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ひと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。われわれは，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．われわれは，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わかった．

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 Reservatikon 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

グリッドの目的の一つとして，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，もっと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一つとして，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ーフェイス，の 3 つが必要となる．

われわれは，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ケジューラ [2)] を，3) に関しては WSRF(Web Services Resource Framework)

† 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.

をベースとした予約インターフェイス [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 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 [5] 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[6] および Grid Engine[7] 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．われわれは，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わかった．

本稿の以下の構成を以下に示す．2 節で事前予約と 2 相コミットプロトコルの必要性に関して述べる．3 節に PluS 事前予約機構の設計を示す．4 節に実装の詳細を示す．5 節に 2 つの手法の評価を示す．7 節はまとめである．

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，予約を一旦もとの時間帯に戻した上で，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，変更時に一旦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 [5]，いくつものプロトコルが提案されている．そのもっとも基本的なプロトコルが 2 相コミットである．

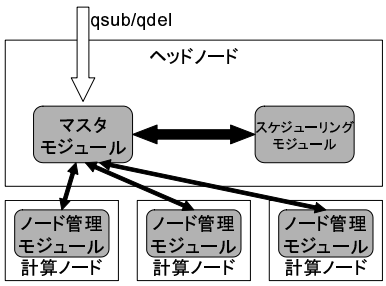


図 1 バッチキューイングシステムの一般的な構成

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，ひと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マンドを受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，ひと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上，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，われわれは，1) と 2) に着目し，それぞれ設計と実装を行った．

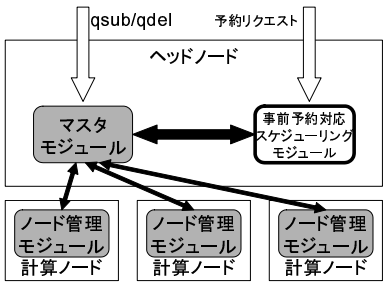


図 2 スケジューリングモジュール置換法

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もっとも望ましい．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．ひとつは，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伴い，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．

もうひとつの問題点は，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．これは一般に容易な事ではない．

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

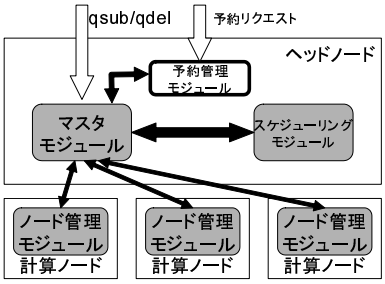


図 3 キュー制御法

用意しているコマンドライン インターフェイスを利用する．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り当てを制御する．

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．さらに，公開されているコマンドラインインターフェイスを使用して，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ーフェイ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つ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ーフェイスを利用する．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事が予想される．

もうひとつのデメリットは，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

われわれ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 のキュー機構が計算ノードに対して束縛する機能を提供しないためである．

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 インターフェイスから行う．コマンドライン インターフェイスの一覧を表 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．われわれは，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[8] を用いた．db4objects は，非常に簡便なインターフェイスを提供しており，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

表 1　予約コマンドラインインターフェイス

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	

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　予約に対するノードの割り当て

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り当ての例である．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，常に $N-1$ 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．

4.4　TORQUE 向け PluS

TORQUE は OpenPBS の亜種のひとつである．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．

4.4.1　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．われわれ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．

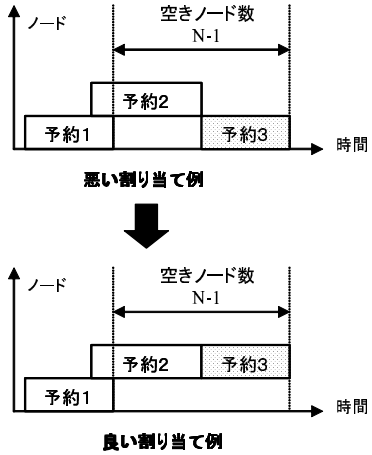


図 4　予約ノードの配置

4.4.2　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ーフェイスもこれと同じ機構を用いてユーザ認証を行う．

4.5　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．

4.5.1　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

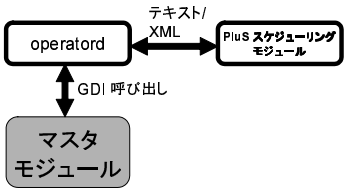


図 5 operatord による通信の中継

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．われわれは 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)．

4.5.2 キュー制御法による実装

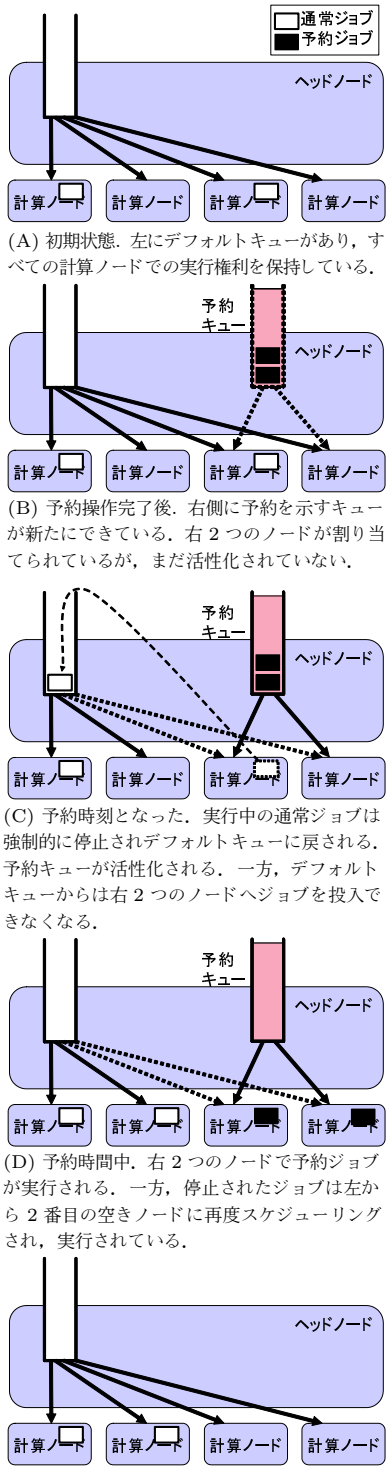
Grid Engine におけるキュー制御法による PluS 事前予約機構の動作を示す．

(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，そのジョブを一旦サスペンドし，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．



(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

5. 評　　価

5.1 コード量による評価

実装の容易性を評価するために，ソースコードの行数による定量的評価を行った．厳密には，使用言語が

表 2　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)

	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

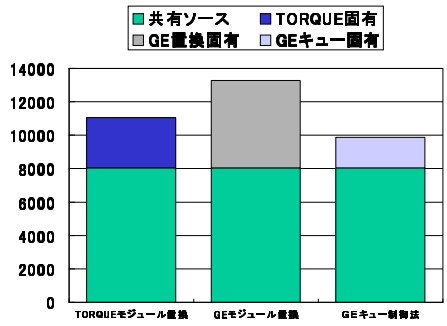


図 7　コード行数による評価

Java, C, sh と多岐にわたるため，単純な行数の加算は意味を成さないが，一定の傾向を読み取ることはできる．

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 8000 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 版で 3000 行程度，Grid Engine スケジューラ置換版で 5200 行程度，Grid Engine キュー制御版で 1800 行程度である．Grid Engine の実装同士を比較すると，キュー制御版のコード行数は 3 分の 1 程度と，はるかに 小さいことがわかる．TORQUE 版はスケジューラ置換法であるにもかかわらず，比較的行数が少ない．これは TORQUE のスケジューリング機能が比較的単純で再実装が容易なためである．それでも，Grid Engine キュー制御版よりも行数が多い．

さらに，今回のスケジューラ置換法でのスケジューリングモジュールの実装は，TORQUE 版，Grid Engine 版ともサブセット実装となっており，コード量ベースで，TORQUE 版で 5 割，Grid Engine 版で 3 割程度の実装にとどまっている．既存スケジューリングモジュールの機能を完全に再現するためには，それぞれ，4000 行，20000 行程度の追加実装が必要になると思われる．これに対し，キュー制御版では既存スケジューリングモジュールの機能を活用できるため，これ以上の実装は必要ない．このことから，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．

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．

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 2Gbyte, 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，数十の予約がはいった状態でもほぼ同じ結果を得ている．

結果を表 2 に示す．一般に，スケジューラ置換法が若干高速であることがわかる．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．

また，キャンセル操作においてはそれほど大きな差がないが，予約操作においてはおよそ 1 秒余分に時間がかかっている．これは予約操作時には，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 回のみであることによる．

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 相コミットを許していない．これは，これらのシステムの機構が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 [9] 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．

☆ 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

6.2 Catalina

Catalina[10] は TORQUE と協調動作するスケジューラで，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．通常実行のキューに入っている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. 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わかった．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挙げられる．

- 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 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，現在われわれが対象として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．
- 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., Carl Kesselman, C. L., 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., et al. : G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS, *Future Generation Computing Systems*, Vol. 22(2006), 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, *Proceedings of CIT2006* (2006).
5) Tannenbaum, A. S.: *Distributed Operating Systems*, Prentice Hall (1994).
6) TORQUE Resource Manager. http://www.clusterresources.com/pages/products/torque-resource-manager.php.
7) Grid Engine. http://gridengine.sunsource.net.
8) db4objects. http://www.db4o.com/.
9) Maui Cluster Scheduler. http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php.
10) 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.)), Springer Verlag, pp. 146–156 (2005). Lect. Notes Comput. Sci. vol. 3834.