グローバルスケジューリングのための計算資源予約管理機構

中田秀基[†] 竹房あつ子[†] 大久保克彦[†,††]
工藤知宏[†] 田中良夫[†] 関口智嗣[†]

グリッド上で複数の資源の同時確保を実現する方法の 1 つとして事前予約を行う方法がある．これを実現するためには，同時確保の実現に求められる協調プロトコルを実装した事前予約機構を，グリッド上の各資源が持つことが必要となる．我々は，同時確保のプロトコルとして分散トランザクションで一般に用いられる 2 相コミットプロトコルを採用し，これを実装した計算資源予約管理機構 PluS を開発した．PluS は，既存のローカルキューイングシステムである TORQUE もしくは Grid Engine と協調動作し，事前予約機能を提供する．既存ローカルキューイングシステムに事前予約機能を追加する手法としては，1）ローカルキューイングシステムのスケジューリングモジュールを完全に置き換える方法，2）予約をキューとして実現し，外部からキューを操作することで予約を実現する方法，の 2 つがある．我々は，この両者に関して予約機能を設計，実装し評価した．その結果，前者はオーバヘッドが少なくポリシ実現の自由度が高いこと，後者は既存スケジューリングモジュール機能の再実装が必要ないため実装コストが小さいことが分かった．

An Advance Reservation-based Computation Resource Manager for Global Scheduling

Hidemoto Nakada,[†] Atsuko Takefusa,[†] Katsuhiko Ookubo,[†,††]
Tomohiro Kudoh,[†] Yoshio Tanaka[†] and Satoshi Sekiguchi[†]

Advance Reservation is one possible way to enable resource co-allocation on the Grid. This method requires all the resources to have advance reservation capability as well as coordination protocol support. We employed 2-phased commit protocol as a coordination protocol, which is common in the distributed transaction area, and implemented an Advance Reservation Manager called **PluS**. PluS works with existing local queuing managers, such as TORQUE or Grid Engine, and provides users advance reservation capability. To provide the capability, there are two implementation methods; 1) completely replace the scheduling module of the queuing manger, 2) represent reservation as a queue and control the queues using external interface. We designed and implemented a reservation manager with both way, and evaluated them. We found that the former has smaller overhead and allows arbitrary scheduling policy, while the latter is much easier to implement with acceptable response time.

1. はじめに

グリッドの目的の 1 つとして，ネットワーク上に散在する資源を同時に確保，使用する大規模な計算がある[1)]．資源の同時確保を実現する方法はいくつか考えられるが，最も直接的な方法は各資源で事前予約を行う方法である．各資源上のローカル資源管理機構が事前予約機構を持ち，全体を統合するスーパスケジューラから，同時刻に事前予約を行うことで，その時刻にすべての資源を同時に確保する．

この際に留意するべき点の 1 つとして，スーパスケジューラとローカル資源管理機構間のプロトコルがある．複数資源に対する同時予約操作は，本質的に分散トランザクションであり，操作失敗時の挙動を保障するためには，スーパスケジューラとローカル資源管理機構の間で適切なプロトコルを用いる必要がある．

したがって，事前予約を用いた同時確保を実現するためには，1) 同時確保をサポートするスーパスケジューラ，2) 事前予約機構を持つローカル資源管理機構，3) 両者を結ぶ適切なプロトコルを用いた外部インタフェース，の 3 つが必要となる．

我々は，1) に関してネットワークと計算機を同時確保することのできるスーパスケジューラ[2)] を，3) に関

† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology（AIST）
†† 数理技研
SURIGIKEN Co., Ltd.

してはWSRF（Web Services Resource Framework）をベースとした予約インタフェース[3]を提案している．本稿では2)として開発中の事前予約管理機構PluS[4]について，その設計と実装を詳述する．PluS事前予約機構をWSRFベースの予約プロトコルでラップすることで，外部インタフェースを持つローカル資源管理機構を構成し，これとスーパスケジューラが協調動作することで資源の同時確保を実現する[5]．

PluSは，同時確保のプロトコルとして分散トランザクションで一般に用いられる2相コミットプロトコル[6]をサポートする事前予約管理機構である．PluSは，既存のローカルキューイングシステムであるTORQUE[7]およびGrid Engine[8]と協調動作し，2相コミット機構をサポートした事前予約機能を提供する．

既存ローカルキューイングシステムに事前予約機能を追加する手法としては，1) ローカルキューイングシステムのスケジューリングモジュールを完全に置き換える方法，2) ローカルキューイングシステムのスケジューリングモジュールをそのまま使用し，外部からキューを操作することで予約を実現する方法，がある．我々は，この両者に関して予約機能を設計，実装し評価した．その結果，前者はオーバヘッドが少なくポリシ実現の自由度が高いこと，後者は既存スケジューリングモジュール機能の再実装が必要ないため実装コストが小さいことが分かった．

本稿の以下の構成を以下に示す．2章で事前予約と2相コミットプロトコルの必要性に関して述べる．3章にPluS事前予約機構の設計を示す．4章に実装の詳細を示す．5章に2つの手法の評価を示す．8章はまとめである．

2. 同時予約と2相コミットプロトコル

2.1 同時予約の問題点

複数資源の予約においてコミットプロトコルが必要であることを示すために，複数の資源（A，B）で予約した時間帯を変更することを考えてみよう．単純な実装では，資源AおよびBに対して順番に時間帯変更のリクエストを出すことになるだろう．この場合，資源Aで時間帯の変更に成功したのち，資源Bで変更に失敗した場合に，問題が生じる可能性がある．期待される動作は，資源Aでの変更をキャンセルし，予約をいったんもとの時間帯に戻したうえで，変更の失敗を上位レイヤに返すことである．

しかし，資源Aではすでに予約時間帯が変更されているため，変更をキャンセルする際には，再度時間帯を変更しなければならない．しかし，変更時にいったん開放した資源をキャンセル時に再度取得できる保証はない．このため，最悪の場合，時間帯の変更に失敗しただけでなく，もともと保持していた時間帯も維持できないことになってしまう．

2.2 2相コミット

複数資源の同時予約は，本質的に分散トランザクションである．分散トランザクションに関しては長い研究の歴史があり[6]，いくつものプロトコルが提案されている．その最も基本的なプロトコルが2相コミットである．

2相コミットの本質は，1回目の通信で操作を確定せず，2回目の通信まで動作の確定を引き伸ばすことにある．スーパスケジューラは，予約などの操作を行う際にローカルスケジューラに対して，まず，コミットリクエストを行う．すべてのローカルスケジューラがコミット可能であると返答してきた場合のみ，コミット操作を行い，予約を確定する．こうすることで前述した問題は生じなくなる．

現在，いくつかのキューイングシステム，外部スケジューリングモジュールが事前予約機能には対応しているが，2相コミットに対応したものはない．

3. PluS事前予約機構の設計

3.1 バッチキューイングシステムの一般的な構造

バッチキューイングシステムは一般にヘッドノードと計算ノードの2種類のノードから構成される．一般にヘッドノードはシステム全体に1つだけ存在し，ユーザはこのノードにジョブをサブミットする．計算ノードは，ヘッドノードからの指令に応じて計算を行う．もちろん，ヘッドノードの機能と計算ノードの機能は排他ではなく，1つの物理ノードが双方の機能を果たす場合もある．

ヘッドノードの機能は，一般に**マスタモジュール**と**スケジューリングモジュール**の2つのモジュールで実現される．計算ノードの機能は**ノード管理モジュール**で実現される（**図1**）．

3つのモジュールの機能は以下のとおりである．

- **マスタモジュール**
  マスタモジュールは，ユーザからの通信を直接受け付けるデーモンである．このデーモンの役割は多岐にわたるが，1) ジョブキューの管理，2) 計算ノード群の遠隔管理，3) スケジューリングの開始と，スケジューリング結果の実行，の3つにまとめることができる．
  マスタモジュールは，ユーザからのジョブ管理コ

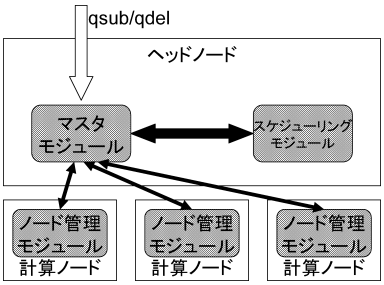


図 1 バッチキューイングシステムの一般的な構成
Fig. 1 Generic configuration of queuing systems.

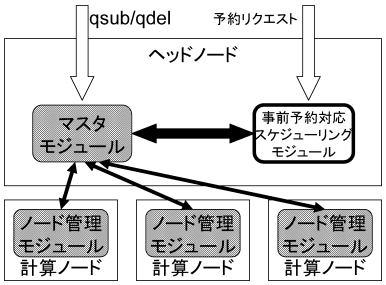


図 2 スケジューリングモジュール置換法
Fig. 2 Replacing scheduling module method.

マンドを受け付けジョブのキューイングを行う．また，計算ノード上のノード管理モジュールと通信し，計算ノードの状態を把握する．これらの情報をスケジューリングモジュールに送信し，スケジューリングを主導し，また得られたスケジューリング結果に基づいて，ジョブをノード管理モジュールに送信して，実行する．

TORQUE では pbs_server が，Grid Engine では sge_qmaster が，このモジュールである．

- **スケジューリングモジュール**

  スケジューリングを実際に行うモジュールである．マスタデーモンからノードおよびジョブの情報を取得し，それに基づいて，どのジョブをどのノードで実行するべきかを決定する．

  TORQUE では pbs_sched が，Grid Engine では sge_schedd がこのモジュールである．

- **ノード管理モジュール**

  ノード管理モジュールは計算ノードを管理するモジュールである．定常的に計算機のロードアベレージ，ディスクスペースなどの情報を取得し，マスタデーモンに報告するノードのモニタリングのほか，割り当てられた，ジョブの起動，終了，モニタリングも行う．

  TORQUE では pbs_mom が，Grid Engine では sge_execd が，このモジュールである．

3.2 ジョブキュー

キューイングシステムにおけるジョブキューは，複数のジョブを FIFO（First In First Out）で保持する構造である．ユーザがジョブキューに投入したジョブは，FIFO で取り出され，スケジューリングの対象となる．

多くのキューイングシステムでは，複数のキューを保持することができる．複数のキューに対してそれぞれ使用可能なノード数の上限を指定することで，1 つのキューイングシステムを事実上独立した複数の計算ノード群として運用することも広く行われている．

また，多くのキューイングシステムでは，特定のユーザグループからのサブミットのみを受け付けるよう，キューを設定することも可能である．

3.3 予約管理機構の実装手法

上述のような構造を持つバッチキューイングシステムに事前予約機能を付加する方法は，1) スケジューリングモジュールを完全に独自のものに置き換える，2) ジョブキューを外部から制御する，3) スケジューリングモジュールのソースコードに手を加える，の 3 つが考えられる．

このうち 3) は，バッチキューイングシステム本体のソースコードをメンテナンスしている組織にとっては比較的実現が容易である．しかしこの方法は，第三者が行うには，ソースコードの解析に多大なコストがかかるうえ，バッチキューイングシステムのバージョンアップへの追従が非常に困難であるなど問題があり，現実的ではない．そこで，我々は，1) と 2) に着目し，それぞれ設計と実装を行った．

3.3.1 スケジューリングモジュール置換法

この手法では，スケジューリングモジュールを，事前予約機能を持つ独自のもので置き換える．ユーザからの事前予約のリクエストはスケジューリングモジュールが受け付け，なんらかの予約 ID を返却する．ユーザは，ジョブをサブミットする際に，その予約 ID をジョブの属性として付加する．マスタモジュールは，スケジューリングモジュールにジョブのスケジュールを依頼するが，その際にこの属性値もスケジューリングモジュールに渡される．スケジューリングモジュールは，属性値（予約 ID）を参照して，特定のタイムスロットでだけそのジョブを実行する（**図 2**）．

この手法は，実装の自由度において最も望ましい．スケジューリングモジュールというキューイングシステムの心臓部を置き換えるため，実装者は，既存スケジューリングモジュールの構造に縛られることなく任

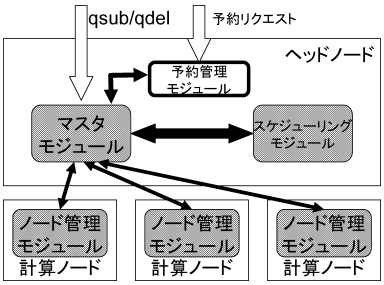


図 3 キュー制御法
Fig. 3 Queue control method.

意のポリシを設定することができる．

反面，この手法にはいくつかの欠点がある．1 つは，スケジューリングモジュールを置換するため，マスタモジュールとスケジューリングモジュール間の通信プロトコルが既知である必要があることである．さらに，キューイングシステムのバージョンアップにともない，プロトコルが変更されてしまった場合には，これに追従しなければならない．

もう 1 つの問題点は，既存スケジューリングモジュールの機能を再実装しなければならないことである．すでに運用されているキューイングシステムに対して事前予約機能を提供する場合，ユーザに違和感なくスケジューリングモジュールを受け付けてもらうためには，少なくとも事前予約機能を使用しない際のキューイングシステムの挙動が変化しないようにしなければならない．しかし，既存キューイングシステムの実現する豊富な機能を完全に再実装することは，記述量の観点から容易ではない．

3.3.2 キュー制御法

この手法では，事前予約機能は独立した予約管理モジュールとして実現され，予約タイムスロットはキューとして実現される．ユーザからの事前予約リクエストはこのモジュールが受け付ける（**図 3**）．

予約管理モジュールは，ユーザからのリクエストに応じて，動的にキューを生成し，そのキューの名前をユーザに返却する．生成にはキューイングシステムが用意しているコマンドラインインタフェースを利用する．キューは不活性状態（サブミットは可能だがジョブは実行されない状態）で生成される．また，そのユーザ（および明示的に指定されたユーザ）のみがサブミットできるよう設定される．

予約開始時刻が来ると，予約管理モジュールはそのキューを活性化すると同時に，他のキューが予約対象ノードを使用できないよう，他のキューのノード割当てを制御する．

この方法の最大のメリットは，既存スケジューリングモジュールをそのまま利用するため，事前予約機能を用いない際のキューイングシステムの挙動が変化しないことが保障できることである．また，スケジューリングモジュールの機能を再実装しなくてすむため，記述量が少ないこともメリットである．さらに，公開されているコマンドラインインタフェースを使用して，キューを外部から制御するだけで実現できるため，キューイングシステムのバージョンアップにともなう内部プロトコルの変更による影響を受けることがない．もちろんコマンドのインタフェースが変更されてしまうことも考えられるが，その場合でも追従ははるかに容易であることが期待できる．

デメリットとしては，キューイングシステムのキューに一定の機能が要求されることがあげられる．具体的にはキューを特定のユーザ群と計算ノード群の双方に結び付ける機能が必要となる．

また，キューを外部から制御するコストも問題となる可能性がある．制御には，コマンドラインインタフェースを利用する．事前予約にも予約時刻になった際のキューの変更にも，数回のキュー操作が必要となるが，これをそれぞれコマンドとして実行しなければならず，操作に時間がかかることが予想される．

もう 1 つのデメリットは，実現できる予約とスケジューリングポリシに制約が生じることである．スケジューリングモジュールを置換する方法では，事実上任意のポリシを実現することが可能であるが，キューを操作する方法では，キューイングシステムのデフォルトのスケジューリングモジュールのポリシと矛盾のない範囲でポリシを設定しなければならない．

4. PluS 事前予約機構の実装

4.1 PluS 事前予約機構の概要

我々は前節で示した実装方針に基づき，PluS 事前予約機構を実装した．PluS 事前予約機構は一部の通信ルーチンを除いては Java 言語で実装されており，TORQUE および Grid Engine と協調動作することができる．TORQUE に対しては前述のスケジューリングモジュール置換法で，Grid Engine に対しては 2 つの手法双方で実装されている．ただしスケジューリングモジュール置換法での実装は，オリジナルのスケジューリングモジュールの機能を完全に再現してはおらず，サブセットとなっている．また，TORQUE に対して，キュー制御法で実装することができなかったのは，TORQUE のキュー機構がキューと特定の計算ノード群に対応付ける機能を提供しないためである．

表 1 予約コマンドラインインタフェース
Table 1 Command line interface for reservation management.

コマンド名	機能	引数	出力
plus_reserve	予約をリクエストする	[-R ホスト名] [-T] [-U ユーザリスト] -s 開始時刻 -e 終了時刻 -n ノード数	予約 ID
plus_cancel	予約をキャンセルする	[-R ホスト名] [-T] -r 予約 ID	
plus_modify	予約の修正を行う	[-R ホスト名] [-T] [-U ユーザリスト] [-s 開始時刻] [-e 終了時刻] [-n ノード数] -r 予約 ID	
plus_status	予約の状態を表示	[-R ホスト名] [-r 予約 ID]	予約状態
plus_commit	予約操作をコミットする	[-R ホスト名] -r 予約 ID	
plus_abort	予約操作をアボートする	[-R ホスト名] -r 予約 ID	

PluS 事前予約機構に対する予約リクエストなどの操作はコマンドラインインタフェースから行う．コマンドラインインタフェースの一覧を**表 1** に示す．いくつかのコマンドに `-T` オプションがあるが，これは 2 相トランザクションモードでの実行を意味する．このオプションを指定して実行した場合，操作は仮操作となり，コミット・アボートを待つ状態となる．これに引き続き，`plus_commit` もしくは `plus_abort` を発行することで操作が完結する．

個々のコマンドは小さいシェルスクリプトによるラッパと Java で記述された本体とで構成されている．Java で記述されたコマンドは，PluS 予約モジュールと RMI で通信する．

PluS 事前予約機構は予約テーブルを保持・管理する．ヘッドノードのリブートや停止に備えて，この予約テーブルの情報を永続化する必要がある．我々は，永続化のために Java ネイティブのオブジェクトデータベースである db4objects [9] を用いた．db4objects は，非常に簡便なインタフェースを提供しており，JDBC を用いて関係データベースをアクセスする方法と比較してはるかに容易に実装することができた．

4.2 PluS の事前予約ポリシ

現在の PluS においては事前予約が最重視される設定となっている．事前予約は通常のキューイングジョブとは完全に独立しており，キューイングジョブの存在に影響を受けない．事前予約が影響を受けるのは他の事前予約ジョブからのみである．

すなわち，通常キューにジョブが存在する場合でも，他の事前予約ジョブがなければ，事前予約は成功する．たとえば，10 分後から 1 時間のタイムスロットを予約することを考えてみよう．この時間帯に他の事前予約がなければ，たとえすべてのノードで現在通常ジョブを実行中であっても事前予約は成功する．10 分後には，必要な数の通常ジョブを停止し，事前予約のためにノードを空ける．

このポリシは，他の資源との協調動作を最優先で考え，ローカルな通常ジョブをバックフィルとして考えることから導き出されている．

4.3 2 相コミットの実装

本節では，PluS 事前予約機構における 2 相コミットの実装法を示す．PluS 事前予約機構では，予約時，予約変更時，予約キャンセル時に 2 相コミットを行うことができる．このうち，予約時および予約キャンセル時の実装は容易である．すなわち，予約時にはコミットを待たずに通常の予約操作を行い，アボートした場合には予約を取り消せばよい．また，予約キャンセル時には，操作がコミットされるまで実際の操作は行わず，コミットされた際に初めてキャンセルの操作を行えばよい．

問題となるのは予約変更時の動作である．リクエストからコミットされるまでの間は，アボートしてもとの状態に戻ることができるように変更前リクエストを満たす資源と，コミットして変更後のリクエストを満たすだけの資源の両方を確保しておかなければならない．したがって，リクエスト時には変更前と変更後のリクエストの和集合を確保し，コミット時には，変更後必要なくなる資源を解放，同様にアボート時には，変更前の状態に戻すと同時に変更のために新たに確保した資源を解放する．

4.4 予約に対するノードの割当て

予約に対してノードを割り当てる際には，ノードの選択が重要である．単純にノード番号の順に割り振るなどの単純な戦略では，論理的には可能な予約が実現できなくなってしまう．

図 4 上に示すのはそのような割当ての例である．ここで N は管理対象のノード数である．予約 1，予約 2，予約 3 がそれぞれ 1 ノードを使用するとすると，予約 1 終了後，予約 3 終了までは，つねに $N-1$ ノードが使用可能であるが，この期間に $N-1$ 台を使用する予約を入れることはできない．なぜなら，途中で使用可能なノードが変わってしまうからである．

図 4 下に示すように，予約 3 を配置すれば，上述のような予約をすることが可能になる．ポイントは可能な限り，ノードを連続して使用することである．これ

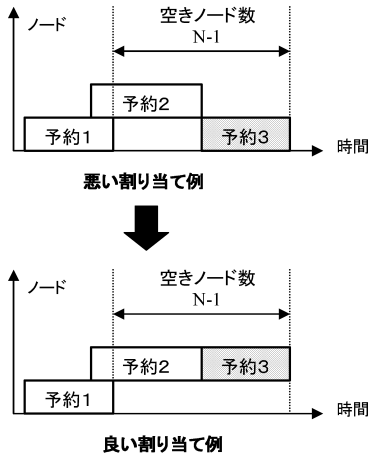


図 4 予約ノードの配置
Fig. 4 Node allocationf for reservation.

を実現するべく，下記のノード選択アルゴリズムを用いた．

(1) すでに入っている予約を終了時刻でソートする．
(2) 終了時刻が遅い予約から順に使用ノードを引き継ぐ．こうすることで，可能な限りノードを連続に使用することになる．
(3) (2) で不足であれば，未使用の番号の若いノードから使用する．

4.5 TORQUE 向け PluS

TORQUE は OpenPBS の亜種の 1 つである．上述したように，TORQUE のキュー機構では，キュー操作による事前予約の実装が不可能だったため，スケジューリングモジュールを完全に置換する方法での実装を行った．

4.5.1 TORQUE 向け PluS の実装

TORQUE のマスタモジュール（pbs_server）とスケジューリングモジュール（pbs_sched）との間の通信は，比較的平易なテキストベースのプロトコルとなっている．我々はこのプロトコルを解析し，このプロトコルを用いて通信するスケジューリングモジュールを作成した．

4.5.2 TORQUE 向け PluS のユーザ認証

TORQUE では，1023 以下の特権ポートを用いたユーザ認証を行う．具体的には setuid された pbs_iff と呼ばれるコマンドが用意されており，通信元は，このコマンドを fork/exec する．このコマンドが特権ポートから通信先に親コマンドのユーザ名を報告することで通信元プロセスのユーザを認証するのである．PluS のコマンドラインインタフェースもこれと同じ機構を用いてユーザ認証を行う．

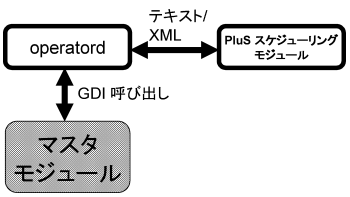


図 5 operatord による通信の中継
Fig. 5 Operatord.

4.6 Grid Engine 向け PluS の実装

Grid Engine は Sun Microsystems が開発したローカルスケジューラで，数多くのプロジェクトで広く用いられている．Grid Engine に対して，スケジューリングモジュール置換法とキュー制御法の双方で実装を行った．

4.6.1 スケジューリングモジュール置換法による実装

スケジューリングモジュールを置換するには，スケジューリングモジュールとマスタモジュールの間の通信プロトコルを PluS のモジュールで解釈する必要がある．TORQUE の場合はプロトコルが比較的単純であったため，プロトコルを解析し PluS のスケジューリングモジュールに直接組み込むことができた．しかし，Grid Engine のプロトコルはそれほど単純ではないため，このアプローチは難しい．

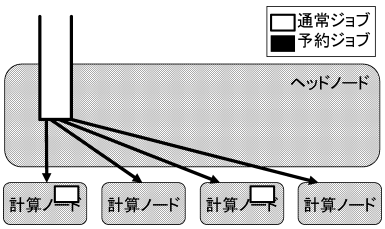
幸いなことに，Grid Engine は GDI（SUN Grid Engine Database Interface）と呼ばれる一種の通信ライブラリを C 言語で提供している．しかし PluS のモジュールは Java で記述されているため，このライブラリを直接組み込むことはできない．我々は operatord と呼ぶ中継デーモンを GDI を用いて C 言語で実装することでこの問題を回避した．

operatord は Java で記述された PluS スケジューリングモジュールと行単位のテキスト通信と XML からなる単純なプロトコルで通信し，GDI 呼び出しで Grid Engine のマスタモジュールと通信する（**図 5**）．

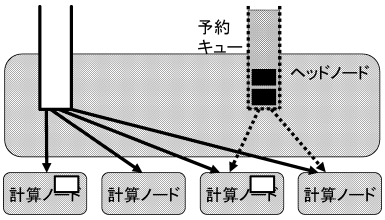
4.6.2 キュー制御法による実装

図 6 に Grid Engine におけるキュー制御法による PluS 事前予約機構の動作を示す．

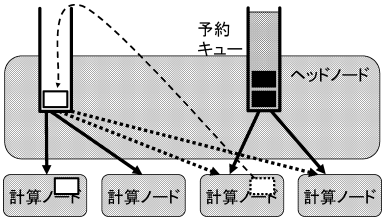
(1) 予約のリクエストを受けると，サスペンド状態のキューを作成し，そのキュー名を予約 ID として返却する．
(2) ユーザはキューに対してジョブをサブミットする．予約開始時刻が到来するまでは，キューがサスペンド状態であるため，ジョブは実行されない．
(3) 予約開始時刻が来たら，予約に対応したキュー



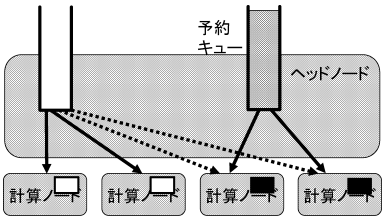
(A) 初期状態．左にデフォルトキューがあり，すべての計算ノードでの実行権利を保持している．



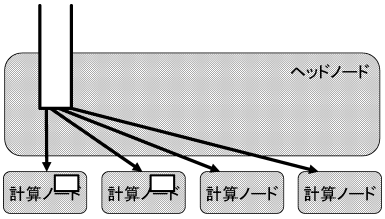
(B) 予約操作完了後．右側に予約を示すキューが新たにできている．右 2 つのノードが割り当てられているが，まだ活性化されていない．



(C) 予約時刻となった．実行中の通常ジョブは強制的に停止されデフォルトキューに戻される．予約キューが活性化される．一方，デフォルトキューからは右 2 つのノードへジョブを投入できなくなる．



(D) 予約時間中．右 2 つのノードで予約ジョブが実行される．一方，停止されたジョブは左から 2 番目の空きノードに再度スケジューリングされ，実行されている．



(E) 予約終了時刻．予約キューは消去される．

図 6 キュー制御法による実装
Fig. 6 Queue control method behavior.

を再活性化する．この際に他のキューが予約対象ノードを使用することがないよう，他のキューのノード情報も操作する．投入されていたジョブが走り出す．

この際に，当該ノードで実行中のジョブがあった場合には，PluS は，そのジョブをいったんサスペンドし，qresub コマンドを用いて当該ジョブのコピーを再投入し，もとのジョブを削除する．これによって，ジョブは他のノードで，最初から再実行されることになる．

(4) 予約終了時刻が来たら予約に対応したキューを削除する．同時に，(2) で操作した他のキューに対して，当該ノードを再び使用するように再設定する．

5. 評　　価

本章では，PluS 事前予約機構の 2 つの実装方法について，実装の容易性と実行速度の両面で比較を行った．前者は，スケジューラ置換法ではスケジューリングモジュールの再実装のために記述量が大きくなることが予想されるため，後者は，キュー制御法でキュー操作のオーバヘッドが予想されるためである．

実装の容易性は，ソースコードの行数によって定量的に評価した．厳密には，使用言語が Java，C，sh と多岐にわたるため，単純な行数の加算は意味をなさないが，一定の傾向を読み取ることはできる．

PluS 事前予約機構実行速度に関しては，2 つの測定項目が考えられる．1 つは，予約コマンド入力に対するレスポンス，もう 1 つは，予約開始，終了時刻における処理の速度である．後者は，周期的に起動されるスクリプトが，その周期の間に終了すれば十分であるため，それほど本質的ではない．これに対し前者は，予約コマンドが上位スケジューラとの連携にも使用され，その際のレイテンシを決定するファクタとなり，ひいてはグローバルスケジューリングシステム全体のレスポンスを決定するファクタともなることから，本質的である．このため我々は，前者のコマンド入力のレスポンスに着目し評価を行った．

5.1 コード量による評価

5.1.1 各実装のコード量

PluS の 3 つの実装は，Java の予約管理・スケジューリングを行うコードを共有し，それに個々の実装法に依存したコードが付随する構造をとる．**図 7** に各実装法における，コードの量を示す．下部の 8,000 行あまりは，すべての実装法で共有されるソースである．

共有部分を除いた固有部分の行数は，TORQUE 版

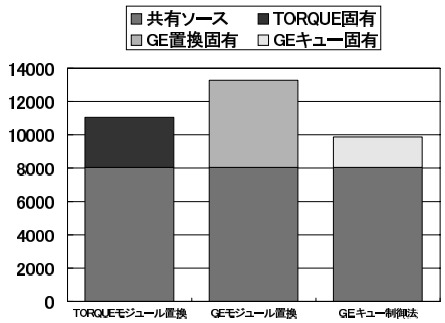


図 7 コード行数による評価
Fig. 7 Evaluation based on Num. of lines.

で 3,000 行程度，Grid Engine スケジューラ置換版で 5,200 行程度，Grid Engine キュー制御版で 1,800 行程度である．

5.1.2 コード量による評価に関する議論

2 つの実装法でコード量を比較を比較する際，PluS 事前予約機構のスケジューラ置換法の実装が現在のところ TORQUE 版，Grid Engine 版ともにサブセット実装となっていることに注意する必要がある．使用頻度が高い機能から順に実装を進めているため，典型的な利用環境では問題がないものの，PluS 事前予約機構を利用することでいくつかのスケジューリング機能が使用できなくなる可能性がある．これに対して，キュー制御法による実装では，既存のスケジューリングモジュールをそのまま利用するため，このような問題点はない．

コード量をベースとして評価するためには，本来スケジューラ置換法の完全な実装を行ったうえでキュー制御法と比較するべきであるが，実装時間の問題で不可能であった．参考資料として，未実装機能とそれらを実装した場合に必要となると思われる行数を概算したものを**表 2** および **表 3** に示す．これは，Grid Engine のコードを解析して算出した値である．

まず，実際に実装したコード量で比較してみよう．Grid Engine の実装どうしを比較すると，キュー制御版のコード行数は 3 分の 1 程度と，小さいことが分かる．TORQUE 版は SGE のスケジューラ置換法と比較すると行数が少ないが，Grid Engine キュー制御版よりも行数が多い．

この傾向は，表 2，3 に示した予測実装行数を加えるとさらに顕著になる．共通部分を除いた行数は Grid Engine のスケジューラ置換版で 25,000 行程度となることが予測されるが，キュー制御版では 1,800 行のままである．TORQUE 向けのスケジューラ置換版では，7,000 行程度となることが予測される．TORQUE 向けの行数が少ないのは TORQUE のスケジューリングモジュールの機構が比較的単純であるためである．

表 2 TORQUE 版未実装機能と行数
Table 2 Lines of code for unimplemented capability (TORQUE).

機能	行数概算
要求リソース指定（ホスト名指定など）	200
詳細スケジューリングアルゴリズム（load balancing）	300
ジョブ依存関係指定	2,000
フェアシェア	800
アルゴリズム時間帯別指定	200
休日・運用停止時間帯対応	500
計	4,000

表 3 Grid Engine 版未実装機能と行数
Table 3 Lines of code for unimplemented capability (Grid Engine).

機能	行数概算
フェアシェア	4,000
デッドラインスケジューリング	100
ジョブ依存関係指定	100
チケット管理	4,000
その他	15,000
計	23,200

以上から，キュー制御法による実装がスケジューラ置換法による実装と比較して，容易であることが確認できた．また，キュー制御法のメリットは，スケジューリングモジュールの機構が複雑なシステムで，より顕著となることも分かった．

5.2 予約実行時間

キュー制御法では外部からキューイングシステムのキューを制御するため，予約などの操作が遅くなることが予想される．このコストがどの程度になるかを評価するために，Grid Engine 向け PluS を用い，スケジューラ置換法とキュー制御法で，予約時間およびキャンセル時間を計測した．この予約およびキャンセル操作にかかる時間が数秒を超えると，上位階層であるスーパスケジューラのレイテンシに影響が出ることが予想される．

実行環境としては，1 台のヘッドノードと 4 台のワーカノードから構成されるミニクラスタを用いた．各ノードは Pentium III 1.4 GHz Dual CPU，メモリ 2 Gbyte，OS は RedHat 8 となっている．測定は 10 回行い，time コマンドを用いてコマンドの実行時間を計測した．測定は他に予約がまったく入っていない状態で行ったが，数十の予約が入った状態でもほぼ同じ結果を得ている．

実際の予約，キャンセルの動作を行った結果を**表 4** に示す．一般に，スケジューラ置換法が若干高速であ

表 4 スケジューラ置換法とキュー制御法のコマンド実行時間（秒）
Table 4 Execution time for reservation and cancellation.

	予約				キャンセル			
	平均	分散	最小	最大	平均	分散	最小	最大
スケジューラ置換法	1.02	0.04	0.91	1.54	0.92	0.00	0.85	1.03
キュー制御法	1.95	0.02	1.76	2.25	1.02	0.00	0.97	1.11

表 5 スケジューラ置換法とキュー制御法のサーバでの実行時間の内訳（秒）
Table 5 Breakdown of execution time.

	スケジューラ置換法		キュー制御法	
	予約	キャンセル	予約	キャンセル
内部処理	0.017	0.008	0.051	0.006
DB 操作	0.334	0.214	0.255	0.219
qconf 実行	-	-	0.871	0.095
計	0.351	0.222	1.177	0.320

ることが分かる．この差は前者では完全に予約管理モジュール（兼スケジューリングモジュール）内部に閉じた動作になるのに対して，後者ではキューイングシステムに対して，キュー制御コマンドを行うためである．

予約およびキャンセル操作の内訳は，1) Java コマンドの起動，2) RMI による接続，3) PluS 事前予約機構内部処理，4) データベース操作，5)（キュー制御法の場合のみ）qconf によるキューの制御，となる．このうち 3) と 5) のみがキュー制御法とスケジューラ置換法とで異なり，残りの 1)，2)，4) は共通の動作である．

1) と 2) のコストを知るために，1) と 2) のみを行うコマンドを作成し測定を行った．その結果，1) と 2) の動作に平均で 0.62 秒かかることが分かった．さらに，3)，4)，5) のコストを調べるために，PluS 事前予約機構内部のログから，それぞれの処理にかかる時間を算出した．この結果を**表 5** に示す．qconf コマンド実行のコストがスケジューラ置換法とキュー制御法の大きな相違となっていることが分かる．予約時のデータベース操作のコストが，両手法で 0.1 秒近く異なっているが，この値は計測ごとにかなり不安定であり，測定の誤差の範囲だと考えられる．qconf コマンドの実行が，予約時とキャンセル時で大きく異なるのは，予約操作時には，4 回のキュー制御が必要なのに対し，キャンセル操作時には 1 回のみであるためであると思われる．

また，予約・キャンセル操作の与える計算機への負荷も懸念される．上述のように，キュー制御法のオーバヘッドの大半は qconf コマンドであるため，qconf コマンドの動作の詳細を調査した．qconf コマンドは Grid Engine のキューを操作するコマンドであるが，動作の大半は sge_qmaster モジュール内部でのデータベースに対する操作となる．使用するデータベースは Grid Engine インストール時に選択することが可能だが，今回の実験ではデフォルトの Berkeley DB を用いている．qconf コマンドを連続して発行する実験を行い，ホスト計算機のロードアベレージを監視したが，変動はなかった．

以上の結果から，1) 予約，キャンセル操作時のコストはいずれの手法でも 1–2 秒程度であり，レスポンス時間として許容範囲であること，2) スケジューラ置換法とキュー制御法でのコストの相違の大半は qconf によるキュー操作のコストであり，その他の部分はほとんど変わらないこと，3) キュー操作のホスト計算機にあたえる CPU 負荷は軽微であることが分かった．

6. 関 連 研 究

商用のバッチキューイングシステムである PBS Professional，LSF は事前予約機能をもともとスケジューリングモジュールの機能の一部として持っている．既存のバッチキューイングシステムにプラグインするタイプのスケジューラとしては Maui と Catalina がある．これらのいずれも 2 相コミットを実装していない．2 相コミットの実装は 4.3 節に述べたとおり，それほど困難ではない．これらのシステムが 2 相コミットを実装していないのは，主として単一のシステム内での事前予約を想定しており，人間の手を介さない完全に自動的なコアロケーションを考慮していないためであると思われる．

6.1 Maui

Maui スケジューラ[10] は，TORQUE をメンテナンスしている Cluster Resources 社が提供しているスケジューリングモジュールで，TORQUE と協調動作し事前予約機能を提供する☆．

Maui スケジューラは，スケジューラ置換法で実装されており，TORQUE のスケジューラモジュールを完全に置換する．

6.2 Catalina

Catalina[11] は TORQUE と協調動作するスケ

☆ かつては Grid Engine をサポートしていたが，現在はサポートしていない．

ジューラで，米国 Tera Grid プロジェクトで使用されている．Catalina は，User-Settable Reservation という名前で事前予約を許している．Catalina はスケジューラ置換法で実装されている．ほとんどの部分は Python で実装されており，速度クリティカルな部分のみ C のモジュールが使用されている．

Catalina の事前予約ポリシは通常実行キュー優先となっている．Catalina では，事前予約ジョブに限らず，すべてのジョブがサブミット時にスケジュールされ，予定実行時間を与えられる．そして，通常ジョブをすべてスケジューリングしたうえで，空いているタイムスロットがあったときのみ事前予約を受け入れる．これは，PluS の現在の事前予約ポリシが，事前予約優先になっていることと好対照をなしている．

7. 議　　論

7.1 スケジューラ置換法とキュー制御法の特性

5 章で見たとおり，既存のスケジューリングモジュールとの互換性を保ちながら，事前予約を実現するためには，実装の容易性の観点からはキュー制御法が優れている．しかし，キュー制御法にはいくつかの制限事項がある．まず，キュー制御法を適用するためには，キューイングシステム側の機能として，キューと計算ノード集合，キューとユーザ集合の紐付けが可能である必要がある．さらに，キューの生成と紐付けの変更をキューイングシステムを停止することなく実現できなければならない．4.1 節で述べたとおり．Grid Engine はこの条件を満たしているが，TORQUE は満たさないため，キュー制御法での実装を行うことはできなかった．TORQUE は，キューと計算ノード集合の対応付けをすることができないだけでなく，キューを制御した際に，スケジューリングモジュールの再起動が必要であったためである．もう 1 つの制限事項としては，スケジューリングアルゴリズム設計の自由度がある．スケジューラ置換法と異なり，既存スケジューラをそのまま利用するキュー制御法では，既存スケジューラの提供する以上の機能を提供することはできない．

これに対してスケジューラ置換法は，既存スケジューラとの互換性を意識する必要がない場合に特に有効であるといえる．たとえば前述の Catalina は，TORQUE をジョブ管理およびノード管理のいわば足回りとして利用しているだけで，TORQUE の提供するスケジューリングポリシとの互換性は無視されている．これは Catalina が SDSC（San Diego Supercomputer Center）組織内部の要請によって実装されたため，SDSC で必要としない機能を考慮する必要がなかったことによる．Catalina が行ったような，すべてのジョブを事前にスケジュールする，という本質的なスケジューリング手法の変更は，キュー制御法では不可能である．

上述のように，スケジューラ置換法とキュー制御法にはそれぞれメリットとデメリットがあり，対象とするキューイングシステム，および必要とされる機構に応じてどちらを採用するべきかを，そのつど判断する必要があると思われる．

7.2 予約開始時の既存ジョブの強制終了に関する議論

4.2 節で述べたとおり，予約開始時刻に予約対象ノードで他のジョブが走っていた場合，PluS 事前予約機構はそのジョブを強制終了し，再度キューイングする．このジョブが実行中に何らかの副作用を外界に及ぼす性質のものであった場合，強制終了によって再実行時の挙動が期待しないものとなる可能性がある．たとえば，データベースやファイルを上書き変更するジョブであれば，ジョブが中途で強制終了されることにより，データベースやファイルの状態が不整合になることがありうる．ジョブが再実行されても，状態不整合によって，ジョブの挙動が変わってしまう．

しかしこの問題は，PluS 事前予約機構による強制終了によってのみ生じる問題ではない．たとえばハードウェアの問題や，システムメンテナンスのためにジョブが強制終了される可能性はつねに存在する．したがって，プログラマはいずれにせよ，強制終了されても不整合を起こさないようにプログラムを記述しなければならない．たとえば，データベースであればトランザクションを利用すれば，このような不整合は生じない．ファイルであってもファイル構造を工夫し追記型で更新するようにすることでこのような事態を未然に防ぐことができる．

8. お わ り に

グリッド上で資源同時確保を実現するために，2 相プロトコルをサポートする PluS 事前予約機構を実装した．PluS は，既存のローカルキューイングシステムである TORQUE もしくは Grid Engine と協調動作し，事前予約機能を提供する．既存ローカルキューイングシステムに事前予約機能を追加する手法としては，スケジューリングモジュール置換法とキュー制御法が考えられるが，この両者に関して予約機能を設計，実装し評価した．

その結果，前者はオーバヘッドが少なくポリシ実現の自由度が高いこと，後者は既存スケジューリングモジュール機能の再実装が必要ないため実装コストが小

さいことが分かった．後者には，オーバヘッドが大きく，スケジューリングポリシ設定の自由度が小さいという欠点があるが，実験の結果オーバヘッドはキャンセル時でも 1 秒程度と許容範囲内であることが確認できた．

今後の課題としては以下があげられる．

- PluS 資源予約機構の改良
  現在，PluS 資源予約機構は，計算資源が均質であることを仮定しており，指定できる項目はノード数のみである．今後，非均質なクラスタ環境でも適切な予約操作が行えるよう，アーキテクチャ，メモリ量，ディスク容量などに対応する予定である．
- 事前予約とローカルジョブの関連
  4.2 節で述べたとおり，現在の PluS では事前予約を最優先し，ローカルジョブをバックフィルとして扱う．また，事前予約は早い者勝ちで，ユーザ間のプライオリティ制御なども行われない．このポリシは，現在我々が対象としている実験グリッド環境では十分であるが，プロダクション運用を行う場合には問題が生じる可能性がある．
  すべての環境ですべてのユーザを満足させるポリシは存在しないと思われるので，いくつかのオプションを準備し，管理者が最低限のポリシ変更を行えるようにする必要がある．また，ClassAd [12] などのポリシ記述言語を利用し管理者がある程度カスタマイズ可能な構造とすることも検討する必要がある．
- 他のキューイングシステムへの適用
  キュー制御法による実装は，特定のユーザに対して特定のノードを割り当てることができるキューイングシステムであれば，適用可能である．他のキューイングシステムへの適用を検討する．

謝辞 本研究の一部は，文部科学省科学技術振興調整費「グリッド技術による光パス網提供方式の開発」による．

参 考 文 献

1) Foster, I., Kesselman, C., Lee, C., Lindell, B., Nahrstedt, K. and Roy, A.: A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation, *Proc. Intl Workshop on Quality of Service* (1999).
2) Takefusa, A., Hayashi, M., Nagatsu, N., Nakada, H., Kudoh, T., Miyamoto, T., Otani, T., Tanaka, H., Suzuki, M., Sameshima, Y., Imajuku, W., Jinno, M., Takigawa, Y., Okamoto, S., Tanaka, Y. and Sekiguchi, S.: G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS, *Future Generation Computing Systems*, Vol.22, pp.868–875 (2006).
3) 中田秀基，竹房あつ子，岸本 誠，大久保克彦，工藤知宏，田中良夫，関口智嗣：グローバルスケジューリングのためのローカル計算資源管理機構，情報処理学会 HPC 研究会 2006-HPC-107 (2006).
4) Nakada, H., Takefusa, A., Ookubo, K., Kishimoto, M., Kudoh, T., Tanaka, Y. and Sekiguchi, S.: Design and Implementation of a Local Scheduling System with Advance Reservation for Co-allocation on the Grid, *Proc. CIT2006* (2006).
5) 竹房あつ子, 中田秀基, 武宮 博, 松田元彦, 工藤知宏, 田中良夫, 関口智嗣：異種の複数スケジューラで管理される資源を事前同時予約するグリッド高性能計算の実行環境，HPCS 2007 予稿集，pp.135–142 (2007).
6) Tannenbaum, A.S.: *Distributed Operating Systems*, Prentice Hall (1994).
7) TORQUE Resource Manager. http://www.clusterresources.com/pages/products/torque-resource-manager.php
8) Grid Engine. http://gridengine.sunsource.net
9) db4objects. http://www.db4o.com/
10) Maui Cluster Scheduler. http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php
11) Yoshimoto, K., Kovatch, P. and Andrews, P.: Co-scheduling with User-Settable Reservations, *Job Scheduling Strategies for Parallel Processing*, Feitelson, D.G., Frachtenberg, E., Rudolph, L. and Schwiegelshohn, U. (Eds.), pp.146–156, Springer Verlag (2005).
12) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *Proc. HPDC-7* (1998).

(平成 18 年 10 月 10 日受付)
(平成 19 年 2 月 11 日採録)

中田　秀基（正会員）

昭和 42 年生．平成 2 年東京大学工学部精密機械工学科卒業．平成 7 年同大学大学院工学系研究科情報工学専攻博士課程修了．博士（工学）．同年電子技術総合研究所研究官．平成 13 年独立行政法人産業技術総合研究所に改組．現在，同所グリッド研究センター主任研究官．平成 13 年より平成 17 年度まで東京工業大学客員助教授を兼務．グローバルコンピューティング，並列実行環境に関する研究に従事．

竹房あつ子（正会員）

昭和 48 年生．平成 8 年お茶の水女子大学理学部情報科学科卒業．平成 10 年同大学大学院理学研究科情報科学専攻修士課程修了．平成 12 年同大学院人間文化研究科複合領域科学専攻博士課程修了．博士（理学）．同年日本学術振興会特別研究員，平成 14 年お茶の水女子大学理学部助手．平成 17 年独立行政法人産業技術総合研究所グリッド研究センター．並列分散処理，グリッドコンピューティング，スケジューリングに興味を持つ．ACM，電子情報通信学会各会員．

大久保克彦（正会員）

平成 12 年 3 月電気通信大学大学院電気通信学研究科修士（工学）．平成 12 年 4 月（株）数理技研入社．分散ファイルシステム，グリッドシステム開発等に関わる．

工藤　知宏（正会員）

平成 3 年慶應義塾大学大学院理工学研究科博士課程単位取得退学．東京工科大学助手，講師，助教授を経て，平成 9 年より新情報処理開発機構並列分散システムアーキテクチャつくば研究室長，平成 14 年より産業技術総合研究所グリッド研究センタークラスタ技術チーム長．博士（工学）．並列処理，通信アーキテクチャに関する研究に従事．IEEE CS 会員．

田中　良夫（正会員）

昭和 40 年生．平成 7 年慶應義塾大学大学院理工学研究科後期博士課程単位取得退学．平成 8 年技術研究組合新情報処理開発機構入所．平成 12 年通産省電子技術総合研究所入所．平成 13 年 4 月より独立行政法人産業技術総合研究所．現在同所グリッド研究センター主幹研究員．博士（工学）．グリッドにおけるプログラミングミドルウェアおよびグリッドセキュリティに関する研究に従事．IC 1999，HPCS 2005，SACSIS 2006 論文賞．ACM 会員．

関口　智嗣（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 59 年筑波大学大学院理工学研究科修了．同年電子技術総合研究所入所．情報アーキテクチャ部主任研究官．以来，データ駆動型スーパーコンピュータ SIGMA-1 の開発等の研究に従事．平成 13 年独立行政法人産業技術総合研究所に改組．平成 14 年 1 月より同所グリッド研究センターセンター長．並列数値アルゴリズム，計算機性能評価技術，グリッドコンピューティングに興味を持つ．市村賞受賞．日本応用数理学会，ソフトウェア科学会，SIAM，IEEE 各会員．