを無効または有効にすることもできます。

次のコマンドは `qmod` の使用方法を表しています。

```
% qmod -s q-name
% qmod -us -f q-name1, q-name2
% qmod -d q-name
% qmod -e q-name1, q-name2, q-name3
```

qmod –s は、キューを一時停止します。qmod –us –f は、2 つのキューを再開 (一時停止を解除) します。qmod –d は、キューを無効にします。qmod –e は、3 つのキューを有効にします。

-f オプションは、たとえばネットワーク問題などが原因で対応する sge_execd を実行できない場合に、ステータスの変更を sge_qmaster に強制的に登録します。

キューの一時停止と再開およびキューの無効化と有効化では、所有者権限、管理者権限またはオペレータ権限が必要です。38 ページの「管理者、オペレータ、および所有者」を参照してください。

注 – qmod コマンドは、crontab または at ジョブで使用できます。

ジョブチェックポイント設定の使用

この節では、ユーザーレベルとカーネルレベルの 2 種類のジョブチェックポイント設定について検討します。

ユーザーレベルのチェックポイント設定

多くのアプリケーションプログラム、特にかなりの CPU 時間を消費するプログラムでは、フォールトトレランスを高めるためにチェックポイント設定および再起動メカニズムが使用されます。処理データのステータス情報と重要部分は、アルゴリズムのある段階で 1 つ以上のファイルに繰り返し書き込まれます。アプリケーションが中止された場合、これらの再開ファイルを処理してあとで再起動することができます。ファイルは、チェックポイントの直前の状況に相当する一貫性のある状態に戻ります。ほとんどの場合、ユーザーは再開ファイルを適切な場所に移動する必要があるので、この種のチェックポイント設定はユーザーレベルの チェックポイント設定と呼ばれます。

統合されたユーザーレベル チェックポイント設定を持たないアプリケーションプログラムは、チェックポイント設定ライブラリを使用できます。チェックポイント設定ライブラリは、ハードウェアベンダーまたはパブリックドメインによって提供される場合があります。University of Wisconsin の Condor プロジェクトはこの一例です。アプリケーションをこのライブラリと再リンクすることによって、チェックポイント設定メカニズムは、ソースコードを変更せずにアプリケーションにインストールできます。

カーネルレベルのチェックポイント設定

オペレーティングシステムによっては、オペレーティングシステムのカーネルの内部でチェックポイント設定をサポートしているものもあります。この場合、アプリケーションプログラムでの準備やアプリケーションの再リンクは必要ありません。

カーネルレベルのチェックポイント設定は、通常 1 つのプロセスやプロセス階層全体に適用されます。それぞれのプロセスの階層にチェックポイントを設定して、いつでも再起動することができます。通常、チェックポイントの開始にはユーザーコマンドと C ライブラリインタフェースの両方が使用できます。

Grid Engine システムは、オペレーティングシステムによるチェックポイント設定も適宜サポートしています。現在サポート対象のカーネルレベルのチェックポイント設定機能については、N1 Grid Engine 6.1 ソフトウェアのリリースノートを参照してください。

チェックポイント設定ジョブの移行

再起動機能によって実行済みの作業を繰り返す必要はほとんどないので、チェックポイント設定ジョブはいつでも割り込みできます。この機能は、Grid Engine システムで移行および動的負荷均衡メカニズムを構築するために使用されます。要求があれば、チェックポイント設定ジョブはオンデマンドで中止されます。ジョブは Grid Engine システムのほかのマシンに移行されるので、クラスタ内の負荷は動的に平均化されます。チェックポイント設定ジョブが中止され移行される理由は、次のとおりです。

- 実行中のキューまたはジョブが `qmod` または `QMON` コマンドによって明示的に一時停止された。
- キューの一時停止しきい値を超過したので、ジョブまたはジョブが実行されているキューが自動的に一時停止された。ジョブのチェックポイント機会の指定には、一時停止の場合も含まれます。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「負荷および一時停止しきい値の構成」と 114 ページの「コマンド行からのチェックポイント設定ジョブの発行、監視、または削除」を参照してください。

ジョブを移行すると、`sge_qmaster` に戻ります。別の適切なキューを使用できる場合、ジョブは続けてそのキューに割り振られます。この場合、`qstat` の出力は `R` をステータスとして示します。

チェックポイント設定ジョブスクリプトの作成

カーネルレベル チェックポイント設定のシェルスクリプトは、通常のシェルスクリプトと同じです。

ユーザーレベルのチェックポイント設定ジョブのシェルスクリプトは、再起動プロセスを適切に処理できるところだけが通常のバッチスクリプトと異なります。環境変数 `RESTARTED` は、再開されたチェックポイント設定ジョブに対して設定されます。初めて呼び出す場合だけ実行する必要があるジョブ スクリプトのセクションをスキップするために、この変数を使用してください。

例 4–3 に、透過的なチェックポイント設定ジョブのスクリプト例を示します。

例 4–3 チェックポイント設定ジョブスクリプトの例

```
#!/bin/sh
# Grid Engine の /bin/sh を強制実行します。
#$ -S /bin/sh

# 再起動したか移行したかをテストします。
if [ $RESTARTED = 0 ]; then
     # 0 = not restarted
     # Parts to be executed only during the first
     # start go in here
     set_up_grid
fi

# チェックポイント実行可能ファイルの起動
fem
#スクリプトファイルの終了
```

ユーザーレベルのチェックポイント設定ジョブが移行されると、ジョブスクリプトは最初から再開されます。シェルスクリプトのプログラムの流れをジョブが割り込まれた場所に導くのはユーザーの役目です。これにより、複数回実行する必要があるスクリプト内の行がスキップされます。

注 – カーネルレベルのチェックポイント設定ジョブはいつでも割り込みできます。周りのシェルスクリプトは、最後にチェックポイントが設定された正確な場所から再開されます。したがって、RESTARTED 環境変数は、カーネルレベルのチェックポイント設定ジョブには影響しません。

コマンド行からのチェックポイント設定ジョブの発行、監視、または削除

適切なオプションを指定して次のコマンドを入力してください。

```
# qsub options arguments
```

チェックポイント設定ジョブの発行は、qsub -ckpt および qsub -c コマンドを除いて通常のバッチスクリプト同様に行われます。これらのコマンドには、チェックポイント設定メカニズムが必要です。これらのコマンドは、ジョブに対してチェックポイントを設定しなければならない場面も定義します。

-ckpt オプションでは、使用するチェックポイント設定環境の名前を示す引数が使用されます。『Sun N1 Grid Engine 6.1 管理ガイド』の「チェックポイント設定環境の構成」を参照してください。

-c オプションは必要ありません。-c でも 1 つの引数が使用されます。チェックポイント設定環境構成の when パラメータの定義を無効にする場合は、-c オプションを使用してください。詳細は、checkpoint(5) のマニュアルページを参照してください。

-c オプションの引数は、次の 1 文字のいずれか、またはこれらの組み合わせです。引数は時間値の場合もあります。

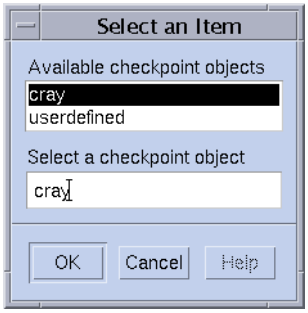
- n – チェックポイントは設定されません。 n は最優先されます。
- s – ジョブホストの sge_execd が停止された場合のみ、チェックポイントを設定します。
- m – 対応するキュー構成で定義される最小 CPU 間隔でチェックポイントが設定されます。queue_conf (5) のマニュアルページの min_cpu_interval パラメータを参照してください。
- x –ジョブが一時停止されると、チェックポイントが設定されます。
- interval – チェックポイントは指定された間隔で設定されますが、この頻度は min_cpu_interval より少なくなります。時間値は、*hh:mm:ss* のように指定してください。この書式では、コロンで区切られた 2 桁の時間、分および秒を指定します。

チェックポイント設定ジョブの監視は、通常のジョブの監視とは異なります。チェックポイント設定ジョブは適宜移行できます。したがって、チェックポイント設定ジョブは 1 つのキューと結合しているわけではありません。ただし、一意のジョブ識別番号とジョブ名は変わりません。

チェックポイント設定ジョブの削除は、99 ページの「コマンド行からのジョブの監視と制御」の説明どおりに行われます。

QMON によるチェックポイント設定ジョブの発行

QMON によるチェックポイント設定ジョブの発行は通常のバッチジョブの発行と同じで、さらに適切なチェックポイント設定環境も指定できます。64 ページの「QMON による高度なジョブの発行」の説明どおり、「Submit Job」ダイアログボックスはジョブと関連付けられるチェックポイント設定環境のフィールドを提供します。フィールド横のボタンをクリックして、次の選択ダイアログボックスを開きます。



使用可能なチェックポイントオブジェクトのリストから適切なチェックポイント環境を選択できます。インストールされているチェックポイント設定環境のプロパティーについては、システム管理者に問い合わせてください。詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「チェックポイント設定環境の構成」を参照してください。

チェックポイント設定のためのファイルシステム要件

ユーザーレベルのチェックポイントまたはチェックポイント設定ライブラリに基づくカーネルレベルのチェックポイントが書き込まれるときには、チェックポイントを設定するプロセスまたはジョブが占める仮想メモリのイメージ全体を保存する必要があります。このためには十分なディスク容量が必要です。チェックポイント設定環境構成パラメータの `ckpt_dir` が設定されている場合、チェックポイント情報は `ckpt_dir` の下のジョブ固有の場所に保存されます。`ckpt_dir` が `NONE` に設定されている場合、チェックポイント設定ジョブが開始されたディレクトリが使用されます。チェックポイント設定環境の構成については、`checkpoint`(5) のマニュアルページを参照してください。

注 – `ckpt_dir` が `NONE` に設定されている場合、チェックポイント設定ジョブは `qsub -cwd` スクリプトで開始する必要があります。

ジョブの移行と再開を正常に行うには、すべてのマシンでチェックポイント設定ファイルと再開ファイルが認識されていなければなりません。ファイルシステムの編成にファイル可視性が必要なため、NFS か類似のファイルシステムである必要があります。ご自分のサイトがこの要件を満たしているかどうか、クラスタ管理者に問い合わせてください。

サイトで NFS を実行していない場合は、シェルスクリプトの最初で再開ファイルを明示的に転送することができます。たとえば、ユーザーレベルのチェックポイント設定ジョブの場合は `rcp` または `ftp` を使用できます。

◆ ◆ ◆ 第 5 章

アカウンティングとレポート

この章の内容は次のとおりです。

アカウンティングおよびレポートコンソールの起動

アカウンティングおよびレポートコンソールは、N1 Grid Engine 6.1 ソフトウェアとは別にインストールされます。インストールプロセスの詳細は、『Sun N1 Grid Engine 6.1 インストールガイド』の第 8 章「アカウンティングおよびレポートコンソールのインストール」を参照してください。さらに、Grid Engine システムでレポート情報を収集できるようにします。レポートデータの集合を有効にする方法の詳細は、『Sun N1 Grid Engine 6.1 管理ガイド』の「レポート統計 (ARCo)」を参照してください。

▼ アカウンティングおよびレポートコンソールを起動する

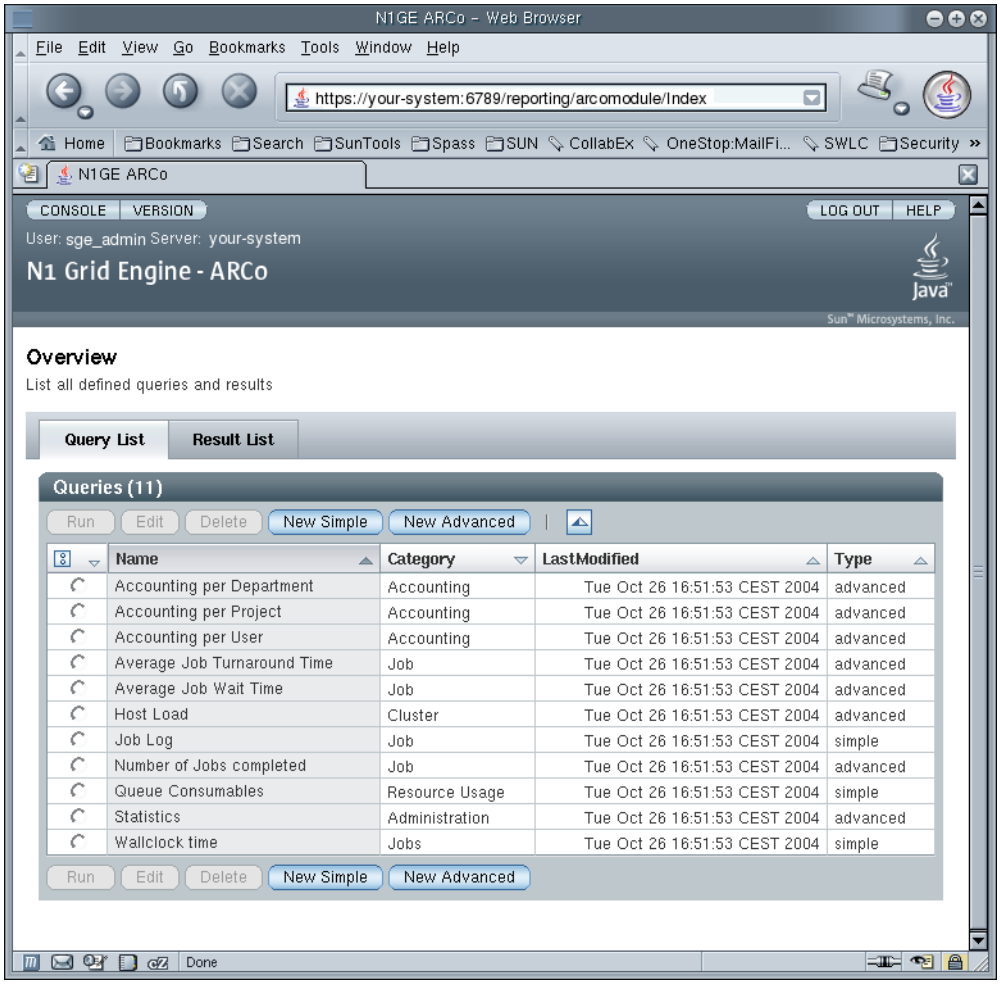
1. **Web** ブラウザを起動します。
2. **URL** を入力して、**Sun Java Web Console** に接続します。

   次の例の *hostname* は、アカウンティングおよびレポートソフトウェアがインストールされたホストです。

   ```
   https://hostname:6789
   ```
3. **UNIX** アカウントにログインします。

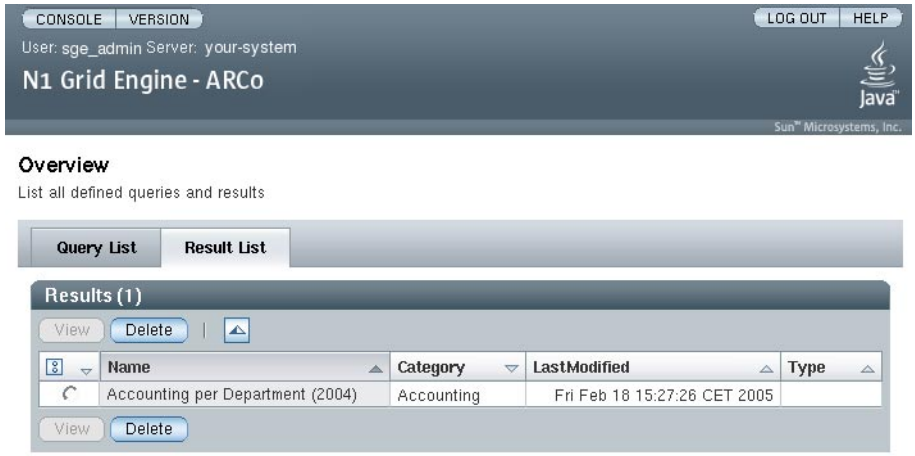
4 **N1 Grid Engine 6 ARCo アプリケーションを選択します。**

「Overview」ページが表示されます。ページには定義済みの ARCo クエリーが一覧表示されています。



ヒント – ARCo アプリケーションへの直接リンクは `https://hostname:6789/console/login/Login?redirect_url=%22/reporting/arcomodule/Index%22)` です。

「Result List」タブを選択すると、格納されているすべてのクエリー結果が表示されます。「Query List」をクリックすると、「Query List Overview」ページに戻ります。



簡単なクエリーの作成と実行

クエリーでは、検索するデータセットを設定します。システムが SQL クエリー文字列を生成し、簡単なクエリーを作成できます。SQL に精通している場合は、自分でクエリーを記述して、高度なクエリーを作成することもできます。

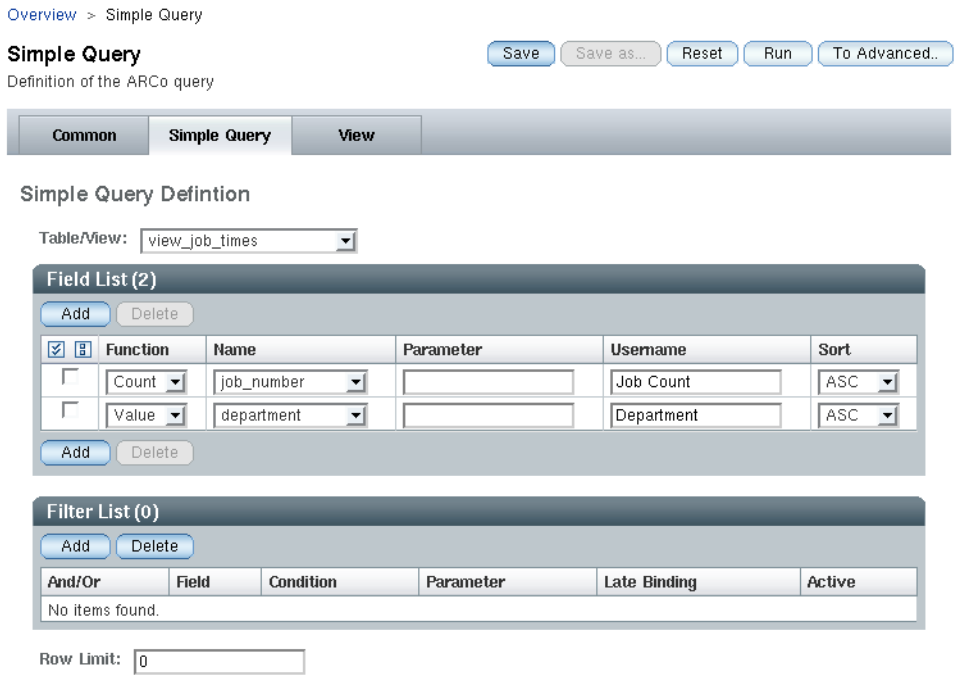
▼ 簡単なクエリーを作成する

1 「**Query List**」ページで「**New Simple**」ボタンをクリックします。

3 つのタブがある次の画面が表示されます。この画面には、クエリーカテゴリや説明などの一般的な情報が表示されます。この情報は省略可能です。クエリーを定義するには、「Simple Query」タブに移動します。クエリーの結果の表示方法を定義するには、「View」タブに移動します。

「Simple Query」タブをクリックして、「Query Definition」ページにアクセスします。このページには、次の機能があります。

- クエリーを事前定義するための、データベーステーブルまたはビューの選択に使用する「Table/View」ドロップダウンメニュー
- すべてのフィールドが 1 行ずつ表示される「Field List」
- クエリーに対してフィルタ条件を定義するための「Filter List」
- クエリーの結果エントリの数を制限するための「Row Limit」フィールド



簡単なクエリーを作成する方法を次の手順で簡潔に示します。

2 テーブルリストからテーブルを選択します。

3 表示するフィールドを定義します。

「Field Function」はフィールドで使用される機能を説明します。「Field Function」でサポートされている値のリストを次に示します。

VALUE　フィールドの現在値を使用します。

SUM　フィールドの値を累積します。

COUNT　フィールドの値の数をカウントします。

MIN　フィールドの最小値を取得します。

MAX　フィールドの最大値を取得します。

AVG　フィールドの平均値を取得します。

- 「Field Name」は、選択されたテーブル内のフィールドです。
- 「User Defined Name」では、より意味のある名前を表示させることができます。
- 「Sort」では、必要に応じて各フィールドのソート順を定義できます。

4 (オプション) フィルタを定義します。

フィルタを定義する前に、1 つ以上のフィールドを指定してください。

- 「AND/OR」は 2 番目以降のフィルタでは必須です。この設定により、前のフィルタ条件と論理的な関連付けを行います。
- 「Field Name」は、フィルタ対象のフィールドの名前です。フィールドがユーザー定義の名前を持つ場合は、その名前が選択リストに表示されます。その他の場合は、生成された名前が表示されます。
- 「Condition 」フィールドでは、データベースからの値のフィルタリングに使用される演算子を指定します。サポートされている演算子を次の表に示します。

フィルタ	シンボル	説明	要件数
Equal	=	値は「Requirement」と同じでなければならない	1
Not Equal	<>、!=	値は「Requirement」と異ならなければならない	1
Less Than	<	値は「Requirement」未満でなければならない	1
Less Than or Equal	<=, ≤	値は「Requirement」以下でなければならない	1
Greater Than	>	値は「Requirement」より大きくなければならない	1
Greater Than or Equal	>=, ≥	値は「Requirement」以上でなければならない	1
Null		値はヌルでなければならない	0
Not Null		値はヌルであってはならない	0
Between		値は指定された値の間になければならない	2
In		値は指定されたリストの要素の 1 つと等しくなければならない	1 以上
Like		値は指定された「Requirement」を含まなくてはならない	1

「Requirement」フィールドには、クエリーの戻り値をフィルタリングするために使用される値が含まれます。次の表に、「Requirement」フィールドに入れられる項目の例を示します。

1 AND 100	between 条件で使用

d%	like 条件で使用
%d%	like 条件で使用
%d%e%	like 条件で使用
Wert-1'、Wert-2'、...、'Wert-*n*	in 条件で使用

5 (オプション) データセットの数を制限します。

データセットの数を制限するには、「Limit Query To First」オプションを選択します。次に、戻り値となるデータセットの数を入力します。

6 「**Save**」をクリックし、クエリーを保存します。

次の図に「Save this Query As」画面を示します。「Query Name」フィールドにクエリーの名前を入力し、「Ok」をクリックします。

クエリーを保存すると、変更された「Simple Query」画面に戻ります。

▼ ビューを作成する

1 クエリーのビューを変更するには、「**View**」タブをクリックします。

保存したクエリーに対するビューを作成するには、次の手順を実行します。

- 「Overview」ページの「Query List」からクエリーを選択します。
- 「編集」ボタンをクリックします。
- 「View」タブをクリックします。

クエリーの現在のビューが表示されます。

2 クエリー結果をどのように表示するかを宣言します。

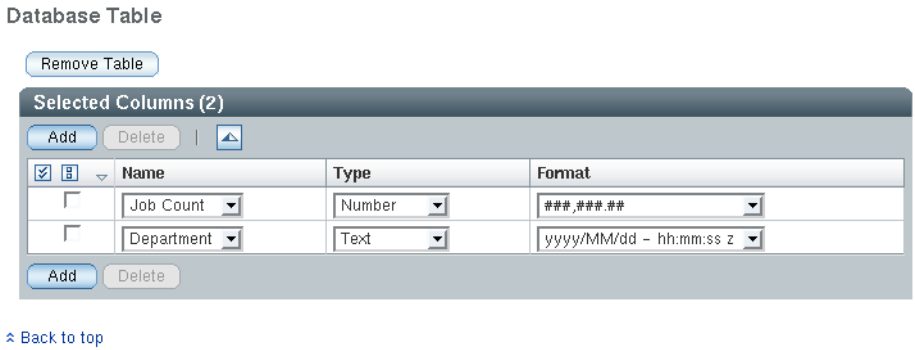
ビュー構成には、3 つの異なるセクションを追加できます。また、クエリーに関する追加情報を表示するかどうかを決定し、どの順番で表示するかも設定できます。

ページ最上部にあるリンクを使用して、対応するセクションに移動できます。ジャンプ可能なセクションには、「Database Table」、「Pivot Table」、および「Graphic」があります。「View Configuration」セクションは常に表示可能で、「Common」タブに入力されたクエリーの説明、フィルタリストに設定されているフィルタ条件、およびクエリー定義で作成した SQL 文や高度なクエリー用の SQL タブの内容を表示するかどうかを切り替えることができます。

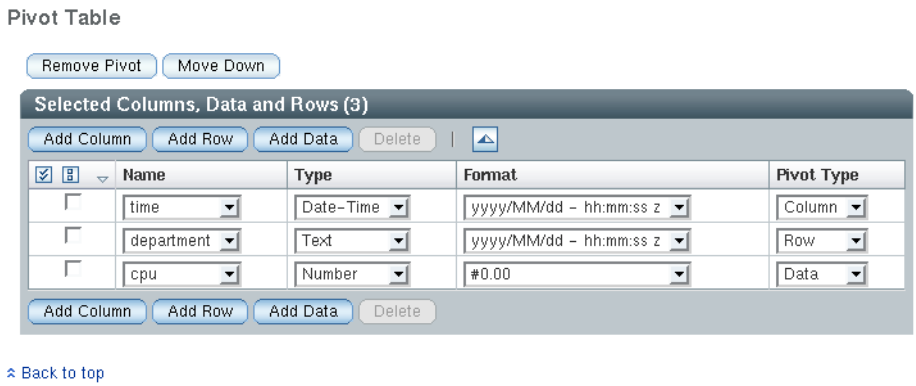
「Add Database」、「Add Pivot」、または「Add Graphic」をクリックすると、対応するセクションが追加されます。

一部のクエリーでは、可能なビュー選択肢のサブセットだけが意味を持ちます。たとえば、選択対象の列が 2 つしかない場合は、ピボットは意味を持ちません。

「Database Table」では、表示する必要のある列を選択して、「Name」の下に追加し、「Type」および「Format」を調整します。列は、追加した順番で表示されます。このレポートで選択した項目は、データに適用されるフィルタには影響されません。



Pivot Table では、ピボット列、行、およびデータエントリを追加します。次に「Name」、「Type」、「Format」などの列を選択します。エントリを別のピボットタイプに移行するには、「Pivot Type」でタイプを選択します。



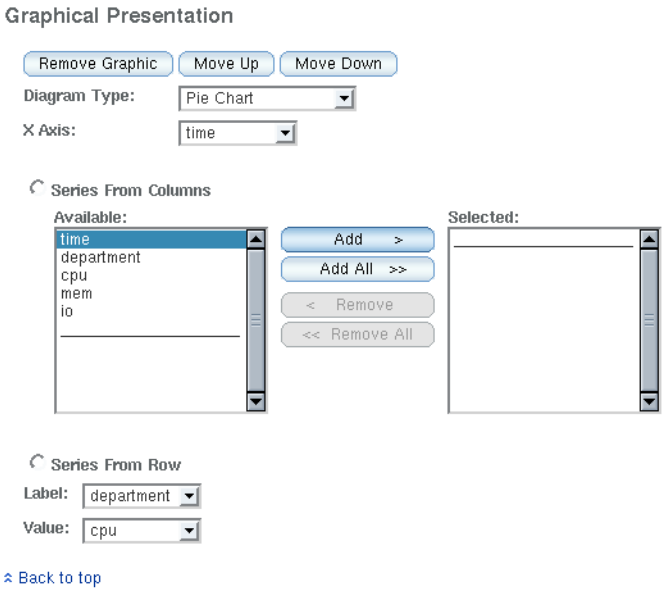
「Graphic」セクションでは、クエリーデータをさまざまなタイプのグラフに貼り付けることができます。「Diagram Type」メニューでは、次のタイプのグラフが使用できます。

- 棒グラフ
- 棒グラフ (3D)
- 積み重ね棒グラフ
- 積み重ね棒グラフ (3D)
- 円グラフ、円グラフ 3D
- 折れ線グラフ
- 積み重ね折れ線グラフ

次の 3 つのタイプのグラフが使用できます。

- 棒グラフ
- 円グラフ
- 折れ線グラフ

棒グラフおよび円グラフは、3D 効果を使用して表示できます。棒グラフおよび折れ線グラフは、y 軸の値が集計された積み重ねグラフとして描画できます。



3 「**Save**」または「**Save As**」をクリックして、クエリーに対するビュー構成を保存します。

4 「**Run**」をクリックして、クエリーを実行します。

図のデータ系列の定義

図のデータ系列を定義する方法は 2 つあります。

- 列からの系列: 列のすべての値が系列に追加されます。系列の名前は列ヘッダーです。
- 行からの系列: 列のすべての値によって系列が定義されます。系列の名前は、ラベル列の値によって定義されます。系列の値は、値列によって定義されます。

例 5–1 部署ごとのアカウンティング円グラフ

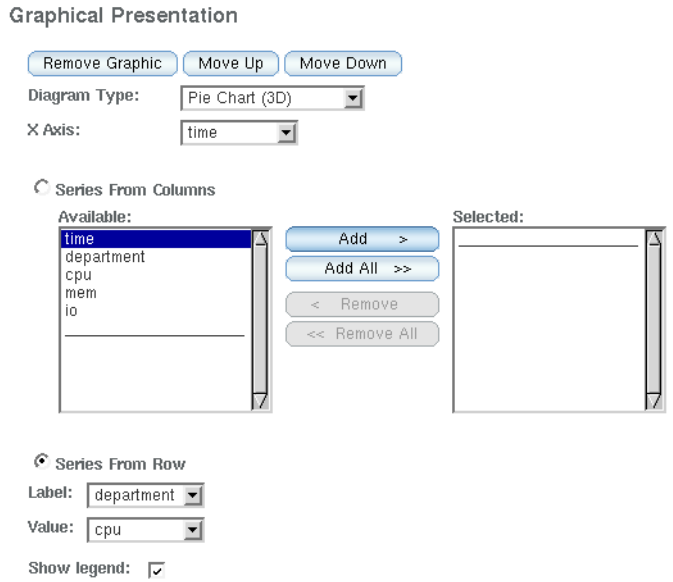
クエリー「部署ごとのアカウンティング」の結果が時間、部署、および CPU の列で構成される次の表に示されています。

Database Table (7)

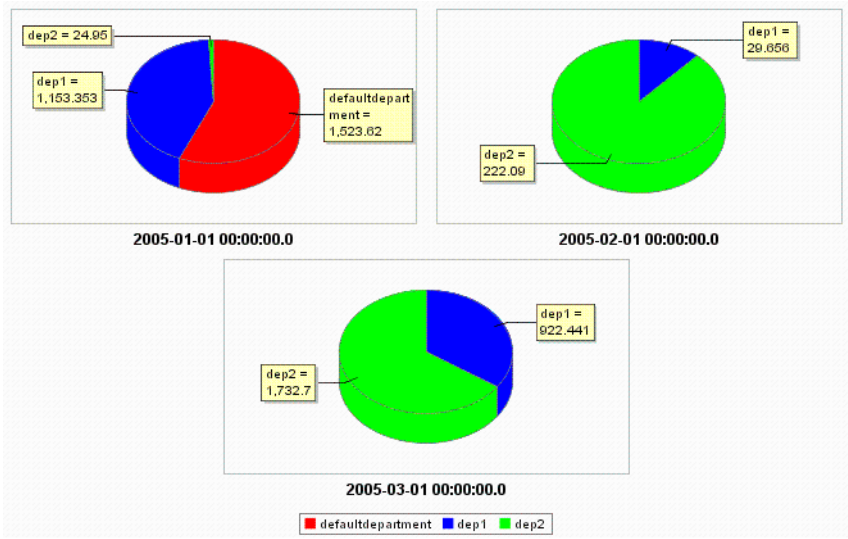
time	department	cpu
2005.01.01	defaultdepartment	1523.62
2005.01.01	dep1	1153.35
2005.01.01	dep2	24.95
2005.02.01	dep1	29.66
2005.02.01	dep2	222.09
2005.03.01	dep1	922.03
2005.03.01	dep2	1732.70

例 5–1　部署ごとのアカウンティング円グラフ　　*(続き)*

結果を円グラフで表示するには、次の構成を選択します。



複数の円グラフで結果が表示されます。



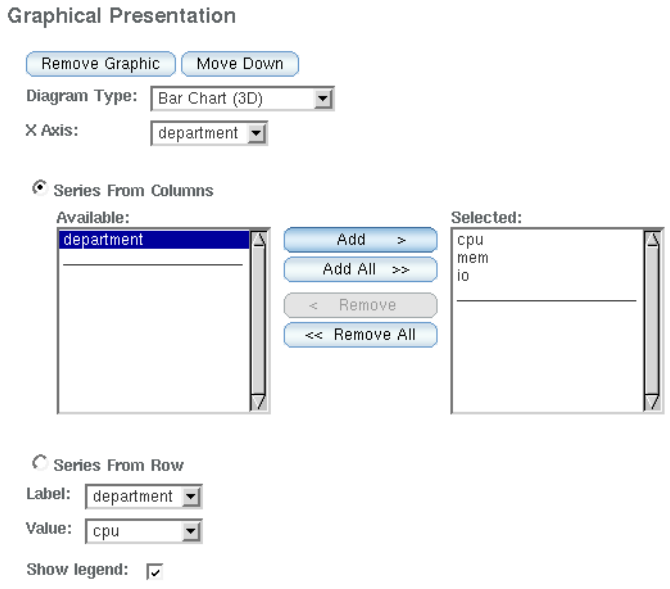
例 5–2　すべての部署に関する CPU、I/O、およびメモリー使用量の棒グラフ

次のクエリーは、すべての部署に関する CPU、I/O、およびメモリー使用量を集計します。

Database Table (3)

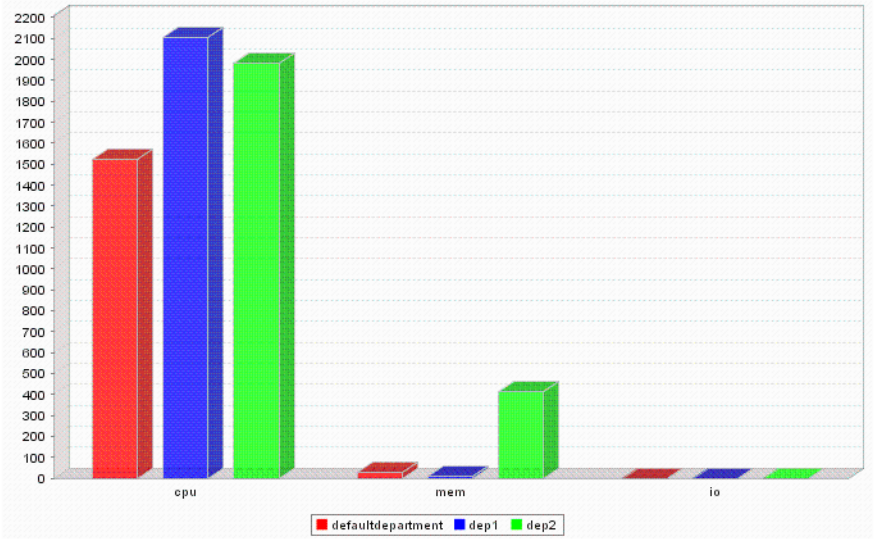
department	cpu	mem	io
defaultdepartment	1523.62	27.00	0.03
dep1	2106.44	8.05	0.07
dep2	1979.74	411.05	0.00

結果を棒グラフで表示するには、次の構成を選択します。



例 5–2　すべての部署に関する CPU、I/O、およびメモリー使用量の棒グラフ　(続き)

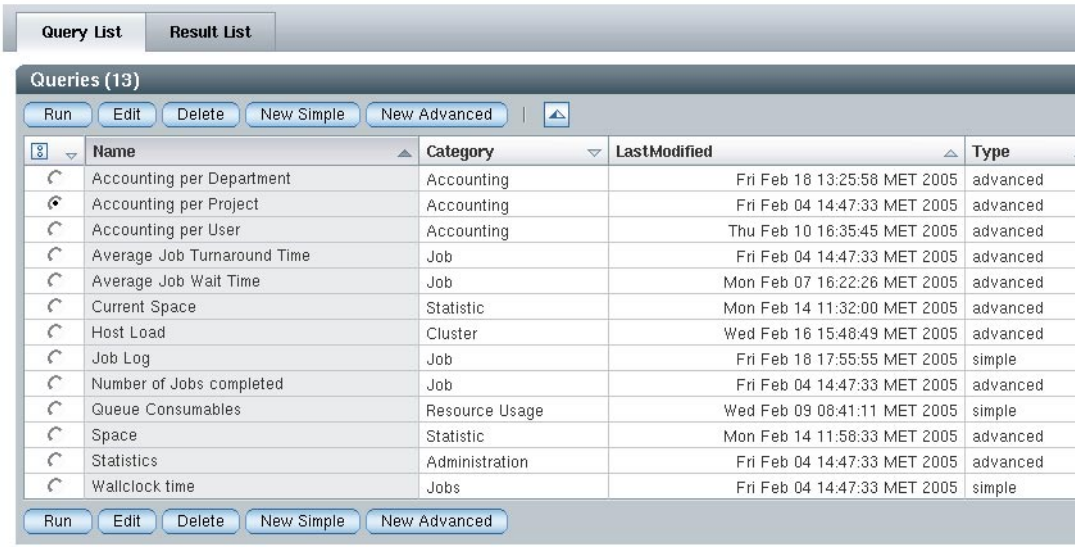
部署ごとに 3 本の棒グラフで結果が表示されます。



▼ 簡単なクエリーを実行する

- クエリーを実行します。
  - 今作成したクエリーを実行するには、「**Simple Query**」画面の「**Run**」をクリックします。

  - 以前に保存したクエリーを実行するには、「**Query List**」画面でクエリーを選択して、「**Run**」をクリックします。



▼ 簡単なクエリーを編集する

1. 「**Query List**」画面のリストからクエリーを選択します。
2. 「**Edit**」をクリックします。
   選択した「Simple Query」画面が表示されます。
3. 簡単なクエリーの作成と同じように、タブ間を移動して変更内容を入力し、「**Simple Query**」画面を変更します。
4. 変更したクエリーを保存または実行します。

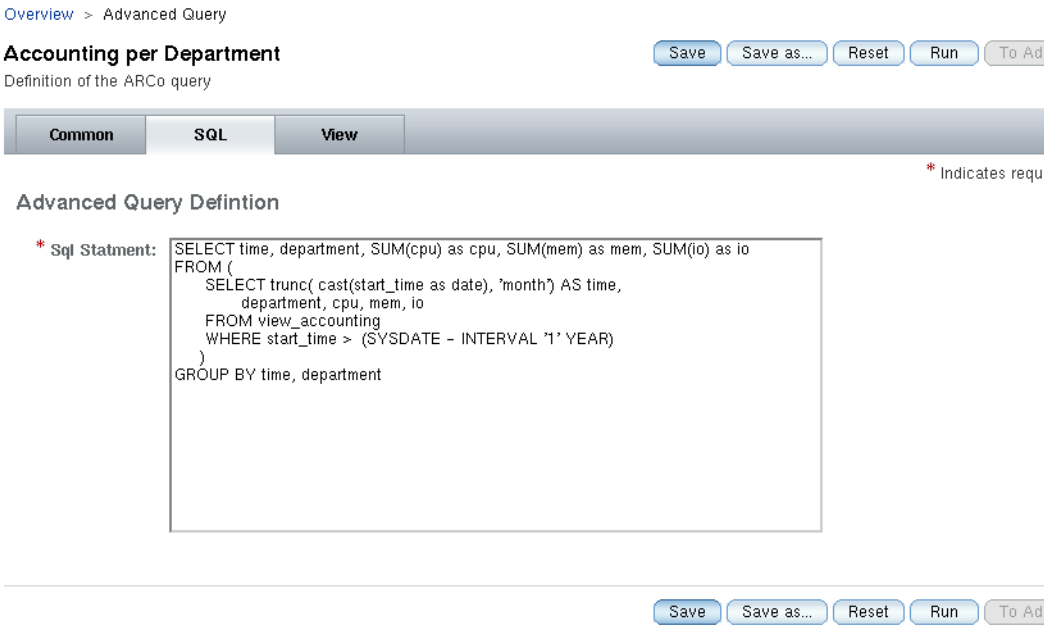
高度なクエリーの作成と実行

アカウンティングおよびレポートコンソールのこの機能を使用するには、SQL クエリーを記述した経験が必要です。

▼ 高度なクエリーを作成する

1 「**Query List**」画面で「**New Advanced Query**」をクリックします。

2 フィールドに **SQL** クエリーを入力します。

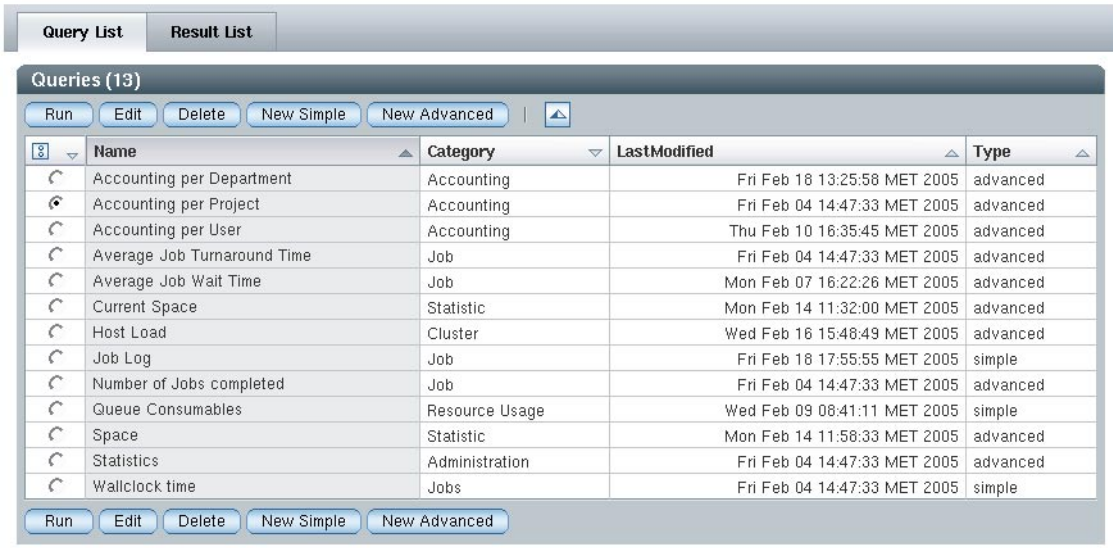


3 クエリーを保存または実行します。

- クエリーを保存するには、「**Save**」をクリックします。
- クエリーを実行するには、「**Run**」をクリックします。

▼ 高度なクエリーを実行する

- クエリーを実行します。
  - 今作成したクエリーを実行するには、「**Advanced Query**」画面で「**Run**」をクリックします。
  - 以前に保存したクエリーを実行するには、「**Query List**」画面でクエリーを選択して、「**Run**」をクリックします。



▼ 高度なクエリーを編集する

1. 「**Query List**」画面のリストからクエリーを選択します。
2. 「**Edit**」をクリックします。
   完成した「Advanced Query」画面が表示されます。
3. **SQL** クエリーを変更します。
4. 変更したクエリーを保存または実行します。
   - 変更したクエリーを保存するには、「**Save**」をクリックします。

- 変更したクエリーを実行するには、「**Run**」をクリックします。

高度なクエリーにおける実行時バインディング

高度なクエリーにおける実行時バインディングの構文は、次のとおりです。

```
LATEBINDING{ <column>;<operator>;<default value> }

<column>    name if the latebinding
<operator>  a SQL operator (e.g. = < > in .. )
<value>     default value (e.g. 'localhost' )
```

例 **5–3** 実行時バインディングの例

```
select hostname from sge_host where LATEBINDING{hostname, like, 'a%'}
select hostname from sge_host where LATEBINDING{hostname, in, ('localhost', 'foo.bar')}
```

◆ ◆ ◆ 第 6 章

Distributed Resource Management Application API による Grid Engine 関数の自動実行

N1 Grid Engine のコマンドを実行し、その結果を解析するスクリプトを作成することによって、N1 Grid Engine の関数を自動実行できます。ただし、より安定した、効率的な結果を得るために、C または Java™ 言語と Distributed Resource Management Application API (DRMAA) を使用できます。この章では DRMAA の概念を紹介し、C および Java 言語での DRMAA の使用方法について説明します。

この章の内容は、次のとおりです。

Distributed Resource Management Application API (DRMAA) の概要

Distributed Resource Management Application API (DRMAA、drama に近い発音) は、Distributed Resource Management System (DRMS) でのジョブの発行、監視、および制御を標準化するための Open Grid Forum 仕様です。DRMAA Working Group の目的は、実装が簡単で学びやすく、標準的な方法で DRMS との有用なアプリケーション統合を可能にする API を作成することです。

DRMAA 仕様は、言語、プラットフォーム、および DRMS に依存しません。さまざまな種類のシステムが、DRMAA 仕様を実装できるはずです。特定言語での DRMAA 実装のための追加の手引きとして、DRMAA Working Group は DRMAA 言語バインド仕様もいくつか作成しました。これらの仕様では、各言語で DRMAA 実装を何に似せるべきかを定義しています。

DRMAA 仕様は、現在 version 1.0 です。DRMAA Java Language Binding Specification も version 1.0 であり、DRMAA C Language Binding Specification も同じです。N1 Grid Engine 6.1 は、1.0 Java 言語バインドと 1.0 C 言語バインドの両方の実装を提供するば

かりでなく、下位互換性のためのそれぞれの旧バージョンの実装も提供します。DRMAA 1.0 仕様および言語別のバインド仕様については、Open Grid Forum DRMAA Working Group の Web サイト (http://drmaa.org/wiki/?sfProjectId=proj1076)を参照してください。

C 言語バインドを利用した開発

C 言語バインド用の重要ファイル

N1 Grid Engine 6.1 に付属する DRMAA C 言語バインド実装を使用するには、重要なファイルがある場所を知っておく必要があります。もっとも重要なファイルは、C アプリケーションから DRMAA の関数を利用できるようにするために、アプリケーションからインクルードする DRMAA ヘッダーファイルです。DRMAA ヘッダーファイルは、*sge-root*/include/drmaa.h にあります。ここで *sge-root* は、デフォルトで /usr/SGE です。DRMAA 関数の詳細な参照情報については、*sge-root*/man ディレクトリにある、N1 Grid Engine のマニュアルページの 5 節を参照してください。アプリケーションのコンパイルとリンクでは、*sge-root*/lib/*arch*/libdrmaa.so にある DRMAA 共有ライブラリを使用してください。

DRMAA ヘッダーファイルのインクルード

アプリケーションで DRMAA の関数を使用するには、DRMAA 関数を使用するすべてのソースファイルに DRMAA ヘッダーファイルをインクルードします。ソースファイルに DRMAA ヘッダーファイルをインクルードするには、ソースコードの通常は先頭近くに次の行を追加します。

```
#include "drmaa.h"
```

C アプリケーションのコンパイル

DRMAA アプリケーションをコンパイルする場合、コンパイラとリンカーに DRMAA を使用するよう指示するための追加のコンパイラ指令を含める必要があります。次の指示は Sun Studio Compiler Collection と gcc に該当しますが、ほかのコンパイラおよびリンカーには該当しません。ご使用のコンパイラおよびリンカー製品のマニュアルを参照してください。

2 つの指令を含めます。

- コンパイラコマンド行に次の文を追加して、DRMAA ヘッダーファイルをインクルードするようコンパイラに指示します。

```
-I<sge-root>/include
```

- コンパイラかリンカーコマンド行、またはその両方に次の文を追加して、DRMAA ライブラリをインクルードするようリンカーに指示します。

```
-ldrmaa
```

また、*sge-root*/lib/*arch* ディレクトリがライブラリ検索パス (Solaris オペレーティング環境および Linux の場合は LD_LIBRARY_PATH) に含まれていることを確認する必要があります。settings.sh または settings.csh ファイルを使用して環境設定した場合、*sge-root*/lib/arch ディレクトリは自動的には追加されません。

例 6–1 Sun Studio Compiler による C アプリケーションのコンパイル

Sun Studio Compiler を使用して DRMAA アプリケーションをコンパイルする例を示します。この例は、次のことを前提にしています。

- Solaris ホストで csh シェルを使用。
- N1 Grid Engine のインストール場所は /sge。
- DRMAA アプリケーションの保存場所は app.c。

コマンド例は、次のようになります。

```
% source /sge/default/common/settings.csh
% cc -I/sge/include -ldrmaa app.c
```

C アプリケーションの実行

コンパイルした DRMAA アプリケーションを実行するには、次のことを確認します。

sge-root/lib/*arch* ディレクトリがライブラリ検索パス (Solaris オペレーティング環境および Linux の場合は LD_LIBRARY_PATH) に含まれている。settings.sh または settings.csh ファイルを使用して環境設定した場合、*sge-root*/lib/arch ディレクトリは自動的には追加されません。

N1 Grid Engine 発行ホストであるマシンにログインしている。マシンが N1 Grid Engine 発行ホストでない場合、DRMAA 関数呼び出しはすべて失敗し、「DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE 」が返されます。

▼ DRMAA 0.95 C 言語バインドを使用する

デフォルトで有効な DRMAA 共有ライブラリは、version 1.0 の DRMAA C Language Binding Specification をサポートしています。ただし、下位互換性のため、Grid Engine には、version 0.95 の DRMAA C Language Binding Specification の実装も含まれていま

す。すべての新しいアプリケーションは、1.0 の共有ライブラリを使用して開発することを推奨しますが、0.95 の実装を必要とするアプリケーションが見つかることもあります。

version 0.95 の共有ライブラリを有効にする手順は、次のとおりです。

1 インストールした **Grid Engine** に対する変更権限を持つユーザーでログインします。

```
% su -
```

2 *sge-root*/lib/*arch* ディレクトリに移動します。

```
% cd /sge/lib/sol-sparc64
```

3 libdrmaa.so シンボリックリンクを削除します。

```
%  rm libdrmaa.so
```

4 **0.95** ライブラリへの新しいシンボリックリンクを作成します。

```
% ln -s libdrmaa.so.0.95 libdrmaa.so
```

Solaris または Linux プラットフォームの場合、共有ライブラリにはバージョン番号のタグが付いています。version 1.0 向けにコンパイルおよびリンクしたアプリケーションで、version 0.95 の共有ライブラリが有効にされている場合、および version 0.95 向けにコンパイルおよびリンクしたアプリケーションで、version 1.0 の共有ライブラリが有効にされている場合は、どちらもライブラリが見つからないというエラーになります。そのほかのプラットフォームの場合、1.0 アプリケーションは 0.95 共有ライブラリを正しく読み込みますが、未知のシンボルが原因でエラーになることがあります。0.95 アプリケーションは 1.0 共有ライブラリを正しく読み込みますが、DRMAA 関数によって予期しないエラーコードが返されて、エラーになる可能性があります。

- **version 1.0** の共有ライブラリに戻すには、手順 **1** ～ **3** を繰り返し、**1.0** ライブラリへの新しいシンボリックリンクを作成します。

  ```
  % ln -s libdrmaa.so.1.0 libdrmaa.so
  ```

C アプリケーション例

ここでは、C 言語バインドを使用したアプリケーション対話例をいくつか紹介します。Grid Engine Community サイトの「Howtos」セクションでも、そのほかの例を紹介しています。

例 **6–2** セッションの開始と停止

次のコードセグメントは、もっとも基本的な DRMAA C バインドプログラムの例です。

例 6–2 セッションの開始と停止 *(続き)*

DRMAA 関数を呼び出すたびにエラーコードが返されます。何も問題がなければ、コードは DRMAA_ERRNO_SUCCESS になります。エラーが発生すると、そのエラーに応じたエラーコードが返されます。各 DRMAA 関数は少なくとも 2 つのパラメータを取ります。2 つのパラメータは、エラーの場合にエラーメッセージに埋め込む文字列およびエラー文字列の最大の長さを表す整数です。

例の行 8 では、drmaa_init() を呼び出しています。この関数は DRMAA セッションを設定するもので、ほかの大部分の DRMAA 関数の前に呼び出します。drmaa_get_contact () のように、drmaa_init() の前に呼び出せる関数もありますが、これらの関数は一般的な情報を提供するだけです。drmaa_run_job() や drmaa_wait() のようにアクションを実行する関数は、drmaa_init() の復帰後に呼び出します。drmaa_init() が復帰する前にこのような関数を呼び出すと、「DRMAA_ERRNO_NO_ACTIVE_SESSION」というエラーコードが返されます。

dmraa_init() 関数はセッションを作成し、イベントクライアントリスナースレッドを開始します。このセッションは DRMAA を使用して発行されたジョブを整理するためのもので、スレッドはジョブの状態およびシステム全般に関する最新情報をキューマスターから受け取ります。drmaa_init() が正しく呼び出されたあと、呼び出し元のアプリケーションは、終了する前に drmaa_exit () の呼び出しも行います。終了する前にアプリケーションが drmaa_exit() を呼び出さないと、キューマスターに無効なイベントクライアントハンドルが残り、キューマスターのパフォーマンスが低下することがあります。

プログラムが終了する行 17 では、drmaa_exit() はセッションをクリーンアップして、イベントクライアントリスナースレッドを停止します。ほかの大部分の DRMAA 関数は、drmaa_exit() の前に呼び出します。drmaa_get_contact() のように、drmaa_exit() のあとで呼び出せる関数もありますが、これらの関数は一般的な情報を提供するだけです。drmaa_run_job() あるいは drmaa_wait() のようにアクションを実行する関数は、drmaa_exit() を呼び出す前にに呼び出します。drmaa_exit() を呼び出したあとでこのような関数を呼び出すと、「DRMAA_ERRNO_NO_ACTIVE_SESSION」というエラーコードが返されます。

```
01: #include
02: #include "drmaa.h"
03:
04: int main(int argc, char **argv) {
05:    char error[DRMAA_ERROR_STRING_BUFFER];
06:    int errnum = 0;
07:
08:    errnum = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER);
09:
10:    if (errnum != DRMAA_ERRNO_SUCCESS) {
11:       fprintf(stderr, "Could not initialize the DRMAA library: %s\n", error);
12:       return 1;
```

例 6–2 セッションの開始と停止 *(続き)*

```
13:    }
14:
15:    printf("DRMAA library was started successfully\n");
16:
17:    errnum = drmaa_exit(error, DRMAA_ERROR_STRING_BUFFER);
18:
19:    if (errnum != DRMAA_ERRNO_SUCCESS) {
20:       fprintf(stderr, "Could not shut down the DRMAA library: %s\n", error);
21:       return 1;
22:    }
23:
24:    return 0;
25: }
```

例 6–3 ジョブの実行

次は、DRMAA C バインドを使用して N1 Grid Engine にジョブを発行するコードセグメントの例です。このプログラムの最初と最後は、例 6–2 と同じです。違いは行 16 ～ 59 にあります。行 16 では、DRMAA がジョブテンプレートを割り当てています。ジョブテンプレートは、発行するジョブの情報を格納するための構造です。`drmaa_run_job()` または `drmaa_run_bulk_job()` の複数の呼び出しでは、同じテンプレートを再利用できます。

行 22 では、`DRMAA_REMOTE_COMMAND` 属性を設定しています。この属性は、実行するプログラムがある場所を DRMAA に伝えます。この属性の値は、実行可能ファイルへのパスです。パスは相対または絶対のどちらでもかまいません。相対の場合は、`DRMAA_WD` 属性を基準にしたパスで、デフォルトではユーザーのホームディレクトリです。DRMAA の属性については、`drmaa_attributes` のマニュアルページを参照してください。このプログラムが動作するには、`sleeper.sh` スクリプトがデフォルトパスになければなりません。

行 32 では、`DRMAA_V_ARGV` 属性を設定しています。この属性は、実行可能ファイルに渡す引数を DRMAA に伝えます。DRMAA の属性については、`drmaa_attributes` のマニュアルページを参照してください。

行 43 の `drmaa_run_job()` はジョブを発行します。DRMAA は、ジョブに割り当てられた ID を文字配列に書き込み、この配列が `drmaa_run_job()` に渡されます。これで、ジョブは `qsub` によって発行されたかのように動作します。この時点で `drmaa_exit()` を呼び出すか、プログラムを終了しても、ジョブは何の影響も受けません。

クリーンアップするために、行 54 でジョブテンプレートが削除されます。これによって、ジョブテンプレート用に DRMAA が確保していたメモリーが解放されますが、発行されたジョブは何の影響も受けません。

例 6–3 ジョブの実行 *(続き)*

最後に、行 61 で drmaa_exit() が呼び出されています。drmaa_exit() の呼び出しは、行 18 から始まっていた *if* 構造の外側にあります。これは、 drmaa_init() が呼び出されたあと、ほかのコマンドが成功したかどうかに関係なく、終了する前に drmaa_exit() を呼び出さなければならないためです。

```
01: #include
02: #include "drmaa.h"
03:
04: int main(int argc, char **argv) {
05:    char error[DRMAA_ERROR_STRING_BUFFER];
06:    int errnum = 0;
07:    drmaa_job_template_t *jt = NULL;
08:
09:    errnum = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER);
10:
11:    if (errnum != DRMAA_ERRNO_SUCCESS) {
12:       fprintf(stderr, "Could not initialize the DRMAA library: %s\n", error);
13:       return 1;
14:    }
15:
16:    errnum = drmaa_allocate_job_template(&jt, error, DRMAA_ERROR_STRING_BUFFER);
17:
18:    if (errnum != DRMAA_ERRNO_SUCCESS) {
19:       fprintf(stderr, "Could not create job template: %s\n", error);
20:    }
21:    else {
22:       errnum = drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, "sleeper.sh",
23:                                    error, DRMAA_ERROR_STRING_BUFFER);
24:
25:       if (errnum != DRMAA_ERRNO_SUCCESS) {
26:          fprintf(stderr, "Could not set attribute \"%s\": %s\n",
27:                   DRMAA_REMOTE_COMMAND, error);
28:       }
29:       else {
30:          const char *args[2] = {"5", NULL};
31:
32:          errnum = drmaa_set_vector_attribute(jt, DRMAA_V_ARGV, args, error,
33:                                              DRMAA_ERROR_STRING_BUFFER);
34:       }
35:
36:       if (errnum != DRMAA_ERRNO_SUCCESS) {
37:          fprintf(stderr, "Could not set attribute \"%s\": %s\n",
38:                   DRMAA_REMOTE_COMMAND, error);
39:       }
40:       else {
41:          char jobid[DRMAA_JOBNAME_BUFFER];
```

例 6–3 ジョブの実行 *(続き)*

```
42:
43:          errnum = drmaa_run_job(jobid, DRMAA_JOBNAME_BUFFER, jt, error,
44:                                 DRMAA_ERROR_STRING_BUFFER);
45:
46:          if (errnum != DRMAA_ERRNO_SUCCESS) {
47:             fprintf(stderr, "Could not submit job: %s\n", error);
48:          }
49:          else {
50:             printf("Your job has been submitted with id %s\n", jobid);
51:          }
52:       } /* else */
53:
54:       errnum = drmaa_delete_job_template(jt, error, DRMAA_ERROR_STRING_BUFFER);
55:
56:       if (errnum != DRMAA_ERRNO_SUCCESS) {
57:          fprintf(stderr, "Could not delete job template: %s\n", error);
58:       }
59:    } /* else */
60:
61:    errnum = drmaa_exit(error, DRMAA_ERROR_STRING_BUFFER);
62:
63:    if (errnum != DRMAA_ERRNO_SUCCESS) {
64:       fprintf(stderr, "Could not shut down the DRMAA library: %s\n", error);
65:       return 1;
66:    }
67:
68:    return 0;
69: }
```

Java 言語バインドを利用した開発

Java 言語バインド用の重要ファイル

N1 Grid Engine 6.1 に付属する DRMAA Java 言語バインド実装を使用するには、重要なファイルがある場所を知っておく必要があります。もっとも重要なファイルは DRMAA JAR ファイルの *sge-root*/lib/drmaa.jar です。DRMAA アプリケーションをコンパイルするには、CLASSPATH にこの DRMAA JAR ファイルを含めます。DRMAA クラスについては、*sge-root*/doc/javadocs ディレクトリにある DRMAA Javadoc ™ で解説しています。この Javadoc にアクセスするには、ブラウザで *sge-root*/doc/javadocs/index.html ファイルを開いてください。アプリケーションを実行する準備ができたら、必須のネイティブルーチンを提供する DRMAA 共有ライブラリの *sge-root*/lib/*arch*/libdrmaa.so も必要です。

DRMAA Java クラスとパッケージのインポート

アプリケーションで DRMAA クラスを使用するには、DRMAA クラスまたはパッケージをインポートします。たいていの場合、使用されるのは `org.ggf.drmaa` パッケージ内のクラスだけです。これらのパッケージは、個別にインポートすることも、ワイルドカードを使用してインポートすることもできます。めったにありませんが、`com.sun.grid.drmaa` にある N1 Grid Engine DRMAA 実装クラスへの参照が必要になることがあります。この場合は、クラスを個別にインポートすることも、特定パッケージ内のすべてのクラスをインポートすることもできます。`com.sun.grid.drmaa` クラスの名前は `org.ggf.drmaa` クラスの名前と重複していないため、名前空間の衝突を生むことなく両方のパッケージをインポートできます。

Java アプリケーションのコンパイル

DRMAA アプリケーションをコンパイルするには、`CLASSPATH` に *sge-root*`/lib/drmaa.jar` ファイルを追加します。`settings.sh` または `settings.csh` ファイルを使用して環境設定した場合、`drmaa.jar` ファイルは自動的には追加されません。

▼ NetBeans 5.*x* で DRMAA を使用する

NetBeans 5.0 または 5.5 プロジェクトで DRMAA クラスを使用する手順は、次のとおりです。

1. プロジェクトノードでマウスボタン 3 をクリックし、「プロパティー」を選択します。
2. プロジェクトで新しく構築ファイルを生成するか、または既存のファイルを使用するかを決めます。
   - プロジェクトで生成した構築ファイルを使用する場合は、次を実行します。
     a. 左の列にある「ライブラリ」を選択します。
     b. 「ライブラリを追加」をクリックします。
     c. 「ライブラリ」ダイアログボックスで「ライブラリを管理」をクリックします。
     d. 「ライブラリの管理」ダイアログボックスで「新規ライブラリ」をクリックします。

e. 「新規ライブラリ」ダイアログボックスの「ライブラリ名」フィールドに **DRMAA** と入力します。

f. 「了解」をクリックして、「新規ライブラリ」ダイアログボックスを閉じます。

g. 「**JAR**/フォルダを追加」をクリックします。

h. ファイル選択ダイアログボックスで *sge-root*/lib ディレクトリに移動し、drmaa.jar ファイルを選択します。

i. 「**JAR**/フォルダを追加」をクリックして、ファイル選択ダイアログボックスを閉じます。

j. 「了解」をクリックして、「ライブラリの管理」ダイアログボックスを閉じます。

k. **DRMAA** ライブラリを選択し、「ライブラリを追加」をクリックして、「ライブラリ」ダイアログボックスを閉じます。

- プロジェクトで既存の構築ファイルを使用する場合は、次を実行します。

a. 左の列にある「**Java** ソースのクラスパス」を選択します。

b. 「**JAR**/フォルダを追加」をクリックします。

c. ファイル選択ダイアログボックスで *sge-root*/lib ディレクトリに移動し、drmaa.jar ファイルを選択します。

d. 「選択」をクリックして、ファイル選択ダイアログボックスを閉じます。

3 「了解」をクリックしてプロパティーダイアログボックスを閉じます。

4 **DRMAA** 共有ライブラリがライブラリ検索パスに含まれていることを確認します。

NetBeans からアプリケーションを実行するには、DRMAA 共有ライブラリファイルの *sge-root*/lib/ *arch*/libdrmaa.so がライブラリ検索パス (Solaris オペレーティング環境および Linux の場合は LD_LIBRARY_PATH) に含まれていなければなりません。settings.sh または settings.csh ファイルを使用して環境設定した場合、*sge-root*/lib/*arch* ディレクトリは自動的には追加されません。共有ライブラリのパスを設定するには、次のいずれかを実行します。

- **NetBeans** を起動する前にシェルで環境設定します。

- 次のように、*netbeans-root*/etc/netbeans.conf ファイルに追加して環境設定します。

```
# Setup environment for SGE
. <sge-root>/<sge_cell>/common/settings.sh
ARCH=‘$SGE_ROOT/util/arch‘
LD_LIBRARY_PATH=$SGE_ROOT/lib/$ARCH; export LD_LIBRARY_PATH
```

Java アプリケーションの実行

コンパイルした DRMAA アプリケーションを実行するには、次のことを確認します。

- *sge-root*/lib/*arch* ディレクトリがライブラリ検索パス (Solaris オペレーティング環境および Linux の場合は LD_LIBRARY_PATH) が含まれている。settings.sh または settings.csh ファイルを使用して環境設定した場合、*sge-root*/lib/*arch* ディレクトリは自動的には追加されません。
- N1 Grid Engine 発行ホストであるマシンにログインしている。マシンが N1 Grid Engine 発行ホストでない場合、DRMAA 関数呼び出しはすべて失敗し、「DrmCommunicationException」がスローされます。

DRMAA 0.5 Java 言語バインドの利用

デフォルトで使用される DRMAA 共有ライブラリは、version 1.0 の DRMAA Java Language Binding Specification をサポートしています。ただし、下位互換性のため、N1 Grid Engine には version 0.5 の DRMAA Java Language Binding Specification の実装も含まれています。すべての新しいアプリケーションは、1.0 の共有ライブラリを使用して開発することを推奨しますが、0.5 実装を必要とするアプリケーションが見つかることもあります。

version 0.5 の drmaa.jar ファイルを使用するには、通常の *sge-root*/lib/drmaa.jar の前に、またはこのファイルの代わりに *sge-root*/lib/drmaa-0.5.jar ファイルを CLASSPATH に追加します。また、0.5 Java 言語バインドは、0.95 C 言語バインドの有効化を必要とします。137 ページの「DRMAA 0.95 C 言語バインドを使用する」を参照してください。

Java アプリケーション例

ここでは、Java 言語バインドを使用するアプリケーション対話例をいくつか紹介します。Grid Engine Community サイトの「Howtos」セクションでも、そのほかの例を紹介しています。

例 6–4 セッションの開始と停止

次のコードセグメントは、もっとも基本的な DRMAA Java バインドプログラムの例です。

プログラマが DRMAA を使用して行うことのすべては、Session オブジェクトを介して行います。Session オブジェクトは、SessionFactory から取得します。SessionFactory は、静的な SessionFactory.getFactory() メソッドから取得します。この連鎖の理由は、すべての org.ggf.drmaa.* クラスが、あらゆる DRMAA Java 言語バインド実装によって使用される不変のパッケージとみなされるべきであるためです。パッケージが不変であるため、特定の実装を読み込む際、SessionFactory はシステムプロパティーを使用して、その実装のセッションファクトリを探し、読み込みます。そして、このセッションファクトリが、適当とみなす方法でセッションを作成します。セッションファクトリが存在するとしても、一度に存在できるセッションは 1 つだけであることに注意してください。

行 9 の SessionFactory.getFactory() はセッションファクトリインスタンスを取得します。行 10 の SessionFactory.getSession () はセッションインスタンスを取得します。行 13 の Session.init () はセッションを初期化します。"" は、初期化引数が必要ないため、新しいセッションを作成するための連絡文字列として渡されます。

Session.init() はセッションを作成し、イベントクライアントリスナースレッドの実行を開始します。このセッションは DRMAA を使用して発行されたジョブを整理するためのもので、スレッドはジョブの状態およびシステム全般に関する最新情報をキューマスターから受け取ります。Session.init() が正しく呼び出されたあと、呼び出し元アプリケーションは、終了する前に Session.()exit() の呼び出しも行います。終了する前にアプリケーションが Session.()exit() を呼び出さないと、キューマスターに無効なイベントクライアントハンドルが残り、キューマスターのパフォーマンスが低下することがあります。Session.()exit() が呼び出されていることを確認するには、Runtime.addShutdownHook() メソッドを使用します。exit() gets called.

プログラムの終了時の 行 14 の Session.()exit() は、セッションのクリーンアップをして、イベントクライアントリスナースレッドを停止します。ほかの大部分の DRMAA メソッドは、Session.()exit() の前に呼び出します。Session.getContact() のように、Session.()exit() の後で呼び出せる関数もありますが、これら関数は一般的な情報を提供するだけです。Session.runJob() または Session.()wait() のようにアクションを実行する関数は、Session.()exit() を呼び出す前に呼び出します。Session.()exit() を呼び出したあとでこのような関数を呼び出すと、「NoActiveSessionException」がスローされます。

```
01: package com.sun.grid.drmaa.howto;
02:
03: import org.ggf.drmaa.DrmaaException;
04: import org.ggf.drmaa.Session;
05: import org.ggf.drmaa.SessionFactory;
```

例 6–4 セッションの開始と停止 (続き)

```
06:
07: public class Howto1 {
08:    public static void main(String[] args) {
09:       SessionFactory factory = SessionFactory.getFactory();
10:       Session session = factory.getSession();
11:
12:       try {
13:          session.init("");
14:          session.exit();
15:       } catch (DrmaaException e) {
16:          System.out.println("Error: " + e.getMessage());
17:       }
18:    }
19: }
```

例 6–5 ジョブの実行

次は、DRMAA Java 言語バインドを使用して N1 Grid Engine にジョブを発行するコードセグメントの例です。このプログラムの最初と最後は、例 6–4 と同じです。違いは行 16 ～ 24 にあります。

行 16 では、DRMAA がジョブテンプレートを割り当てています。ジョブテンプレートは、発行するジョブの情報を格納するためのオブジェクトです。`Session.runJob()` または `Session.runBulkJobs()` の複数の呼び出しでは、同じテンプレートを再利用できます。

行 17 では、`remoteCommand` 属性を設定しています。この属性は、実行するプログラムがある場所を DRMAA に伝えます。この属性の値は、実行可能ファイルへのパスです。パスは相対または絶対のどちらでもかまいません。相対の場合は、`workingDirectory` 属性を基準にしたパスで、デフォルトではユーザーのホームディレクトリです。DRMAA の属性については、DRMAA Javadoc または `drmaa_attributes` のマニュアルページを参照してください。このプログラムが動作するには、`sleeper.sh` スクリプトがデフォルトパスになければなりません。

行 18 では、`args` 属性を設定しています。この属性は、実行可能ファイルに渡す引数を DRMAA に伝えます。DRMAA の属性については、DRMAA Javadoc または `drmaa_attributes` のマニュアルページを参照してください。

行 20 の `Session.runJob()` はジョブを発行します。このメソッドは、キューマスターがジョブに割り当てた ID を返します。これで、ジョブは `qsub` によって発行されたかのように動作します。この時点で `Session.()exit()` を呼び出すか、プログラムを終了しても、ジョブは何の影響も受けません。

例 6–5 ジョブの実行 *(続き)*

クリーンアップするために、行 24 でジョブテンプレートが削除されます。これによって、ジョブテンプレート用に DRMAA が確保していたメモリーが解放されますが、発行されたジョブは何の影響も受けません。

```
01: package com.sun.grid.drmaa.howto;
02:
03: import java.util.Collections;
04: import org.ggf.drmaa.DrmaaException;
05: import org.ggf.drmaa.JobTemplate;
06: import org.ggf.drmaa.Session;
07: import org.ggf.drmaa.SessionFactory;
08:
09: public class Howto2 {
10:    public static void main(String[] args) {
11:       SessionFactory factory = SessionFactory.getFactory();
12:       Session session = factory.getSession();
13:
14:       try {
15:          session.init("");
16:          JobTemplate jt = session.createJobTemplate();
17:          jt.setRemoteCommand("sleeper.sh");
18:          jt.setArgs(Collections.singletonList("5"));
19:
20:          String id = session.runJob(jt);
21:
22:          System.out.println("Your job has been submitted with id " + id);
23:
24:          session.deleteJobTemplate(jt);
25:          session.exit();
26:       } catch (DrmaaException e) {
27:          System.out.println("Error: " + e.getMessage());
28:       }
29:    }
30: }
```

◆ ◆ ◆ 第 7 章

7 エラーメッセージと障害追跡

この章では、Grid Engine システムのエラーメッセージ処理について説明し、一般的な各問題の解決方法のヒントを紹介します。

ソフトウェアによるエラーレポートの検出方法

Grid Engine ソフトウェアは、メッセージを特定のファイルに記録したり、電子メールを送信したり、またはこの両方を行って、エラーと警告をレポートします。ログファイルには、メッセージ ファイルとジョブ STDERR 出力が含まれます。

stderr ジョブが開始されると、ジョブ スクリプトの標準エラー (STDERR) 出力の出力先がファイルにリダイレクトされます。デフォルトのファイル名とファイルの保存場所を使用するか、qsub コマンドのオプションを使用してファイル名と場所を指定することができます。詳細は、Grid Engine システムのマニュアルページを参照してください。

sge_qmaster、sge_schedd、および sge_execd には、個別のメッセージファイルがあります。 ファイルの名前は同じで、messages です。sge_qmaster ログファイルは、マスタースプールディレクトリに保存されます。sge_schedd メッセージ ファイルは、スケジューラスプールディレクトリに保存されます。実行デーモンのログファイルの場所は、実行デーモンのスプールディレクトリです。スプールディレクトリについては、『Sun N1 Grid Engine 6.1 インストールガイド』の「ルートディレクトリの下のスプールディレクトリ」を参照してください。

各メッセージは、ファイル内の 1 行を占めます。各メッセージは、縦線記号 (|) で区切られた 5 つのコンポーネントに分割されます。

メッセージのコンポーネントは、次のとおりです。

1. 1 番目のコンポーネントは、メッセージのタイムスタンプです。
2. 2 番目のコンポーネントは、メッセージを生成する Grid Engine システムデーモンを指定します。
3. 3 番目のコンポーネントは、デーモンが動作しているホストの名前です。
4. 4 番目は、メッセージの種類です。メッセージの種類は、次のいずれかです。
   - 通知の `N` (情報提供が目的)
   - 情報の `I` (情報提供が目的)
   - 警告の `W`
   - エラーの `E` (エラー状態の検出)
   - 重大の `C` (プログラムの異常終了になる可能性あり)

   クラスタ構成で `loglevel` パラメータを使用して、グローバルまたはローカルベースで、記録するメッセージの種類を指定してください。
5. 5 番目のコンポーネントは、メッセージのテキストです。

注 – なんらかの理由でエラーログファイルにアクセスできない場合、Grid Engine システムは、対応するホストのファイル `/tmp/sge_qmaster_messages`、`/tmp/sge_schedd_messages`、または `/tmp/sge_execd_messages` にエラーメッセージを記録しようとします。

場合によって、Grid Engine システムは、ユーザー、管理者、またはその両方に電子メールでエラーイベントを通知します。Grid Engine システムが送信する電子メールメッセージには、メッセージの本文は含まれません。メッセージのテキストは、メールの件名フィールドにすべて含まれます。

異なるエラーコードまたは終了コードの意味

次の表に、複数のジョブ関連エラーコードまたは終了コードの意味を示します。これらのコードは、あらゆる種類のジョブに該当します。

表 7–1 ジョブ関連のエラーまたは終了コード

スクリプト/方法	終了またはエラーコード	意味
Job スクリプト	0	成功
	99	再度キューに入れる
	Rest	成功。アカウンティングファイルの終了コード

表 7–1 ジョブ関連のエラーまたは終了コード (続き)

スクリプト/方法	終了またはエラーコード	意味
prolog/epilog	0	成功
	99	再度キューに入れる
	Rest	キューエラー状態。ジョブは再度キューに入れられる

次の表に、並列環境 (PE) 構成に関連するジョブのエラーコードまたは終了コードの意味を示します。

表 7–2 並列環境関連のエラーまたは終了コード

スクリプト/方法	終了またはエラーコード	意味
pe_start	0	成功
	Rest	キューをエラー状態に設定。ジョブは再度キューに入れられる
pe_stop	0	成功
	Rest	キューをエラー状態に設定。ジョブは再度キューには入れられない

次の表に、キュー構成に関連するジョブのエラーコードまたは終了コードの意味を示します。これらのコードは、対応メソッドが上書きされた場合だけ有効です。

表 7–3 キュー関連のエラーまたは終了コード

スクリプト/方法	終了またはエラーコード	意味
ジョブ開始	0	成功
	Rest	成功。ほかの意味は特になし
一時停止	0	成功
	Rest	成功。ほかの意味は特になし
再開	0	成功
	Rest	成功。ほかの意味は特になし
終了	0	成功
	Rest	成功。ほかの意味は特になし

次の表に、チェックポイント設定に関連するジョブのエラーコードまたは終了コードの意味を示します。

表 7–4 チェックポイント設定関連のエラーまたは終了コード

スクリプト/方法	終了またはエラーコード	意味
チェックポイント	0	成功
	Rest	成功。ただし、カーネルチェックポイントの場合、チェックポイントが成功しなかったことを意味する。
移行	0	成功
	Rest	成功。ただし、カーネルチェックポイントの場合、チェックポイントが成功しなかったことを意味する。移行は行われる。
再開	0	成功
	Rest	成功。ほかの意味は特になし
後処理	0	成功
	Rest	成功。ほかの意味は特になし

正常に実行されたジョブに対して、`qacct -j` コマンドからの出力は、「`failed`」フィールドに「`0`」を示し、「`exit_status`」フィールドにジョブの終了ステータスを示します。ただし、シェパードがジョブを正常に実行できない場合があります。たとえば、epilog スクリプトが失敗したり、シェパードがジョブを開始できない場合があります。このような場合、「`failed`」フィールドは、次の表のコードの値のいずれかを表示します。

表 7–5 `qacct -j failed` フィールドコード

コード	説明	acctvalid	ジョブに対する意味
0	No failure	t	ジョブは実行され、正常に終了された
1	Presumably before job	f	ジョブを開始できなかった
3	Before writing config	f	ジョブを開始できなかった
4	Before writing PID	f	ジョブを開始できなかった
5	On reading config file	f	ジョブを開始できなかった
6	Setting processor set	f	ジョブを開始できなかった
7	Before prolog	f	ジョブを開始できなかった

表 7–5 `qacct -j failed` フィールドコード *(続き)*

コード	説明	acctvalid	ジョブに対する意味
8	In prolog	f	ジョブを開始できなかった
9	Before pestart	f	ジョブを開始できなかった
10	In pestart	f	ジョブを開始できなかった
11	Before job	f	ジョブを開始できなかった
12	Before pestop	t	ジョブは実行され、PE 停止手続きの呼び出し前に障害が発生した
13	In pestop	t	ジョブは実行され、PE 停止手続きで障害が発生した
14	Before epilog	t	ジョブは実行され、epilog スクリプトの呼び出し前に障害が発生した
15	In epilog	t	ジョブは実行され、epilog 内で障害が発生した
16	Releasing processor set	t	ジョブは実行され、プロセッサセットは解放できなかった
24	Migrating (checkpointing jobs)	t	ジョブは実行され、移行される予定
25	Rescheduling	t	ジョブは実行され、再スケジューリングされる予定
26	Opening output file	f	ジョブを開始できず、stderr/stdout ファイルを開けない
27	Searching requested shell	f	ジョブを開始できず、シェルを検出できない
28	Changing to working directory	f	ジョブを開始できず、エラーで開始ディレクトリへ移動した
100	Assumedly after job	t	ジョブは実行され、信号によってジョブ終了させられた。

「コード」の列には、「`failed`」フィールドの値が一覧表示されています。「説明」列には、`qacct -j` の出力で表示されるテキストが一覧表示されています。`acctvalid` が `t` に設定されている場合、ジョブアカウンティングの値は有効です。`acctvalid` が `f` に設定されている場合、アカウンティングレコードのリソース使用率は有効ではありません。「ジョブに対する意味」の列では、ジョブが実行されたのかどうかが示されています。

デバッグモードでの Grid Engine システムのプログラムの実行

重大なエラーが発生した場合に、問題の特定に十分な情報がエラー記録機構によって生成されないことがあります。このため Grid Engine システムは、ほぼすべての補助プログラムとデーモンをデバッグモードで実行する機能が用意されています。デバッグのレベルは、提供される情報の量および深さに応じて異なります。デバッグのレベルは、0 から 10 の範囲で、10 はもっとも詳細な情報を提供するレベル、0 はデバッグ無効です。

デバッグレベルを設定するために、Grid Engine システムの配布には、`.cshrc` または `.profile` リソースファイルに対する拡張機能が用意されています。`csh` または `tcsh` のユーザーの場合は、ファイル *sge-root*`/util/dl.csh` が含まれています。`sh` または `ksh` のユーザーの場合は、対応するファイルの名前は *sge-root*`/ util/dl.sh` です。標準のリソースファイルに、これらのファイルを取り込む必要があります。`csh` または `tcsh` のユーザーの場合は、`.cshrc` ファイルに次の行を含めます。

```
source sge-root/util/dl.csh
```

`sh` または `ksh` のユーザーの場合は、`.profile` ファイルに次の行を含めます。

```
. sge-root/util/dl.sh
```

いったんログアウトして、ログインし直すと、次のコマンドを使用してデバッグレベルを設定できます。

```
% dl level
```

level が 0 より大きい場合、Grid Engine システムのコマンドを開始すると、トレース出力を `STDOUT` に書き込むようコマンドに強制します。トレース出力には、警告メッセージ、ステータスメッセージ、エラーメッセージ、および内部的に呼び出されたプログラムモジュールの名前が含まれます。メッセージには、ユーザーが指定するデバッグレベルに応じて、エラーの報告に役立つ行番号情報も含まれます。

注 – デバッグトレースを監視するには、大きなサイズのスクロール行バッファーを持つウィンドウを使用するようにします。たとえば、1000 行のスクロール行バッファーを使用します。

注 – ウィンドウが `xterm` の場合は、`xterm` ログ記録機構を使用して、あとでトレース出力を調べることができます。

デバッグモードで Grid Engine システムデーモンの 1 つを実行すると、デーモンが端末接続を維持して、トレース出力を書き出します。こうした端末接続は、使用している端末エミュレーションの割り込み文字を入力することによって打ち切ることができます。たとえば、Control-C などを使用します。

デバッグモードを無効にするには、デバッグレベルを 0 に戻します。

dbwriter デバッグレベルの設定

sgedbwriter スクリプトは、dbwriter プログラムを開始します。このスクリプトは、*sge_root* /dbwriter/bin/sgedbwriter に保存されています。sgedbwriter スクリプトは、dbwriter の構成ファイルである dbwriter.conf を読み取ります。この構成ファイルは、*sge_root*/*cell* /common/dbwriter.conf に保存されています。この構成ファイルは、dbwriter のデバッグレベルを設定します。例は次のとおりです。

```
#
# デバッグレベル
# 有効な値: WARNING、 INFO、 CONFIG、 FINE、 FINER、 FINEST、 ALL
#
DBWRITER_DEBUG=INFO
```

dbwriter コマンドの –debug オプションを使用すると、dbwriter により作成されるメッセージの数を変更できます。通常は、デフォルトのデバッグレベルである info を使用してください。より詳細なデバッグレベルを使用すると、dbwriter によって出力されるデータの量が大幅に増加します。

次のデバッグレベルを指定できます。

warning	重大なエラーと警告のみが表示されます。
info	多数の参考メッセージが追加されます。info がデフォルトのデバッグレベルです。
config	たとえば規則の処理に関する、dbwriter 構成に関連する追加の情報が得られます。
fine	さらに多くの情報が作成されます。このデバッグレベルを選択すると、dbwriter によって実行されるすべての SQL 文が出力されます。
finer	デバッグ用に使用します。
finest	デバッグ用に使用します。
all	すべてのレベルの情報を表示します。デバッグ用に使用します。

問題の診断

Grid Engine システムには、問題の診断に役立ついくつかの報告手段が用意されています。次の節では、それらの使用方法を簡単に説明します。

保留中のジョブが振り分けられない

保留中のジョブが実行可能な状態であることが明らかであるにもかかわらず、振り分けられない場合があります。Grid Engine システムには、その理由を診断するために、`qstat -j` *job-id* と `qalter -w v` *job-id* のユーティリティーとオプションのペアがあります。

- `qstat -j` *job-id*

  有効である場合は、`qstat -j` *job-id* は最後にスケジューリングが行われたときに特定のジョブが割り振られなかった理由のリストを示します。この監視は、有効または無効にすることができます。この監視機能は、`schedd` デーモンと `qmaster` との間の通信で望ましくないオーバーヘッドを引き起こす可能性があるため、無効にすることをお勧めします。`sched_conf` (5) のマニュアルページの `schedd_job_info` を参照してください。ID が `242059` のジョブに関する出力例を次に示します。

```
% qstat -j 242059
scheduling info: queue "fangorn.q" dropped because it is temporarily not available
queue "lolek.q" dropped because it is temporarily not available
queue "balrog.q" dropped because it is temporarily not available
queue "saruman.q" dropped because it is full
cannot run in queue "bilbur.q" because it is not contained in its hard queuelist (-q)

cannot run in queue "dwain.q" because it is not contained in its hard queue list (-q)
has no permission for host "ori"
```

  この情報は、`schedd` デーモンによって直接生成されます。この情報の生成では、クラスタの現在の使用率が考慮されます。この情報には、必要な情報が含まれないことがあります。たとえば、ほかのユーザーのジョブによってすべてのキュースロットがすでに占有されている場合、問題のジョブに関する詳細なメッセージは生成されません。

- `qalter -w v` *job-id*

このコマンドは、基本的にジョブが割り振られない理由を一覧表示します。この目的のため、ドライスケジューリングが実行されます。スロットを含めて消費可能なすべてのリソースが、そのジョブ用に完全に利用可能であるとみなされます。同様に、負荷値も変化するため、すべての負荷値は無視されます。

エラー状態 E と報告されるジョブまたはキュー

ジョブまたはキューのエラーが、qstat 出力で、大文字の E で示されます。

ジョブがエラー状態になるのは、Grid Engine システムがジョブを実行しようとして、そのジョブに固有の理由で実行に失敗した場合です。

キューがエラー状態になるのは、Grid Engine システムがジョブを実行しようとして、そのキューに固有の理由で実行が失敗した場合です。

Grid Engine システムには、ジョブ実行エラーが発生した場合に、ユーザーおよび管理者がその診断情報を収集するための一連の機能が用意されています。キューおよびジョブのエラーの状態のどちらも、原因はジョブの実行失敗にあります。そのため、診断の機能は両方の種類のエラー状態に適用できます。

- ユーザー宛て中止メール。qsub -m a コマンドを使用してジョブが発行された場合は、-M *user*[@*host*] オプションで指定されたアドレスに中止メールが送信されます。中止メールには、ジョブエラーに関する診断情報が含まれています。中止メールを情報源として使用することをお勧めします。
- **qacct** アカウンティング。中止メールが得られない場合は、qacct -j コマンドを実行できます。このコマンドによって、Grid Engine システムのジョブアカウンティング機能からジョブのエラーに関する情報を入手できます。
- 管理者宛て中止メール。 管理者は、適切な電子メールアドレスを指定することによって、ジョブ実行時の問題に関する管理者宛てメールを送信するよう指示できます。sge_conf(5) のマニュアルページの administrator_mail を参照してください。管理者宛てのメールには、ユーザー宛ての中止メールよりも詳しい診断情報が含まれています。ジョブ実行エラーが頻繁に発生する場合に、管理者宛てメールを利用することをお勧めします。
- **Message** ファイル。 管理者宛てメールが得られない場合は、qmaster の messages ファイルをまず調べてください。適切なジョブ ID を検索することによって、特定のジョブに関するエントリを見つけることができます。デフォルトのインストールでは、qmaster messages ファイルは *sge-root*/ *cell*/spool/qmaster/messages に保存されています。

  ジョブの起動元の execd デーモンのメッセージに、補足情報が含まれていることもあります。qacct -j *job-id* を使用して、ジョブの起動元のホストを確認し、*sge-root* /*cell*/spool/*host*/messages でジョブ ID を検索してください。

一般的な問題の障害追跡

この節では、一般的な問題の原因の診断と対処に役立つ情報を説明します。

- 問題 — ジョブの出力ファイルに「Warning: no access to tty; thus no job control in this shell...」というメッセージが出力される。
  - 考えられる原因 — 1 つ以上のログインファイルに stty コマンドが含まれています。これらのコマンドが役立つのは、端末が存在する場合のみです。
  - 考えられる対策 — バッチジョブは端末に関連付けられません。すべての stty コマンドをログインファイルから削除するか、if 文で stty コマンドをまとめてください。if 文は、処理の前に端末をチェックします。次の例に if 文を示します。

```
/bin/csh:
stty -g             # 端末の状態を確認します。
if ($status == 0)   # 端末が存在すれば、
正常に終了します。
<put all stty commands in here>
endif
```

- 問題 — ジョブの標準エラーログファイルに「'tty': Ambiguous」というメッセージが出力される。しかし、ジョブスクリプトで呼び出されるユーザーのシェルには、tty に対する参照が存在しない。
  - 考えられる原因 — デフォルトでは、shell_start_mode は posix_compliant です。このため、すべてのジョブスクリプトは、キュー定義で指定されたシェルで実行されます。スクリプトは、ジョブスクリプトの先頭行に指定されたシェルでは実行されません。
  - 考えられる対策 — qsub コマンドに -S フラグを使用するか、shell_start_mode を unix_behavior に変更してください。
- 問題 — コマンド行からはジョブスクリプトを実行できるが、qsub コマンドを使用して実行すると、ジョブスクリプトが失敗する。
  - 考えられる原因 — ジョブに対するプロセス数が制限されている可能性があります。制限が設定されているかどうかをテストするには、limit と limit -h 関数を実行するテストスクリプトを作成してください。シェルプロンプトから qsub コマンドを使用して両方の関数を対話的に実行し、結果を比較します。
  - 考えられる対策 — 構成ファイルから、シェルで制限を設けるすべてのコマンドを削除してください。
- 問題 — 実行ホストから負荷 99.99 が報告される。
  1. 考えられる原因 — ホストで execd デーモンが動作していません。

     考えられる解決策 — 実行ホストで root になり、$SGE_ROOT/default/common/'rcsge' スクリプトを実行することによって execd デーモンを起動してください。

2. 考えられる原因 — デフォルトドメインの指定に誤りがあります。

   考えられる解決策 — Grid Engine システムの管理者になって、`qconf -mconf` コマンドを実行し、`default_domain` 変数を `none` に変更してください。

3. 考えられる原因 — `qmaster` ホストが認識している実行ホスト名と、実行ホストが認識している自身の名前が異なります。

   考えられる解決策 — 計算クラスタのホスト名の解決に DNS を使用している場合は、主ホスト名として絶対パスによるドメイン名 (FQDN) が返されるように `/etc/hosts` と NIS を構成してください。もちろん、`168.0.0.1 myhost.dom.com myhost` などの短い別名を定義して使用することができます。

   DNS を使用していない場合は、すべての `/etc/hosts` ファイルと NIS テーブルが、`168.0.0.1 myhost.corp myhost` または `168.0.0.1 myhost` のように一致していることを確認してください。

- 問題 — 次のメッセージのような警告が 30 秒ごとに *cell*`/spool/` *host*`/messages` に出力される。

  ```
  Tue Jan 23 21:20:46 2001|execd|meta|W|local
  configuration meta not defined - using global configuration
  ```

  しかし、*cell*`/common/local_conf` には、各ホスト用のファイルがあり、それぞれに FQDN が存在する。

  - 考えられる原因 — 使用しているマシン `meta` では、ホスト名解決でショート名が返されるのに対し、マスターマシンでは、FDQN 付きの `meta` が返されます。
  - 考えられる解決策 — この点に関して、`/etc/hosts` のすべてのファイルと NIS テーブルの間に矛盾がないようにしてください。この例では、ホスト `meta` の `/etc/hosts` ファイルに、誤って次のようなテキストの行が含まれている可能性があります。

    ```
    168.0.0.1 meta meta.your.domain
    ```

    正しくは、この行は次のようにします。

    ```
    168.0.0.1 meta.your.domain meta.
    ```

- 問題 — デーモンの `messages` ファイルに `CHECKSUM ERROR`、`WRITE ERROR`、または `READ ERROR` というメッセージが表示される場合がある。
  - 考えられる原因 — 1 秒間隔で出力されるのでない限り、何もする必要はありません。一般的にこれらのメッセージは、1 日に 1 から 30 回表示されます。
- 問題 — ジョブが特定のキューで終了し、`qmaster/messages` に次のメッセージを返す。

  ```
  Wed Mar 28 10:57:15 2001|qmaster|masterhost|I|job 490.1
  finished on host exechost
  ```

次に、実行ホストの exechost/messages ファイルには次のエラーメッセージが出力される。

```
Wed Mar 28 10:57:15 2001|execd|exechost|E|can't find directory
"active_jobs/490.1" for reaping job 490.1

Wed Mar 28 10:57:15 2001|execd|exechost|E|can't remove directory
"active_jobs/490.1": opendir(active_jobs/490.1) failed:
Input/output error
```

- 考えられる原因 — 自動マウントされる $SGE_ROOT ディレクトリがマウント解除されたため、sge_execd デーモンが現在の作業ディレクトリを失った可能性があります。
- 考えられる解決策 — execd ホストにローカルのスプールディレクトリを使用してください。qmon または qconf コマンドを使用して、execd_spool_dir パラメータを設定してください。

- 問題 — qrsh ユーティリティーで対話型ジョブを発行すると、次のエラーメッセージが表示される。

```
% qrsh -l mem_free=1G error: error: no suitable queues
```

しかし、qsub コマンドを使用したバッチジョブの発行に対してキューは使用可能で、これらのキューは、qhost -l mem_free=1G と qstat -f -l mem_free=1G を使用して照会できます。

- 考えられる原因 — 「error: no suitable queues」というメッセージの原因は、qrsh などの対話型ジョブに対してデフォルトで有効になる submit の -w e オプションにあります。qrsh(1) のマニュアルページの -w e を参照してください。現在のクラスタ構成に従ってジョブが振り分け可能であるかどうかを qmaster が確実に判断できない場合、このオプションがあると、submit コマンドで問題が発生します。この仕組みの意図は、許可できないジョブ要求を事前に拒否することです。
- 対策 — この場合は、mem_free が消費可能リソースに設定されているにもかかわらず、ユーザーが各ホストで使用可能にするメモリーサイズを指定しなかったことが原因です。メモリー負荷値はそれぞれに異なるため、この検査ではメモリー負荷値は意図的に考慮されません。このため、クラスタ構成の一部としては表示されません。次のいずれかを行います。
  - qrsh のデフォルトオプションである -w e を -w n オプションに使用して明示的に無効にすることでこの検査を全般的に省略する。このコマンドを *sge-root*/ *cell*/common/cod_request に含めることもできます。
  - mem_free を消費可能リソースとして管理する場合は、qconf -me *hostname* を使用して、host_conf の complex_values にあるホストに対し mem_free の容量を指定する。

- mem_free を消費可能リソースとして管理しない場合は、 qconf -mc *hostname* を使用して、complex(5) の consumable 列で消費不可リソースに再度戻す。

- 問題 — qrsh が自分のノードに振り分けられない。このとき qsh シェルから次のようなメッセージが返される。

```
host2 [49]% qrsh -inherit host2 hostname
error: executing task of job 1 failed:

host2 [50]% qrsh -inherit host4 hostname
host4
```

  - 考えられる原因 — gid_range が十分ではないことが考えられます。gid_range には、1 つの数字ではなく範囲として定義してください。Grid Engine システムは、ホスト上の各ジョブに固有の gid を割り当てます。
  - 考えられる解決策 — gid_range を qconf -mconf コマンドまたは QMON で調整してください。次のような範囲が考えられます。

```
gid_range                   20000-20100
```

- 問題 — 並列ジョブ内で使用すると qrsh -inherit -V が機能しない。次のメッセージが返される。

```
cannot get connection to "qlogin_starter"
```

  - 考えられる原因 — この問題は入れ子にされた qrsh 呼び出しで発生します。問題の原因は -V オプションです。最初の qrsh -inherit 呼び出しでは、環境変数 TASK_ID が設定されます。TASK_ID は、並列ジョブ内の密接に統合されたタスクの ID です。2 番目の qrsh -inherit 呼び出しでは、この環境変数を使用してタスクを登録します。すでに実行中の最初のタスクと同じ ID を持つタスクを開始しようとするため、このコマンドは失敗します。
  - 考えられる解決策 — qrsh -inherit を呼び出す前に TASK_ID の設定を解除するか、-V ではなく -v オプションを使用してください。このオプションにより、実際に必要な環境変数だけがエクスポートされます。

- 問題 — qrsh がまったく機能していないように見える。次のようなメッセージが返される。

```
host2$ qrsh -verbose hostname
local configuration host2 not defined - using global configuration
waiting for interactive job to be scheduled ...
Your interactive job 88 has been successfully scheduled.
Establishing /share/gridware/utilbin/solaris64/rsh session
to host exehost ...
rcmd: socket: Permission denied
/share/gridware/utilbin/solaris64/rsh exited with exit code 1
```

```
reading exit code from shepherd ...
error: error waiting on socket for client to connect:
Interrupted system call
error: error reading return code of remote command
cleaning up after abnormal exit of
/share/gridware/utilbin/solaris64/rsh
host2$
```

- 考えられる原因 — qrsh に対する権限が正しく設定されていません。
- 考えられる解決策 — $SGE_ROOT/utilbin/ にある次のファイルの権限を確認してください。rlogin および rsh は、 setuid で、 root に所有されていなければならないことに注意してください。

```
-r-s--x--x 1 root root 28856 Sep 18 06:00 rlogin*
-r-s--x--x 1 root root 19808 Sep 18 06:00 rsh*
-rwxr-xr-x 1 sgeadmin adm 128160 Sep 18 06:00 rshd*
```

注 – *sge-root* ディレクトリも setuid オプションで NFS マウントされている必要があります。*sge-root* が発行クライアントから nosuid でマウントされている場合、qrsh と関連するコマンドは機能しません。

- 問題 – 分散 make を開始しようとすると、qmake が次のエラーメッセージを表示して終了する。

```
qrsh_starter: executing child process
qmake failed: No such file or directory
```

- 考えられる原因 — Grid Engine システムは、実行ホストで qmake のインスタンスを起動します。Grid Engine システム環境 (特に PATH 変数) がユーザーのシェルリソースファイル (.profile または .cshrc) に設定されていない場合、この qmake 呼び出しは失敗します。
- 考えられる解決策 — -v オプションを使用して、PATH 環境変数を qmake ジョブにエクスポートします。一般的な qmake 呼び出しは次のようになります。

```
qmake -v PATH -cwd -pe make 2-10 --
```

- 問題 — qmake ユーティリティーを使用すると、次のエラーメッセージが返される。

```
waiting for interactive job to be scheduled ...timeout (4 s)
expired while waiting on socket fd 5

Your "qrsh" request could not be scheduled, try again later.
```

- 考えられる原因 — qmake の呼び出し元であるシェルに ARCH 環境変数が正しく設定されていない可能性があります。
- 考えられる解決策 – クラスタで使用可能なホストに対応するサポート値を ARCH 変数に適切に設定するか、発行時に qmake -v ARCH=solaris64 ... などの適切な値を指定してください。

一般的なアカウンティングおよびレポートコンソールエラー

問題: Version 2.0.3 の Sun Web コンソールのインストールが失敗し、次のエラーメッセージが表示される。

```
# ./inst_reporting
...
Register the N1 SGE reporting module in the webconsole

    Registering com.sun.grid.arco_6u3.

Starting Sun(TM) Web Console Version 2.0.3...
Ambiguous output redirect.
```

対処方法: このバージョンの Sun Web コンソールは、ログインシェルとして /bin/sh を持つユーザー noacces のみがインストールできます。次のコマンドを使用してユーザーを追加します。

```
# useradd -u 60002 -g 60002 -d /tmp -s /bin/sh -c "No Access User" noaccess
```

問題: 簡単なクエリー定義の「table/view」ドロップダウンメニューにエントリがないが、テーブルがデータベースで定義されている。

対処方法: Oracle データベースを使用している場合に、通常、この問題が発生します。レポートモジュールのインストール時に、誤ったデータベーススキーマ名が指定されています。Oracle では、データベーススキーマ名は dbwriter によって使用されるデータベースユーザーの名前と同じになります。デフォルトの名前は arco_write です。Postgres では、データベーススキーマ名は public にしてください。

問題: 接続が拒否される。

対処方法: smcwebserver が停止している可能性があります。smcwebserver を起動または再起動してください。

問題: クエリーリストまたは結果リストが空である。

対処方法: 原因は次のいずれかが考えられます。

- データベースが停止状態です。データベースを起動または再起動してください。

- これ以上データベース接続を使用できません。データベースへの接続数を増やしてください。
- アプリケーションの構成ファイル内にエラーがあります。構成をチェックして、データベースユーザー、ユーザーパスワード、データベースの種類が誤ってないかどうか確認し、アプリケーションを再起動してください。
- クエリーを実行できません。クエリーディレクトリ `/var/spool/arco/queries` が空でない場合は、次のエラーが発生している可能性があります。
  - XML ファイルでのクエリーの構文が誤っています。XML パーサーからログファイルでエラーメッセージを確認してください。
  - ユーザー `noaccess` は、クエリーディレクトリへの読み取り権も書き込み権も持っていません。

問題: 使用可能なデータベーステーブルリストが空である。

対処方法: 原因は次のいずれかが考えられます。

- データベースが停止状態です。データベースを起動または再起動してください。
- これ以上データベース接続を使用できません。データベースへの接続数を増やしてください。
- アプリケーションの構成ファイル内にエラーがあります。構成をチェックして、データベースユーザー、ユーザーパスワード、データベースの種類が誤ってないかどうか確認し、アプリケーションを再起動してください。

問題: 選択可能なフィールドリストが空である。

対処方法: テーブルが選択されていません。リストから表を選択します。

問題: フィルタリストが空である。

対処方法: フィールドが選択されていません。1 つ以上のフィールドを定義してください。

問題: ソートリストが空である。

対処方法: フィールドが選択されていません。1 つ以上のフィールドを定義してください。

問題: 定義したフィルタが使用されていない。

対処方法: フィルタがアクティブでない可能性があります。未使用のフィルタを変更して、アクティブにしてください。

問題: 高度なクエリーの遅延結合が無視され、実行がエラーになる。

対処方法: 遅延結合マクロに構文エラーがあります。高度なクエリーの遅延結合マクロの正しい構文は、次のとおりです。

```
latebinding{attribute;operator}
latebinding{attribute;operator;defaultvalue}
```

問題: 前の画面に戻るために breadcrumb を使用したが、ログイン画面が表示された。

対処方法: セッションがタイムアウトになっています。再度ログインして、セッション時間を `app.xml` で増やしてください。

問題: ビュー構成を定義したが、デフォルト構成が表示されている。

対処方法: 定義されたビュー構成は表示設定になっていません。ビュー構成を開いて、使用するビュー構成を定義してください。

問題: ビュー構成を定義したが、最後の構成が表示されている。

対処方法: 定義されたビュー構成は表示設定になっていません。ビュー構成を開いて、使用するビュー構成を定義してください。

問題: クエリーの実行に時間がかかりすぎる。

対処方法: データベースから得られた結果が膨大です。結果に制限を設定するか、フィルタ条件を追加してください。

◆ ◆ ◆ 付 録 A

データベーススキーマ

この付録には、データベーススキーマの情報が表形式で記載されています。次の各項目について説明します。

スキーマテーブル

sge_job

sge_job テーブルは、各配列タスク (配列タスク番号 1 の付いた非配列ジョブに対して 1 レコード) および緊密に統合された並列ジョブ内で開始される各並列タスクに対して 1 つのレコードを格納します。

N1GE 6.1 システムでは、ジョブ、配列タスク、または並列タスクがスケジュールされるとすぐにレコードが作成されます。レコードは、ジョブの実行中に更新されます。

N1GE のジョブ、配列ジョブ、並列ジョブ、およびこれらの違いについての簡単な説明は、『Sun Grid Engine ユーザーズガイド』を参照してください。特に、用語集が導入として役立ちます。

列	タイプ	説明
j_id	Integer	一意のレコード識別子
j_job_number	integer	JOB_ID
j_task_number	integer	配列タスクの ID。
j_pe_taskid	text	緊密に統合された並列タスクの ID。

列	タイプ	説明
j_job_name	text	ジョブの名前 (スクリプト名、または発行オプション -N を使用した値のセット)
j_group	text	ジョブが実行された一次グループの UNIX グループ名。 グループテーブルを参照します。
j_owner	text	ジョブが実行された UNIX ユーザーアカウント。 ユーザーテーブルを参照します。
j_account	text	発行オプション -A を使用したアカウントの文字列セット。
j_priority	integer	発行オプション -p を使用した優先順位設定、またはキュー構成で割り当てられた優先順位設定。
j_submission_time	timestamp	ジョブ発行の時間。
j_project	text	プロジェクト (Sun Grid Engine、Enterprise Edition のみ) プロジェクトテーブルを参照します。
j_department	text	部署 (Sun Grid Engine、Enterprise Edition のみ) 部署テーブルを参照します。

sge_job_usage

`sge_job_usage` テーブルは、時間ごとのジョブのリソース使用率を保持します。

N1GE 5.3 システムでは、完了ジョブ、配列タスク、および並列タスクごとに 1 レコードのみ存在します。`ju_curr_time` 列はジョブの終了時刻を保持します (`sge_job` の `j_end_time`)。

N1GE 6.1 システムでは、オンラインの使用量も格納されます。これにより、1 つのジョブ、配列タスク、および並列タスクに対して複数のレコードが `sge_job` に格納されることになります。ジョブのリソース使用率は、時間ごとに監視でき (`ju_curr_time`)、ジョブごとの最終レコード、配列タスク、または並列タスクは、ア

カウンティングで使用可能な総使用量を保持します。このレコードの ju_curr_time は sge_job の j_end_time と等しくなります。

列	タイプ	説明
ju_id	Integer	一意のレコード識別子
ju_parent	Integer	sge_job テーブルへの参照
ju_curr_time	Integer	使用量に対する現在の時刻
ju_qname	text	ジョブが実行されたキューの名前。N1GE 6.1 システムでは、クラスタキューの名前になります。 キューテーブルのキューを参照します。
ju_hostname	text	ジョブが実行されたホストの名前。 ホストテーブルのホストを参照します。
ju_start_time	timestamp	ジョブの開始時間。
ju_end_time	timestamp	ジョブの終了時間。
ju_failed	integer	if != 0 は問題が発生したことを表します
ju_exit_status	integer	ジョブの終了状態
ju_granted_pe	text	そのジョブ用に選択された並列環境。
ju_slots	integer	ジョブに振り分けられたスロットの数。
ju_state	text	ジョブの状態
ju_ru_wallclock	integer	end_time から start_time を引いた値
ju_ru_utime	double	使用されたユーザー時間
ju_ru_stime	double	使用されたシステム時間
ju_ru_maxrss	integer	最大常駐セットサイズ
ju_ru_ixrss	integer	現在は 0
ju_ru_ismrss	integer	

列	タイプ	説明
ju_ru_idrss	integer	全体の常駐セットサイズ
ju_ru_isrss	integer	現在は 0
ju_ru_minflt	integer	物理 I/O を必要としないページフォルト
ju_ru_majflt	integer	物理 I/O を必要とするページフォルト
ju_ru_nswap	integer	スワップ
ju_ru_inblock	integer	ブロック入力操作
ju_ru_oublock	integer	ブロック出力操作
ju_ru_msgsnd	integer	送信済みメッセージ
ju_ru_msgrcv	integer	受信済みメッセージ
ju_ru_nsignals	integer	受信済み信号
ju_ru_nvcsw	integer	任意コンテキスト切り替え
ju_ru_nivcsw	integer	強制コンテキスト切り替え
ju_cpu	double	CPU 時間使用量 (秒)。
ju_mem	double	全体のメモリー使用量 (G バイト秒)。
ju_io	double	I/O 操作で転送されたデータ量。
ju_iow	double	I/O 待ち時間 (秒)。
ju_maxvmem	double	仮想メモリーの最大サイズ (バイト)。

sge_job_request

ジョブの要求に対するリソースを格納します。

現在、2 種類の要求 (qsub オプション) を扱うことができます。

1. -l リソース要求。例: -l arch=solaris,mem_total=100M
   各要求に対して 1 つのレコードが作成されます。
2. -q キュー要求。例: -q balrog.q
   「キュー」および要求内容を変数として含むレコードが 1 つ作成されます。

列	タイプ	説明
jr_id	Integer	一意のレコード識別子
jr_parent	Integer	sge_job テーブルへの参照
jr_variable	text	要求された複合変数の名前
jr_value	text	要求された値

sge_job_log

sge_job_log テーブルは、ジョブのログ情報を格納します。

列	タイプ	説明
jl_id	Integer	一意のレコード識別子
jl_parent	integer	sge_job テーブルへの参照
jl_time	unix timestamp	ジョブのログインエントリが生成された時間。
jl_event	text	
jl_job_number	integer	
jl_task_number	integer	
jl_pe_task_id	text	
jl_state	text	報告されたイベント後のジョブの状態
jl_user	text	イベントに対して処理を開始したユーザー
jl_host	text	イベントの処理を開始したホスト
jl_state_time	unix timestamp	ジョブが特定の状態にあった時間。以下の説明を参照。
jl_message	text	何が発生したのかを説明するメッセージ

sge_share_log

sge_share_log テーブルは、N1GE(EE) 共有ツリーの構成および使用率についての情報を格納します。

詳細は、N1GE の sharetree(5) マニュアルページを参照してください。

列	タイプ	説明
sl_id	Integer	共有のログレコードの一意の識別子
sl_curr_time	timestamp	現在の時刻
sl_usage_time	timestamp	使用時間
sl_node	text	共有ツリー内のノード名
sl_user	text	ユーザー名 (ジョブの所有者) ユーザーテーブルを参照します。
sl_project	text	プロジェクトの名前 プロジェクトテーブルを参照します。
sl_shares	integer	共有ツリーで構成された共有
sl_job_count	integer	共有ツリーポリシーとみなされるジョブの数
sl_level	double	このツリーレベル内の共有 (%)
sl_total	double	共有ツリー全体における共有の合計 (%)
sl_long_target_share	double	対象となる長期共有 (%)
sl_short_target_share	double	対象となる短期共有 (%)
sl_actual_share	double	実際の共有 (%)
sl_usage	double	複合的な使用状況。CPU、メモリー、および I/O のウェイトを構成できます。
sl_cpu	double	CPU 使用量 (秒)
sl_mem	double	全体のメモリー使用量 (G バイト秒)
sl_io	double	I/O 操作で転送されたデータ量。
sl_ltcpu	double	長期 CPU
sl_ltmem	double	長期メモリー

列	タイプ	説明
sl_ltio	double	長期 I/O

sge_host

sge_host テーブルには、クラスタ内のすべてのホストが一覧表示されます。

列	タイプ	説明
h_id	Integer	一意のホスト識別子
h_hostname	text	ホスト名。

sge_host_values

sge_host_values テーブルは、負荷平均などの変更される可能性のあるホスト変数の値を格納します。

さらに、時間平均や合計などの派生ホスト値も格納されます。

列	タイプ	説明
hv_hostname	text	ホストテーブルを参照します。
hv_time_start	timestamp	値の有効期間の開始時間。
hv_time_end	timestamp	値の有効期間の終了時間。
hv_variable	text	load_avg などの変数名。
hv_value	text	0.34 などの変数値。
hv_dvalue	double precision	数値としての変数値
hv_dconfig	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)。

sge_queue

sge_queue テーブルには、クラスタ内で構成されたすべてのキューが一覧表示されます。

列	タイプ	説明
q_id	Integer	一意のキュー識別子
q_qname	text	キューの名前
q_hostname	text	ホストの名前

sge_queue_values

sge_queue_values テーブルは、空きスロットの数などの変更される可能性のあるキュー変数の値を格納します。

さらに、時間平均や合計などの派生キュー値も格納されます。

列	タイプ	説明
qv_parent	integer	sge_queue テーブルの q_id を参照します。
qv_time_start	timestamp	値の有効期間の開始時間。
qv_time_end	timestamp	値の有効期間の終了時間。
qv_variable	text	slots などの変数名。
qv_value	text	5 などの変数値。
qv_dvalue	double precision	数値としての変数値
qv_dconfig	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)。

sge_department

データベースで参照されるすべての部署が一覧表示されます。

列	タイプ	説明
d_id	Integer	一意の部署識別子
d_department	text	部署の名前。

テーブル 9: sge_department テーブル

sge_department_values

sge_department_values テーブルは、変更される可能性のある部署関連の変数値を格納します。これらの値は現在、時間平均、合計などの派生値です。

列	タイプ	説明
dv_parent	integer	sge_department テーブルの d_id を参照します。
dv_time_start	timestamp	値の有効期間の開始時間。
dv_time_end	timestamp	値の有効期間の終了時間。
dv_variable	text	h_sum_jobs などの変数名。
dv_value	text	5 などの変数値。
dv_dvalue	double precision	数値としての変数値
dv_dconfig	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)。

sge_project

データベースで参照されるすべてのプロジェクトが一覧表示されます。

列	タイプ	説明
p_id	Integer	一意のプロジェクト識別子
p_project	text	プロジェクトの名前

sge_project_values

sge_project_values テーブルは、変更される可能性のあるプロジェクト関連の変数値を格納します。これらの値は現在、時間平均、合計などの派生値です。

列	タイプ	説明
pv_parent	integer	sge_queue テーブルの q_id を参照します。
pv_time_start	timestamp	値の有効期間の開始時間。
pv_time_end	timestamp	値の有効期間の終了時間。

列	タイプ	説明
pv_variable	text	h_avg_cpu などの変数名
pv_value	text	345.5 などの変数値
pv_dvalue	double precision	数値としての変数値
pv_dconfig	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)。

sge_user

データベースで参照されるすべてのユーザーが一覧表示されます。

列	タイプ	説明
u_id	Integer	一意のユーザー ID
u_user	text	ユーザーの名前。

sge_user_values

sge_user_values テーブルは、変更される可能性のあるユーザー関連の変数値を格納します。これらの値は現在、時間平均、合計などの派生キュー値です。

列	タイプ	説明
uv_parent	integer	sge_queue テーブルの q_id を参照します。
uv_time_start	timestamp	値の有効期間の開始時間。
uv_time_end	timestamp	値の有効期間の終了時間。
uv_variable	text	h_sum_cpu などの変数名
uv_value	text	23.2 などの変数値
uv_dvalue	double precision	数値としての変数値
uv_dconfig	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)。

sge_group

データベースで参照されるすべてのユーザーグループが一覧表示されます。

列	タイプ	説明
g_id	Integer	一意のグループ ID
g_group	text	グループの名前。

テーブル 15: sge_group テーブル

sge_group_values

sge_group_values テーブルは、変更される可能性のあるグループ関連の変数値を格納します。これらの値は現在、時間平均、合計などの派生値です。

列	タイプ	説明
gv_parent	integer	sge_queue テーブルの q_id を参照します。
gv_time_start	timestamp	値の有効期間の開始時間。
gv_time_end	timestamp	値の有効期間の終了時間。
gv_variable	text	h_sum_jobs などの変数名。
gv_value	text	53 などの変数値
gv_dvalue	double precision	数値としての変数値
gv_dconfig	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)。

定義済みビューのリスト

view_accounting

ジョブ、配列タスク、および緊密に統合されたタスクのアカウンティングレコードです。完了したジョブのみが含まれます。

列	タイプ	説明
job_number	integer	ジョブ番号
task_number	integer	配列タスクの ID
pe_taskid	text	緊密に統合された並列タスクの ID
name	text	ジョブの名前 (スクリプト名、または発行オプション -N を使用した値のセット)
groupname	text	ジョブが実行された一次グループの UNIX グループ名。グループテーブルを参照します。
username	text	ジョブが実行された UNIX ユーザーアカウント。ユーザーテーブルを参照します。
account	text	発行オプション -A を使用したアカウントの文字列セット
project	text	プロジェクト。プロジェクトテーブルを参照します
department	text	部署。部署テーブルを参照します
submission_time	timestamp	ジョブ発行の時間
start_time	timestamp	ジョブの開始時間
end_time	timestamp	ジョブの終了時間
wallclock_time	integer	end_time から start_time を引いた値
cpu	double	CPU 時間使用量 (秒)
io	double	I/O 操作で転送されたデータ量
iow	double	I/O 待ち時間 (秒)
maxvmem	double	仮想メモリーの最大サイズ (バイト単位)
wait_time	integer	start_time から submission_time を引いた値
turnaround_time	integer	end_time から submission_time を引いた値

view_job_times

view_accounting と同じですが、緊密に統合された並列ジョブのタスクは含まれていません。

view_jobs_completed

時間ごとの完了ジョブ (時間ごとに 1 レコード) が表示されます。

列	タイプ	説明
completed	integer	完了ジョブ
time	timestamp	ジョブの終了時間

view_job_log

ジョブのログ (発行、状態変更、ジョブ終了など) が表示されます。

列	タイプ	説明
job_number	integer	ジョブ番号
task_number	integer	配列タスクの ID
pe_taskid	text	緊密に統合された並列タスクの ID
name	text	ジョブの名前 (スクリプト名、または発行オプション -N を使用した値のセット)
username	text	ジョブが実行された一次グループの UNIX グループ名。グループテーブルを参照します
account	text	ジョブが実行された UNIX ユーザーアカウント。ユーザーテーブルを参照します
project	text	プロジェクト。プロジェクトテーブルを参照します
department	text	部署。部署テーブルを参照します

列	タイプ	説明
time	timestamp	ジョブのログエントリが生成された時間
event	text	記録されたイベント
state	text	報告されたイベント後のジョブの状態
initiator	text	イベントに対して処理を開始したユーザー
host	text	イベントの処理を開始したホスト
message	text	何が発生したのかを説明するメッセージ

view_department_values

部署に固有の変数です。

列	タイプ	説明
department	text	部署の名前
time_start	timestamp	値の有効期間の開始時間
time_end	timestamp	値の有効期間の終了時間
variable	text	h_sum_jobs などの変数名。
str_value	text	5 などの変数値
num_value	double precision	数値としての変数値
num_config	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)

view_group_values

グループに固有の変数です。

列	タイプ	説明
groupname	text	グループの名前
time_start	timestamp	値の有効期間の開始時間
time_end	timestamp	値の有効期間の終了時間
variable	text	h_sum_jobs などの変数名。
str_value	text	53 などの変数値
num_value	double precision	数値としての変数値
num_config	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)

view_host_values

ホストに固有の変数です。

列	タイプ	説明
hostname	**text**	ホスト名
time_start	timestamp	値の有効期間の開始時間
time_end	timestamp	値の有効期間の終了時間
variable	text	load_avg などの変数名
str_value	text	0.34 などの変数値
num_value	double precision	数値としての変数値
num_config	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)

view_project_values

プロジェクトに固有の変数です。

列	タイプ	説明
project	text	プロジェクトの名前

列	タイプ	説明
time_start	timestamp	値の有効期間の開始時間
time_end	timestamp	値の有効期間の終了時間
variable	text	h_avg_cpu などの変数名
str_value	text	345.5 などの変数値
num_value	double precision	数値としての変数値
num_config	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)

view_queue_values

キューに固有の変数です。

列	タイプ	説明
qname	text	キューの名前
hostname	text	ホストの名前
time_start	timestamp	値の有効期間の開始時間
time_end	timestamp	値の有効期間の終了時間
variable	text	slots などの変数名
str_value	text	5 などの変数値
num_value	double precision	数値としての変数値
num_config	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)

view_user_values

ユーザーに固有の変数です。

列	タイプ	説明
username	text	ユーザーの名前

列	タイプ	説明
time_start	timestamp	値の有効期間の開始時間
time_end	timestamp	値の有効期間の終了時間
variable	text	h_sum_cpu などの変数名
str_value	text	23.2 などの変数値
num_value	double precision	数値としての変数値
num_config	double precision	消費可能リソースの場合: 消費可能リソースで使用可能な最大値 (構成した値)

派生値のリスト

データベースに格納されている派生値によって、クエリーの処理時間を大幅に短縮できます。レポートデータベースには、時間ごとの集計値 (合計、平均、最小、最大) が含まれます。1 年後といったある程度の期間のあとに、これらの値を、日単位、週単位、月単位などの値にまとめることもできます。

次の派生値が生成されます。

表	**variable**	説明
sge_host_values	h_sum_cpu、d_sum_cpu、m_sum_cpu	ホストごとおよび時間、日、月ごとの CPU 使用量
sge_user_values	h_sum_cpu、d_sum_cpu、m_sum_cpu	ユーザーごとおよび時間、日、月ごとの CPU 使用量
sge_group_values	h_sum_cpu、d_sum_cpu、m_sum_cpu	グループごとおよび時間、日、月ごとの CPU 使用量
sge_department_values	h_sum_cpu、d_sum_cpu、m_sum_cpu	部署ごとおよび時間、日、月ごとの CPU 使用量
sge_project_values	h_sum_cpu、d_sum_cpu、m_sum_cpu	プロジェクトごとおよび時間、日、月ごとの CPU 使用量
sge_host_values	h_avg_load	ホスト負荷の平均
sge_host_values	h_max_load	ホスト負荷の最大

すべての派生値の生成に関する規則は、構成ファイルで指定できます。

用語集

アクセスリスト	キュー、ホストなどのリソースへのアクセスを許可または拒否されたユーザーおよびUNIX グループのリスト。ユーザーやグループは複数のアクセスリストに所属できます。また、同じアクセスリストを複数のコンテキストで使用できます。
移行	ジョブの実行が再開される前に、チェックポイント設定ジョブを別のホストへ移すプロセス。
一時停止	実行中のジョブを実行ホストに格納して停止するプロセス。チェックポイント設定のように、ジョブが「中止」されるわけではありません。ジョブを一時停止しても、スワップメモリーやファイル空間などの一部のリソースは引き続き消費されます。
エンタイトルメント	「共有」と同義。特定のジョブ、ユーザー、ユーザーグループ、またはプロジェクトによって消費される予定のリソースの量を示します。
オペレータ	管理者と同じコマンドを実行できるユーザー。ただし、構成を変更することはできません。オペレータの役割は、操作を維持することです。
管理者	あらゆる局面で Grid Engine ソフトウェアを操作できるユーザー。マスターホストおよび管理ホストとして宣言されるその他のマシンのスーパーユーザーであり、管理者の特権を持っています。管理者の特権は、root ユーザー以外にも割り当て可能です。
管理ホスト	Grid Engine システムの管理作業の実行権を持つホスト。
機能ポリシー	ジョブ、ユーザー、ユーザーグループ、およびプロジェクトに対して、特定のレベルの重要度を割り当てます。これにより、たとえば、優先順位の高いプロジェクトとそのプロジェクトのすべてのジョブには、優先順位の低いプロジェクトより高いリソース配分が割り当てられます。
キャンパスグリッド	組織内の複数のプロジェクトまたは部署による計算リソースの共有を可能にするグリッド。
共有	「エンタイトルメント」と同義。特定のジョブ、ユーザー、またはプロジェクトによって消費される予定のリソースの量を表します。
共有ツリー	共有ベースポリシーの階層定義。

共有ベースポリシー	ユーザー、プロジェクト、および任意のグループのエンタイトルメントを階層的に定義するポリシー。たとえば会社は、課、部門、部門内でアクティブになっているプロジェクト、これらのプロジェクトに関わるユーザーグループ、そしてこのユーザーグループに所属するユーザーに区分できます。共有ベースの階層を「共有ツリー」と呼びます。共有ツリーを定義すると、Grid Engine ソフトウェアは自動的にエンタイトルメントを実装します。
クラスタ	「ホスト」と呼ばれるマシンの集合。Grid Engine システム機能は、このクラスタ上にあります。
クラスタキュー	同時実行可能な複数のジョブのクラスを格納するコンテナ。キューは、特定のジョブ属性、たとえば移動可能なジョブかどうかを特定します。実行中のジョブは、実行が終了するまでキューに関連付けられています。キューに関連付けられていることで、ジョブで発生する事象の一部が影響を受けます。たとえば、キューが一時停止されると、そのキューに関連付けられたすべてのジョブが一時停止されます。
クラスタグリッド	もっとも単純な形式のグリッド。連携して機能し、単一のプロジェクトまたは部署内のユーザーに単一のアクセスポイントを提供する複数のコンピュータホストで構成されます。
グリッド	タスクを実行する複数の計算リソースの集合。ユーザーは、グリッドを単一の計算リソースとして使用します。
グループ	UNIX グループ
グローバルグリッド	複数のキャンパスグリッドの集合。組織の境界を越え、非常に大規模な仮想システムを構築することができます。
コンプレックス	リソース属性定義のセット。キュー、ホスト、またはクラスタ全体に関連付けることができます。
実行ホスト	Grid Engine システムジョブの実行権を持つシステム。キューインスタンスをホストし、実行デーモン `sge_execd` を実行します。
使用率	「消費リソース」の別称。消費される CPU 時間、利用に伴って占有されるメモリー、I/O 実行量の管理者が構成できる値です。
ジョブ	ユーザーからの要求。グリッドに計算リソースを要求します。
ジョブクラス	何らかの共通点があり、同じように取り扱われる複数のジョブのセット。ジョブクラスは、類似したジョブ群に共通する同一の要件と、これらのジョブに適したキューの特性によって定義されます。
所有者	所有するキューの一時停止、再開、無効化、有効化を実行できるユーザー。通常、ユーザーは、自分のワークステーション上のキューのインスタンスの所有者になります。
セル	それぞれ構成およびマスターマシンが異なる独立したクラスタ。複数の独立した管理ユニットを疎結合するために使用されます。
ソフトリソース要件	ジョブが必要とするリソース。ただし、ジョブの開始前に割り当てる必要はありません。利用できるときに割り当てます。「ハードリソース要件」の対語になります。

対話型ジョブ	`qrsh`、`qsh`、または`qlogin`コマンドで開始されるセッション。ユーザーとの対話用に*xterm*ウィンドウを表示します。または、リモートログインセッションのようなものを提供します。
チェックポイント設定	あるジョブの実行ステータスをチェックポイントに保存する手続き。情報とすでに完了した作業の内容を維持したまま、中止したジョブをあとで再開することができます。中止したジョブの実行を再開する前にチェックポイントを別のホストに移した場合は「移行」と呼びます。
チェックポイント設定環境	特定のチェックポイント設定方法に関連付けられたイベント、インタフェース、およびアクションを定義するGrid Engineシステムの構成要素。
チケット	リソース配分定義の一般的な単位。ジョブ、ユーザー、プロジェクト、またはその他のコンポーネントが持つチケットの配分が多いほど、より重要性が高いということになります。たとえば、ほかのジョブの2倍チケットを持っているジョブは、ほかのジョブの2倍のリソースを使用する権利があります。
ハードリソース要件	ジョブの開始前に割り当てなければならないリソース。対語として「ソフトリソース要件」があります。
配列ジョブ	それぞれ独立した同一のタスクの範囲を構成するジョブ。 各タスクは、単独のジョブとして機能します。配列ジョブタスクは、一意のタスク識別子 (整数値) によってのみ区別されます。
発行ホスト	バッチジョブのみを発行し、制御するホスト。 発行ホストにログインしているユーザーは、`qsub`コマンドでジョブを発行し、`qstat`コマンドでジョブのステータスを制御します。また、Grid Engineシステム OSF/1 Motifグラフィカルユーザーインタフェース`QMON`を利用できます。
バッチジョブ	UNIXシェルスクリプト。ユーザーの介入なしで実行できます。また、端末へのアクセスも不要です。
部署	同様の職務を持つと見なされるユーザーおよびグループのリスト。Grid Engineシステムのスケジューリングポリシーより優先されます。ユーザーやグループが複数の部署に所属することはできません。
プロジェクト	Grid Engineシステムプロジェクト。
並列環境	Grid Engineシステム構成。並列ジョブを正常に処理するためにGrid Engineソフトウェアに必要なインタフェースを定義します。
並列ジョブ	複数の密接に関連したタスクで構成されるジョブ。タスクは、複数のホストに分散されます。並列ジョブは、通常、共有メモリーやメッセージ受け渡し (MPI、PVM) などの通信ツールを使用して、タスクの同期や関連付けを行います。
ホスト	Grid Engineシステム機能が置かれたシステム。
ポリシー	管理者がGrid Engineシステムの動作を定義するために使用する規則および構成のセット。ポリシーは、システムによって自動的に実装されます。

マスターホスト	クラスタアクティビティー全体の中枢。マスターデーモン `sge_qmaster` とスケジューラデーモン `sge_schedd` を実行します。デフォルトでは、マスターホストが管理ホストでもあり発行ホストでもあります。
ユーザー	1 台以上の発行ホストおよび実行ホストでの有効なログイン ID を持っている場合、グリッドにジョブを発行し、実行することができるユーザー。
ユーザーセット	「アクセスリスト」または「部署」。
優先順位	あるジョブとその他のジョブを比較したときの重要度のレベル。
優先ポリシー	一般に、機能ポリシーおよび共有ベースポリシーのリソースエンタイトルメント管理の自動化を無効にするために使用されます。クラスタ管理者は、ポリシーの自動実装を変更し、ジョブ、ユーザー、ユーザーグループ、およびプロジェクトを無効にすることができます。
リソース	実行ジョブによって消費または占有される計算デバイス。典型的な例として、メモリー、CPU、I/O 帯域幅、ファイル空間、ソフトウェアライセンスなどがあります。

索引

R

S

し

す

せ

そ

よ

り

ろ