計算資源とネットワーク資源を同時確保する予約ベースグリッドスケジューリングシステム

竹房 あつ子[†] 中田秀基[†] 工藤知宏[†]
田中良夫[†] 関口智嗣[†]

グリッドで異なる組織に管理される分散した計算資源を用いた高性能大規模計算を実行するには，分散した計算資源とその間の高品質なネットワークを同時に確保することが重要である．本研究では，事前予約により計算資源と高品質ネットワーク資源を同時に確保し，その上でユーザジョブの自動実行を可能にするグリッドスケジューリングシステムを提案・開発した．提案システムはグリッド資源スケジューラ (GRS) と計算資源マネージャ(CRM) からなり，GRS が複数 CRM とネットワークキャリアの提供するネットワーク資源管理システムと連携して計算・ネットワーク資源のコアロケーションを実現する．

A Grid Scheduling System for Co-allocation of Computing and Network Resources with Advance Reservation

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

Co-allocation of both network and computing resources from different domains is an essential issue for high-performance distributed computing over the Grid. We propose and develop a Grid scheduling system for co-allocation of both computing and network resources with advance reservation. The proposed system also makes it possible to execute user jobs over the reserved resources. Our system consists of Grid Resource Scheduler (GRS) and Computing Resource Managers (CRM). GRS selects required resources by interworking with CRMs and Network Resource Management Systems provided by network carriers.

1. はじめに

グリッド基盤技術の成熟により，異なる組織により管理される地理的に分散した計算資源を用いた大規模科学技術計算 (以降メタコンピューティングと呼ぶ) が実行可能になった．

メタコンピューティングにおいては分散した資源を同時に確保する事 (コアロケーション) が必要であるが，現状ではコアロケーションを自動的に行う技術は実用段階に至っておらず，コアロケーションを必要とする場合には電子メール等により各サイトの管理者と調停して全ての資源を事前に予約する必要がある．また，実際の予約はサイトの管理者がバッチシステムの設定を変更することにより実現される事が多いが，設定ミスなどにより予約の時刻が来ても投入したジョブが実行されないといった問題も生じている．

また，分散した資源を接続するネットワークは高速化しているが，ベストエフォート型の通信を行うインターネットではアプリケーションに対して性能を保証することができず，ユーザはアプリケーションの実効性能を見積もる事ができない．光ネットワーク技術を用いることで高品質通信が提供可能であるが，利用するネットワークの管理組織のオペレータと E-mail や FAX 等によりに交渉し，人手により事前に必要なネットワークを設定しなければならない．

このように，分散した異なる組織の計算資源の確保と，確保した環境でアプリケーションを実行するための手続きが自動化されておらず，ユーザがメタコンピューティングを実行する際の負担が非常に大きい．グリッドをエンドユーザによるメタコンピューティングのインフラとして実用化するためには，ユーザに対して単一システムの占有利用と同様な利便性および安定した品質を提供する必要があり，コアロケーション技術の開発と通信性能の保証はグリッドの実用化に向けて重要な技術的課題となる．

本研究では，計算資源とネットワーク資源を同時確保する事前予約をベースのグリッドスケジューリング

† 産業技術総合研究所 National Institute of Advanced Industrial Science and Technology (AIST)

システムを設計・開発する．計算資源と通信性能が保証されたネットワーク資源を自動的にコアロケーションし，提供するものはまだ存在せず，本研究の成果により，メタコンピューティングを実行するための安定した環境をユーザの要求に応じて自動的に構成し，提供する事が可能となる．

提案するスケジューリングシステムは，グリッド資源スケジューラと計算資源マネージャからなる．グリッド資源スケジューラはユーザの要求に応じて分散した計算資源マネージャと連携し，事前予約により計算資源をコアロケーションする．また，通信性能を保証するため，G-lambda プロジェクト[1),2)] で提案する GNS-WSI (Grid Network Service - Web Services Interface) を介してネットワーク管理システム (NRM)[3)] と連携し，必要なネットワーク資源も同時に事前予約し，提供する．計算資源マネージャでは，オープンソースのローカルスケジューラ OpenPBS[4)] の亜種のひとつである TORQUE[5)] に対して事前予約可能なスケジューリング機構を実現したローカルスケジューラを開発し，それを用いた．なお，NRM による高品質ネットワーク資源の事前予約の詳細は，文献[2)] で扱う．

グリッド資源スケジューラと計算資源マネージャの予約インタフェースは WSRF(Web Services Resource Framework)[6)] に基づく外部インタフェースを設計し，Globus Toolkit 4 (GT4)[7),8)] を用いて実装した．これにより，GT4 のセキュリティ機構を用いて認証・認可を行い，Globus のジョブ起動機構 GRAM と連動して事前予約によるグリッド環境でのジョブの自動実行を可能にする．本稿では，提案するグリッドスケジューリングシステムの予備評価を行い，事前予約による計算資源とネットワーク資源のコアロケーションの実現可能性を示す．

2. 事前予約ベースのグリッドスケジューリングシステムの概要

計算資源とネットワーク資源の同時確保を可能にするため，G-lambda プロジェクトでは**図 1** に示すようなグリッドコアロケーションシステムを提案している[1),2)]．コアロケーションシステムはグリッド資源スケジューラ (GRS)，ネットワーク資源管理システム (NRM)，計算資源マネージャ(CRM)，分散する実計算資源と計算資源間を結ぶネットワークで構成される．GRS が各サイトで計算資源を管理する OpenPBS[4)] や GridEngine[9)] 等の CRM，ネットワーク資源を管理する NRM と連携し，ユーザの要求を満たす計算資源と資源間のネットワーク帯域を選定し，事前予約する．計算資源とネットワーク資源を統一的に扱うため，NRM と CRM はウェブサービスに基づく外部インタフェースを提供し，GRS と NRM 間のインタフェース GNS-WSI(Grid Network Service - Web Service Interface) を G-lambda プロジェクトで規定している．

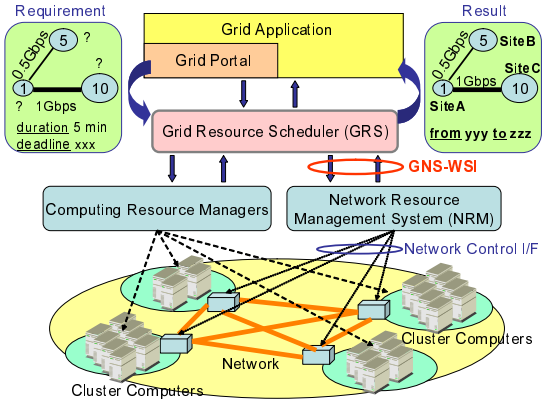

図 1 コアロケーションシステムの概念図．

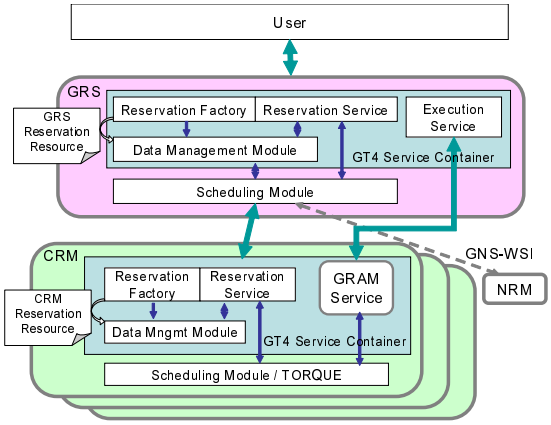

図 2 WSRF に基づくスケジューリングシステムアーキテクチャ．

本研究では，図 1 に示す構成を前提として，GRS と CRM からなる事前予約機能を持つグリッドスケジューリングシステムを開発する．GRS と CRM はいずれも GT4 を用い，ウェブサービスにおけるステートフルリソースのモデリングとアクセスのための仕様 WSRF に基づく外部インタフェースを設計・実装した．

2.1 WSRF によるシステムアーキテクチャ

図 2 に WSRF に基づくスケジューリングシステムアーキテクチャを示す．ユーザと GRS 間，GRS と CRM 間の太字矢印は WSRF に基づく手続きでサービスに関する情報を授受する．

WSRF では一般に，サービスは対になる Factory サービスによって作成される．GRS と CRM の GT4 サービスコンテナでは，それぞれ ReservationFactory と ReservationService を用意し，計算資源やネットワーク資源の予約のための手続きを行う．各データ管理モジュールでは，各予約処理の状況 (ReservationResource) を管理し，格納されたデータの持続性を保証する．各スケジューリングモジュールでは，事前予約に基づきグローバル／ローカルな資源のスケジューリ

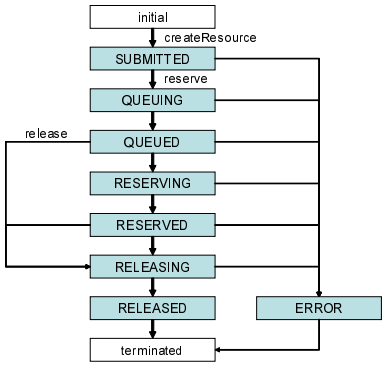

図 3 ReservationResource の状態遷移.

ングを行う.

予約された資源上でユーザのジョブを自動的に実行するため，ユーザからの実行要求を GRS の ExecutionService で受け付ける．ExecutionService はデータ管理モジュールに格納された予約情報から，事前予約された計算資源を管理する CRM の GRAM Service に対して受け付けたジョブを各予約 ID とともにサブミットする．GRAM Service は GT4 で提供される Globus のジョブ起動機構であり，GRAM Service から本研究で開発したローカルスケジューラに対してジョブを投入することで予約時刻に各ジョブを自動的に実行開始することができる.

2.2 GT4 による認証・認可

予約インターフェイスでは，接続してきたクライアントのユーザを認証し，ローカルサイトのユーザにマップしなければならない．われわれはこれを，GT4 の提供する標準的な機能を用いて実現した．ユーザの認証には PKI に基づく証明書を用いた．証明書に書かれたグローバルユーザ名をローカルサイトのユーザ名にマップするためには，grid-mapfile と呼ばれるファイルを用いる.

CRM の予約サービスは認証されマップされたローカルサイトのユーザ名を指定して sudo コマンドを発行することで，予約関連コマンドをローカルユーザの権限で実行する．これらの方法は Globus Toolkit 4 のジョブ実行機構である GRAM インタフェースでも用いられている方法を踏襲している.

3. 提案システムの詳細設計

3.1 ReservationResource の状態遷移

提案するグリッドスケジューリングシステムでは，個々の資源要求に関する情報を ReservationResource として GRS および CRM で管理する．各 ReservationResource には status というリソースプロパティがあり，status で ReservationResource の遷移状態を管理する．status は ReservationResource の生成，事前予約，修正，解放の要求と，予約された資源の状況の変化に応じて**図 3** のように変化する．**表 1** に status の一覧を示す．表中の「資源」は，GRS では計算・ネットワーク資源，CRM では計算資源を示す.

表 1 status リソースプロパティの一覧.

status	ReservationResource の状態
SUBMITTED	ReservationResource 生成済
QUEUING	資源のスケジューリング中
QUEUED	資源の割当完了
RESERVING	資源のプロビジョニング中
RESERVED	資源利用可能
RELEASING	資源解放中
RELEASED	資源解放済
ERROR	エラー発生

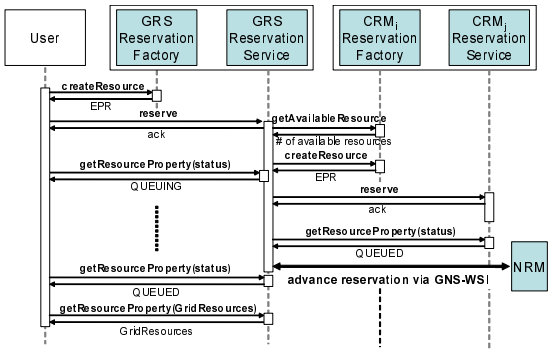

図 4 事前予約プロトコルシーケンス.

3.2 予約プロトコル

以下にポーリングをベースとした提案システムの事前予約，資源解放，資源監視のプロトコルシーケンスを示す．ウェブサービスではサーバからクライアントに対してコールバックするための仕様 WS-Notification[10] が提案されているが，プロトコルが複雑になり，耐故障性のためにポーリングを併用しなければならない．よって，初期インタフェースでは GRS と CRM に対してポーリングベースのオペレーションを設計した.

3.2.1 事前予約プロトコルシーケンス

図 4 に事前予約のプロトコルシーケンスを示す．事前予約では，ユーザの計算・ネットワーク資源と予約時間に関する要求に対して GRS が分散した複数の CRM および通信キャリアの提供するネットワーク資源管理システム (NRM) と連携し，適切な計算・ネットワーク資源を事前予約する.

ユーザが資源要求とともに GRS の ReservationFactory で提供する createResource オペレーションを呼び出すと，GRS ReservationResource が生成され，ユーザに対してそのリソースにアクセスするための参照であるエンドポイントリファレンス (EPR) が返される．次に，ユーザが EPR により GRS ReservationService に対して reserve オペレーションを呼び出すことにより，GRS が資源のスケジューリングを開始する.

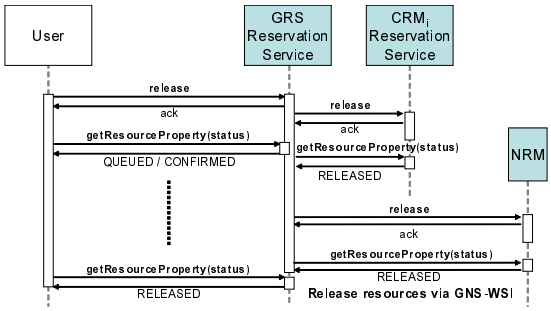

図 5 資源解放プロトコルシーケンス.

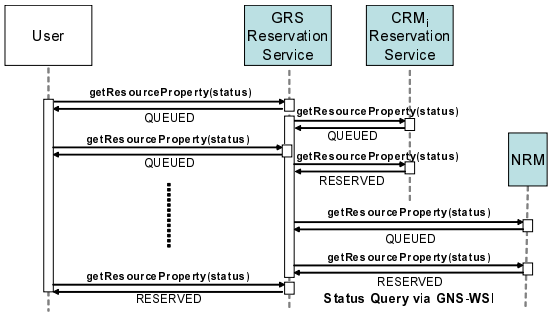

図 6 資源監視プロトコルシーケンス.

GRS は CRM_i の ReservationFactory に対して getAvailableResources オペレーションを呼び出すことで，要求された時間帯の利用可能 CPU 数等の情報を得る．CRM_i に十分な CPU 数がある場合は createResource で CRM ReservationResource を生成して EPR を受け取り，CRM_i の ReservationService に対して reserve で指定した時間帯の指定した CPU 数を確保する．同様に，ネットワーク管理システム (NRM) に対して G-lambda プロジェクトで定めている GNS-WSI の手続き[2] に従って必要なネットワークの帯域を事前予約する．プロトコルでは，CRM，NRM の予約順序は問わない．

全ての資源の事前予約が完了すると GRS ReservationResource の status が QUEUED，失敗すると ERROR になる．ユーザは gatResourceProperty オペレーションで status の値を取得し，要求した事前予約の成功／失敗を知る．成功した場合は，予約された資源の情報を得ることができる．

事前予約に必要な各 CRM の情報は，Ganglia[11] 等の従来のモニタリングシステムが提供するような現時点での負荷状況のみでは不十分である．また，個々の予約情報を全て公開したくない場合もあるため，一元管理することは難しい．例えば，グリッドが実用化されて様々なユーザが利用する場合，各ユーザの利用状況を公開すべきでなく，特に課金が発生する場合は資源提供者が詳細な資源利用状況を公開しないことも考えられる．よって，各 CRM や NRM で限定した問い合わせに対して利用可能状況を通知する仕組みにする．

3.2.2 資源解放プロトコルシーケンス

図 5 の資源解放手続きでは，割当て済または利用可能状態の CPU を解放する．ユーザは事前予約の際に送られた EPR を用いて GRS ReservationService の release オペレーションを呼び出す．GRS は GRS ReservationResource 情報から確保されている資源を管理する CRM の CRM ReservationResource への EPR を得て，それに対して release を呼び出し，解放する．GRS は NRM に対しても同様に資源解放手続きを行い，全ての資源の解放が成功すると GRS ReservationResource の status を RELEASED にし，失敗すると ERROR にする．ユーザは事前予約の際と同様に，status の値を取得することで解放要求の成功／失敗を知る．

3.2.3 資源監視プロトコルシーケンス

図 6 の資源監視手続きでは，予約時刻に事前予約された計算資源・ネットワーク資源が利用可能状態になっているかどうかを調べることができる．ある資源要求の予約時刻が近づくと，GRS は自動的に予約された資源を管理する CRM と NRM の status 値により資源の状況を取得する．資源が利用可能であれば，CRM および NRM の status は RESERVED となる．事前予約された資源の状態が全て RESERVED になると，GRS ReservationResource の status も RESERVED となる．

通常のグリッドアプリケーションでは，資源になんらかのエラーが発生した場合にそれを検知することはできない．特にネットワーク資源における予約のエラーでは，グリッドミドルウェアでも検知することができないことがある☆．グリッドスケジューリングシステムで予約された資源の状況を監視する機能が必要不可欠であり，これによりフォルトリカバリ等に役立てることができる．

4. グリッド資源スケジューラ (GRS) の実装

GRS は各計算資源を管理する CRM とネットワーク資源を管理する NRM と連携し，ユーザの資源に関する要求を満たす資源を事前予約に基づき確保する．提供するネットワーク資源の通信性能を保証するため，ネットワークキャリアが提供する NRM と連携し，ネットワーク資源として帯域が保証された光ネットワークを提供する．NRM との連携では，G-lambda プロジェクトで規定している GNS-WSI インタフェースを用いている．また，GRS では CRM の提供するジョブ操作インタフェースを介して事前予約された資

☆ 本グリッドスケジューリングシステムの実証実験では，ネットワーク資源に帯域保証可能な光パスネットワークを用いたが，設定エラー等で予約時刻にネットワークが利用可能にならなかった場合も，Globus Toolkit 2 ではそのエラーを検出できず，プロセスが通信待ち状態のまま走り続けてしまった．

源上でのユーザジョブの自動実行を支援する．2.1 節で述べたように，GRS ではこれらのサービスに対する WSRF ベースの外部インタフェースを GT4 を用いて実装した．

4.1 GRS の概要

図 2 で示したように，GRS は WSRF のサービスモジュールとスケジューリングモジュールにより構成される．また，WSRF のサービスモジュールはユーザに対して予約操作インタフェース (ReservationFactory，ReservationService) とジョブ操作インタフェース (ExecutionService) を提供する．

予約操作インタフェースでは，個々の資源予約要求を受け付け，予約に関する情報をデータ管理モジュールへ格納する．データ管理モジュールでは格納データの永続性を保証する．スケジューリングモジュールは，格納された資源予約要求に対して NRM，CRM と連携して適切な資源を割当てる．割当て処理が終了すると，スケジューリングモジュールはデータ管理モジュールにその結果を格納する．ユーザは予約操作インタフェースを介して予約結果を知ることができる．

ジョブ操作インタフェースは，事前予約された資源上でユーザのジョブを自動実行するための窓口となる．ユーザからジョブ実行要求を受け取ると，データ管理モジュールに格納されている予約情報をもとに関連する CRM に対してジョブを送信する．我々の開発した CRM は事前予約機能を持つため，送られたジョブは予約された時刻に割当てられた計算ノードで自動的に実行開始する．

4.2 WSRF の GRS サービスインタフェース

予約操作インタフェースである GRSReservationFactoryService と GRSReservationService，およびジョブ操作インタフェースの GRSExecutionService を実装した．**表 2** にサービスオペレーションの一覧を示す．

ReservationFactoryService は GRSReservationResource を生成するオペレーションのみを提供する．このオペレーションは，予約に必要な，予約期間，デッドライン，資源に関する要求を受け取り，予約情報を管理する GRSReservationResource を生成する．予約開始時間は希望があれば指定することができる．また，資源に関する要求には，クラスタ数，各クラス他の CPU 数，クラスタ間のネットワークバンド幅等を指定することができる．createResource は生成したリソースに対する参照である EPR(End Point Reference) を返す．このとき，実際の予約はまだ行われない．

GRSReservationService には資源の予約，情報取得，割当て結果取得，解放のための 4 つのオペレーションがある．reserve および release は操作のトリガに過ぎず，結果はサービスのリソースプロパティである status に反映される．

GRSExecutionService ではユーザのジョブのサブミット，キャンセルに関する 2 つのオペレーションを提供する．これらの操作の結果もリソースプロパティ status に反映される．これにより，メタコンピューティングジョブの自動実行が実現できる．

表 3 GRSReservationResource のプロパティ

リソースプロパティ名	意味
status	予約状況
GridResources	事前予約された資源の情報

表 3 に GRSReservationResource のリソースプロパティを示す．status では現在の予約状況を格納している．GridResource では事前予約された資源の情報がすべて格納されている．この情報には，コアロケーションした各クラスタ数とその CPU 数，クラスタ間のネットワークに関する情報，予約開始時刻，予約終了時刻，予約時間が含まれている．

4.3 GRS スケジューリングアルゴリズム

コアロケーションを目的とした GRS でのスケジューリングは，一種の制約解消問題となる．図 1 中のユーザのリクエストで示すような要求に対して，クラスタとクラスタ間のネットワークを指定した時間内に割当てる．この際，ユーザの要求を満たす資源の組み合わせは複数通り考えられるが，どのような割当てが最適であるかの基準は明らかでない．

ユーザ側の視点では，デッドラインまでのできるだけ早い時刻に割当てる，多くの資源をすでに予約しているユーザは優先度を下げる (フェアシェア)，課金がある場合は値段の低い組み合わせを選択するなどが考えられる．資源提供者側の視点では，大きなクラスタをあとからくるリクエストのために残すようにする，分散した資源が平等に利用されるようにする，利用効率が高まるように割当てるなどが考えられる．

さらに，資源の選択手法も 2 通り考えられる．予約可能なリソースから順次確保していく方法と，全体の情報をはじめに収集して予約プランを何通りか作り，その中から全ての資源が確保できるものを探す方法がある．前者は部分解☆が比較的早く見つかるが，一部の資源が確保できなかった場合にロールバックを繰り返しながら次の候補を探していくため，GRS と CRM および NRM 間のメッセージ数が増大してしまい，さらにそのメッセージをオーバーラップさせることはできない．後者は，1 つの資源の組み合わせに対して CRM および NRM に対して送られるメッセージ数を減らし，また並行して問い合わせることができるが，部分解への収束に時間がかかる．

現状では，デッドラインまでのできるだけ早い時刻にコアロケーションすることを目的とし，利用可能な計算機の台数を考慮して深さ優先探索で資源を確保し，最初に得られた計算・ネットワーク資源の組合わせを選択する．

☆ 最適な資源の組み合わせの基準がないため，部分解とする

表 2　GRM の予約関連サービスオペレーション

オペレーション名	機能	入力	出力
GRSReservationFactoryService			
createResource	GRSReservationResource を生成する	(開始時間，) 予約期間，デッドライン，資源に関する要求	EPR
GRSReservationService			
reserve	実際に予約を行う	なし	なし
getResourceProperty(status)	予約の状況を取得する	リソースプロパティ名	予約状況
getResourceProperty(GridResources)	予約された資源情報を取得する	リソースプロパティ名	資源情報
release	予約した資源を解放する	なし	なし
GRSExecutionService			
submit	ジョブを各 CRM へ送信する	ジョブ情報	なし
cancel	submit したジョブの実行をキャンセルする	なし	なし

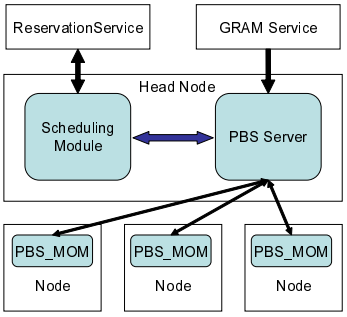

図 7　PluS ローカルスケジューラの概要.

表 5　CRMReservationReservation のプロパティ

リソースプロパティ名	意味
StartTime	予約開始時刻
EndTime	予約終了時刻
NodeNum	予約 CPU 数
ReserveId	予約 ID
ResultStdout	コマンド標準出力

5. 計算資源マネージャ(CRM) の実装

5.1 CRM の概要

事前予約機能を持つスケジューラモジュールを実装し，これを既存ローカルスケジューラ TORQUE と組み合わせて使用することで，事前予約可能なローカルスケジューラ PluS を開発した[12]．**図 7** に PluS の概要を示す．PluS の WSRF ベースの外部インタフェースを GT4 を用いて実装し，CRM を構成した．

5.1.1 PluS

TORQUE は，PBS サーバ，スケジューラモジュール，PBS mom の 3 種のモジュールから構成される (図 7)．PBS サーバはジョブのキューや資源情報を管理する中心サーバであり，スケジューラモジュールは PBS サーバからの依頼をうけて，ジョブに対するリソースの割り当てを決定する．TORQUE の提供するデフォルトスケジューラモジュールは独自のスケジューラモジュールで置き換えることが可能である．

我々は，Java 言語で事前予約機能を持つスケジューラモジュールを実装した．このモジュールは予約テーブルを内部にもち，予約情報を管理する．テーブルはデータベースによって永続化される．予約操作はスケジューラモジュールと RMI 接続するコマンドで行う．

5.1.2 GT4 によるインタフェース

図 7 に示すように，PluS は大きく分けて 2 種類のインタフェースを持つ．予約操作インタフェースとジョブ操作インタフェースである．このうち後者に関しては，GT4 が GRAM インタフェースを提供しているため，これを用いる．予約操作インタフェースは，独自に WSRF によるインタフェースを定義，実装した．

5.2 WSRF の CRM サービスインタフェース

CRM のサービスインタフェースとして，予約に対応する CRMReservationService と，これを生成する CRMReservationFactoryService を実装した．各サービスのオペレーションを**表 4** に示す．

CRMReservationFactoryService は createReservation と getAvailableResources の 2 つのオペレーションを提供する．createReservation は，予約に必要な，開始時間，終了時間，CPU 数などの情報を受け取り，ReservationService のインスタンスを生成し，それに対する参照となる EPR を返す．このとき，実際の予約はまだ行われていない．また，getAvailableResources は指定された時間帯に利用可能な資源の情報を返す．

CRMReservationService には 4 つのオペレーションがある．予約，解放，変更のためのオペレーション reserve，cancel，modify は，PluS の提供する予約コマンドに対応しており，サーバ側でそれぞれの予約コマンドを起動する．また，これらのオペレーションは操作のトリガに過ぎず，操作の出力はサービスのリソースプロパティに反映されるため，出力は持たない．**表 5** に CRMReservationResource の主なリソースプロパティを示す．オペレーションを実行すると，サーバ側で予約コマンドが実行される．コマンドの実行が終了すると，それぞれのプロパティの値が更新される．

5.3 CRM スケジューリングアルゴリズム

グローバルな協調動作を前提とした環境下のローカルスケジューラのスケジューリングアルゴリズムはそれ自身研究の対象である．提案システムの現在の実装

表 4 CRM の予約関連サービスのオペレーション

オペレーション名	機能	入力	出力
CRMReservationFactoryService			
createReservation	CRMReservationResource を生成する	開始時間，終了時間，CPU 数	生成したサービスの EPR
getAvailableResources	利用可能な資源の情報を提供する	開始時間，終了時間	資源情報
CRMReservationService			
reserve	実際に予約を行う	なし	なし
getResourceProperty(status)	予約情報を取得する	リソースプロパティ名	予約情報
cancel	予約をキャンセルする	なし	なし
modify	予約を変更する	開始時間，終了時間，CPU 数	なし

表 6 予約・キャンセルに要する時間の比較 (数値の単位は [sec])

(1)PluS RMI からの所要時間			
reserve	0.78		
cancel	0.68		
(2)CRM WSRF インタフェースからの所要時間			
予約手続き	1.7		
キャンセル手続き	1.3		
(3)GRS からの所要時間			
	全所要時間	CRM	差分
予約手続き	1.9	1.6	0.27
キャンセル手続き	1.6	1.4	0.18

は予備的段階にあり，さまざまな問題を持つことが明らかになっている．

PluS では，事前予約だけでなく通常のキューベースのジョブ投入も可能であるが，キューイングされたジョブと事前予約されたジョブでは，後者が完全に優先される．すなわち，キューイングされたジョブが実行中であっても，予約された時間がくれば，ジョブは直ちに停止されキューに戻される．

また，事前予約にはユーザのプライオリティ等は反映されず，First Come First Served となる．すなわち，どんなに高プライオリティのユーザであっても，低プライオリティユーザの予約を解除して自らの予約を入れることはできない．これは，予約が解除されたことを GRS に確実に伝える方法を確立できていないことと，予約が解除された場合に GRS がどのような挙動を示すべきかが明らかでないためである．

6. 予 備 評 価

提案するグリッドスケジューリングシステムについて，次の予備評価を行う．

6.1 基本性能の評価

4.3 節で述べたように，GRS のコアロケーションでは，分散した複数の CRM に対する指定した時刻の資源情報を問い合わせや，資源の予約・解放の手続きを繰り返すため，各 CRM や NRM でのオペレーションのレイテンシが GRS でのスケジューリング時間に大きく影響する．ここで，スケジューリング時間はユーザが GRSReservationService の reserve オペレーションを呼び出してから要求された資源の事前予約が全て完了し，status プロパティが QUEUED になるまでの時間をさす．よって，予備評価では (1)PluS RMI インタフェースからの reserve，cancel に要する時間，(2)CRM(PluS) WSRF インタフェースから予約，キャンセル手続きに要する時間，(3) ユーザが GRS に対して計算資源の予約・キャンセル要求を送り，GRS が PluS の WSRF インタフェースから予約・キャンセル手続きを完了させるまでの時間を比較する．(2) では GT4 の認証のオーバヘッドも含まれる．

評価では，GRS のサービスコンテナ，CRM のサービスコンテナ，PluS のヘッドノードを 1 つのノードで実行し，PluS の管理するノード数を 2 台とした．各ノードは Pentium III 1.4GHz，2CPU で，メモリ 2Gbytes，OS は RedHat ver. 8 となっている．各サービスコンテナは GRS では no security モード (認証無し)，CRM は security モード (認証あり) で起動した．

表 6 に予約・キャンセルに要する時間の比較結果を示す．値はそれぞれ 10 回呼び出した際の平均値を [sec] で表している．表 6(3)GRS からの所要時間の「CRM」「差分」は，GRS の内部から CRM に対して予約・キャンセルを行った際に要した時間，差分は全体の所要時間から「CRM」の時間を除いた時間，すなわち GT4 WSRF の手続きのオーバヘッドを表す．

表 6 において，(2) と (3) の予約とキャンセルの所要時間を比較すると，予約の方が 0.3-0.4[sec] 程度大きい．これは予約の手続きでは Factory サービスを呼び出すためである．また，(1) と (2) のキャンセル手続きの比較により，security モードで実行したときの WSRF に要するオーバヘッドが 0.6[sec] 程度であることが分かる．一方，(3) では，no security モードで実行したときの WSRF のオーバヘッドが 0.2[sec] 程度となっていた．

GRS のスケジューリングでは，指定された時間空間から分散した資源を予約しなければならないため，CRM や NRM に対して利用可能状況や予約やキャンセルの問い合わせを繰り返す．また，CRM の PluS 自体のオーバヘッド，WSRF オーバヘッドに加えて GRS と CRM，NRM 間の通信遅延も加わるため，大規模環境でのスケジューリング時間は長くなってしまう．スケジューリング時間を短くするよう，GRS のスケジューリングアルゴリズムの検討が必要である．

6.2 ネットワーク資源管理システムとの連携

GRS が NRM と連携してユーザの要求する計算資

源と性能保証されたネットワーク資源を，事前予約によりコアロケーションし，提供することが実現可能であることを示すため，2005 年 9 月に米国サンディエゴで開催された iGrid2005[13] で実証実験を行った[2]．実験では，会場に設置した GRS，つくば，秋葉原，上福岡，金沢，大阪，福岡に設置されたクラスタ計算機と，拠点間を結ぶ総長 1260km のネットワークテストベッド，および KDDI 研究所で開発した NRM[3] を用いた．本実験により，GRS が NRM と連携し，ユーザの要求を満たす計算・ネットワーク資源を事前予約して，適宜構成・提供できることを示した．

7. 関 連 研 究

VIOLA は本研究同様，グリッドで計算資源とネットワーク資源を同時に提供することを目的としている[14]．VIOLA では GGF において標準化がすすめられている WS-Negotiation/-Agreement 仕様に基づくメタスケジューラを UNICORE[15] ベースのグリッド環境で開発することを提案している．しかしながら，実装の詳細は明らかでない．

グリッドスーパースケジューラでは Silver[16]，CSF[17]，NAREGI スーパースケジューラ[18] など，複数開発されているが，その多くが実用段階には至っていない．また，計算資源の割当てが主体であり，性能が保証されたネットワーク資源と計算資源のコアロケーションを実現しているものはない．

8. まとめと今後の課題

本研究では，計算資源とネットワーク資源を同時確保する事前予約をベースのグリッドスケジューリングシステムを設計・開発した．本システムでは，計算資源と通信性能が保証されたネットワーク資源を自動的にコアロケーションし，提供することができる．本研究の成果により，メタコンピューティングを実行するための安定した環境をユーザの要求に応じて自動的に構成し，提供する事が可能となる．

今後は複数資源マネージャに対する同時予約・修正・解放処理を実現するため，2 フェーズコミットをベースとしたプロトコルを導入し，GRS と CRM の開発を進めるとともに，NRM との連携を考慮した GRS と CRM における適切なスケジューリングアルゴリズムを検討する．

謝辞 G-lambda プロジェクトの皆様に感謝いたします．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) G-lambda: `http://www.g-lambda.net/`.
2) 竹房，林，長津，中田，工藤，宮本，大谷，田中 (英)，鮫島，今宿，神野，滝川，岡本，田中 (良)，関口：G-lambda: グリッドにおける計算資源と光パスネットワーク資源のコアロケーション，情報処理学会研究報告 2006-HPC-105，pp.121–126 (2006).
3) Hayashi, Miyamoto, Otani, Tanaka, Takefusa, Nakada, Kudoh, Nagatsu, Sameshima, and Okamoto: Managing and Controlling GMPLS Network Resources for Grid Application, *Optical Fiber Communications Conference (OFC) 2006* (2006).
4) OpenPBS: `http://www.openpbs.org/`.
5) TORQUE Resource Manager: `http://www.clusterresources.com/resource-manager.php`.
6) OASIS Web Services Resource Framework (WSRF) TC: Web Services Resource 1.2 (WS-Resource) Committee Specification (2005).
7) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit., *International Journal of Supercomputer Applications*, pp.115–128 (1997).
8) Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems, *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, pp.2–13 (2005).
9) Grid Engine: `http://gridengine.sunsource.net/`.
10) OASIS Web Services Notification (WSN) TC: Web Services Base Notification 1.3 (WS-BaseNotification) Public Review Draft 02 (2005).
11) Ganglia: `http://ganglia.info/`.
12) 中田，竹房，大久保，岸本，工藤，田中，関口：事前予約機能を持つローカルスケジューリングシステムの設計と実装，情報処理学会研究報告 2006-HPC-105，pp.217–222 (2006).
13) iGrid2005: `http://www.igrid2005.org/`.
14) H. Barz: Dynamic allocation of network resources in VIOLA, *VIOLA workshop* (2005).
15) Romberg, M.: The UNICORE Architecture: Seamless Access to Distributed Resources, *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, pp.287–293 (1999).
16) Moab Grid Scheduler (Silver) Administrator's Guide version 4.0: `http://www.clusterresources.com/products/mgs/docs/`.
17) Community Scheduler Framework: `http://sf.net/projects/gcsf`.
18) 畑中，中野，井口，大野，佐賀，秋岡，中田，松岡：OGSA アーキテクチャに基づく NAREGI スーパースケジューラの設計と実装，情報処理学会研究報告 2005-HPC-102，pp.33–38 (2005).