

多様な資源を事前予約で同時確保するための グリッドコアロケーションシステムフレームワーク GridARS

竹房 あつ子[†] 中田 秀基[†] 工藤 知宏[†]
田中 良夫[†] 関口 智嗣[†]

グリッドで並列高性能計算の性能を向上させるには、コアロケーションシステムが既存資源スケジューラと連携して多様な資源を事前同時予約できることが重要である。本稿では WSRF に基づく事前予約インタフェースを提供するコアロケーションフレームワーク GridARS を提案する。GridARS では分散トランザクションによる同時予約手続きを行うため、汎用的な 2 相コミット事前予約プロトコルを設計し、実装した。GridARS の有効性を示すため、複数資源マネージャに対する同時予約手続きの基礎性能を評価した。また、応用事例として複数資源スケジューラで管理される日米間に跨るネットワーク、計算資源を同時確保する実証実験について報告する。これにより、1)GridARS の 2 相コミットによる予約手続きが有効であり、2)GridARS のコアロケーションシステムが、多様な資源のスケジューラと連携し、安定して分散資源を同時確保できることを示す。

GridARS: An Advance Reservation-based Grid Co-allocation System Framework for Simultaneous Reservation of Various Resources

ATSUKO TAKEFUSA,[†] HIDEMOTO NAKADA,[†] TOMOHIRO KUDOH,[†]
YOSHIO TANAKA[†] and SATOSHI SEKIGUCHI[†]

For high performance parallel computing on actual Grids, one of the important issues is to co-allocate the distributed resources that are managed by various local schedulers with advance reservation. To address the issue, we proposed and developed the GridARS resource co-allocation framework, and a general advance reservation protocol that uses WSRF/GSI and a two-phased commit (2PC) protocol to enable a generic and secure advance reservation process based on distributed transactions, and provides the interface module for various existing resource schedulers. To confirm the effectiveness of GridARS, we describe the performance of a simultaneous reservation process and a case study of GridARS grid co-allocation over transpacific computing and network resources. Our experiments showed that: 1) the GridARS simultaneous 2PC reservation process is scalable and practical and 2) GridARS can co-allocate distributed resources managed by various local schedulers stably.

1. はじめに

グリッドでは、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（メタコンピューティング）が実現できる。しかしながら、メタコンピューティングで MPI で実装されるような並列アプリケーションの実効性能を高めるには、分散した計算機やネットワーク、ストレージ等の多様な資源を同時に確保（コアロケーション）し、性能が保証された環境でユーザのプログラムを実行する必要がある。一方、現状ではメタコンピューティングは次のよう

に実現されている。

- (1) 紳士協定 + SSH 紳士協定により占有環境を用意し、SSH 経由でのジョブ起動を行う。
 - (2) 手動予約手続き + 既存キューイングスケジューラ 各クラスタはキューイングスケジューラで管理されており、電子メール等の人的な予約手続きの後、各管理者がクラスタごとに手動で専用キューを設定し、それらのキューにジョブを投入する。
 - (3) 自動予約手続き + 既存キューイングスケジューラ 各クラスタは事前予約機能を持つキューイングスケジューラで管理されており、スーパースケジューラが自動でコアロケーションし、ユーザがクラスタごとの予約キューにジョブを投入する。
- グリッドチャレンジテストベッド¹⁾では(1)の方針

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

で運用されたが、一般には計算機の有効利用ができず、他の人が利用しているノードにログインできてしまうなど、信頼性の面でも問題がある。また、利用可能なすべての環境において各ユーザがローカルアカウントを持っているという前提は現実的でない。TeraGrid²⁾は(2)の方針で運用されており、クラスタをサイト(組織)ごとのポリシーにしたがって管理することができるが、人的な手続きによる設定ミス等の問題も数多く報告されている。(3)は(2)同様現実的な運用ができ、人的ミスの問題も回避できるが、異なるサイトの異なるキューイングスケジューラの資源を自動的に確保してジョブを投入するため、2節で述べるような技術的課題が複数ある。

我々は、既存資源スケジューラと連携して分散した計算機やネットワーク等の資源を事前同時予約するグリッドコアロケーションフレームワーク GridARS(Grid Advance Reservation-based System framework)を提案する^{3),4)}。また、多様な資源の確保のため、WSRF(Web Service Resource Framework)⁵⁾に基づく汎用的な2相コミット事前予約プロトコルを設計し、その実装を行った。

GridARSの想定するアーキテクチャはグローバルスケジューラ(GRS)と資源マネージャ(RM, 資源スケジューラ)からなり、それらはWSRFインタフェースを介して多様な資源をユーザの要求に応じて自動的に割当てる。また、階層的な2相コミットを実現しているため、トランザクション処理による複数資源の同時予約手続きが可能である。

GridARSは、ユーザの要求に従って適切な資源群を探し、トランザクション処理により同時予約手続きを行うGridARS-Coschedulerと、ユーザ-GRS間、GRS-RM間でWSRF/GSIによる2相コミットによる事前予約手続きを可能にするGridARS-WSRFインタフェースモジュールからなる。GridARS-WSRFの実装では、WSRFの参照実装の1つであるGlobus Toolkit 4(GT4)⁶⁾を用いた。

GridARSのトランザクションによる予約手続きの有効性を示すため、GRSと複数RMを用いて、WSRF/GSIでの2相コミットの基礎性能の評価を行った。また、GridARSの導入事例としてG-lambda⁷⁾と米国Enlightened Computing⁸⁾プロジェクトが共同で行った計算・ネットワーク資源の日米間コアロケーション実験について紹介し、GridARSの実用性について述べる。

2. グリッドコアロケーションの技術的課題

各組織やネットワークのドメイン内でローカルユーザに効率的に資源を提供しつつ、グローバルユーザにグリッドコアロケーション環境を提供するには、次の要件を満たす必要がある。

既存キューイングスケジューラとの連携

メタコンピューティングで利用されるクラスタ計算機等の資源は、ローカルユーザに対しても提供されており、資源を有効利用するために、多様なキューイングスケジューラで管理されている。よって、既存キューイングスケジューラと連携しつつ、グローバルユーザに資源を提供しなければならない。

事前予約 通常の資源スケジューラはFCFS等で先に投入したジョブから資源を割当てるため、ジョブ投入してから実行開始までの時間が一定でない。グローバルユーザが複数の資源を無駄なく同時に確保するには、事前予約機能が必要である。

多様な資源への対応 高性能計算では、計算資源だけでなく通信帯域など他の資源の性能も保証しなければならない。よって、異なるインタフェースで提供される資源を同時に確保しなければならない。

WSRF/GSI 複数組織の提供する資源を利用する際、すべての資源上に各ユーザのローカルアカウントがあるとは限らない。よって、グローバルユーザに対して安全で標準的なインタフェースで資源を提供するため、標準化が進められているWSRF(Web Services Resource Framework)⁵⁾およびGSI(Grid Security Infrastructure)に基づく通信が求められる。

2相コミット 分散する複数資源をグローバルスケジューラが同時に確保するため、各資源に対する事前予約手続きをトランザクションとして処理する必要がある。そのためには、各資源スケジューラは2相コミットをサポートしなければならない。例えば、図1で示すように、1相コミットで分散する資源の予約時刻をトランザクション処理により同時に修正するケースを考える。まず、(1)ユーザが予約時刻の修正要求をコアロケータに送ると、コアロケータから各資源マネージャ(RM)に修正要求が送られる。(2)既にある予約によりRM2の修正手続きは失敗してしまうが、他のRMでは修正処理が行われる。よって、(3)元の予約時刻に戻すロールバック処理が行われる。しかしながら、(4)RM1のようにその間に他の予約が入ってしまうとロールバックに失敗するという致命的な問題が発生する。また、課金等が発生する場合には、各予約手続きを慎重に

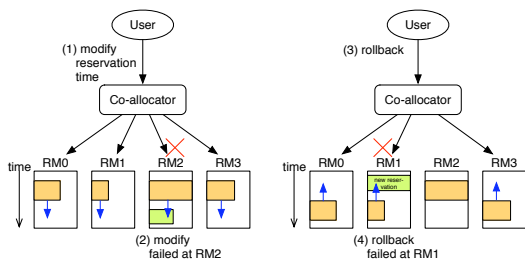


図 1 1 相コミットでのロールバック失敗例
Fig. 1 An example of rollback failure by one-phase commit.

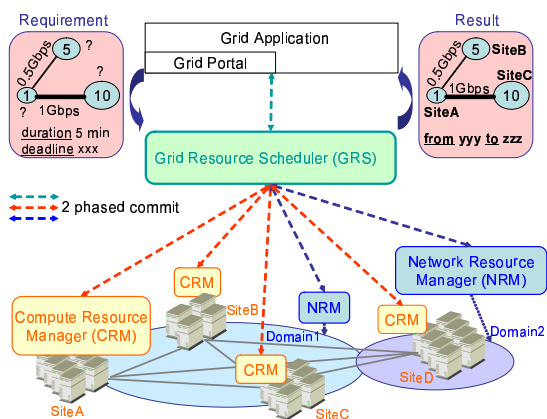


図 2 GridARS コアロケーションフレームワークの概要
Fig. 2 Overview of the GridARS co-allocation framework.

行わなければならない。

よって、これらの技術的要件を満たす、グリッドコアロケーションフレームワークを設計・実装する。

3. GridARS コアロケーションフレームワーク

我々は、広域に分散し、かつ異なる組織により管理される資源を確保するためのグリッドコアロケーションフレームワーク GridARS(Grid Advance Reservation-based System framework) を提案・開発した。GridARS は以下の特徴をもつ。

- 事前予約ベース
- 既存資源スケジューラと連携可能
- 計算・ネットワーク資源に対応
- WSRF/GSI インタフェースの提供
- 2相コミット (トランザクション処理) のサポート

GridARS の想定するコアロケーションフレームワークを図 2 に示す。本フレームワークは主にグローバルスケジューラ (GRS) と計算資源マネージャ(CRM)、ネットワーク資源マネージャ(NRM) 等の資源マネー

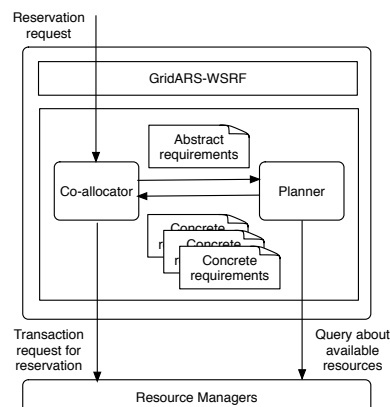


図 3 GridARS-Coscheduler アーキテクチャ
Fig. 3 GridARS-Coscheduler System Architecture.

ジャ(RM) により構成される。各 RM では、事前予約機能を有する既存資源スケジューラが資源の予約状況进行管理している。各既存資源スケジューラは、資源管理者の予約ポリシーに従ってローカルユーザとグローバルユーザの予約を受け付ける。予約手続き完了後は、予約時刻にそのユーザが予約資源を占有できることを各資源スケジューラが保証する。GridARS では、ユーザが計算機と計算機間のネットワーク、時刻に関する要求を GRS に送ると、GRS が関連する複数の CRM, NRM と連携し、各 RM の事前予約インタフェースを介して各資源を適切に事前同時予約することにより、予約時刻に複数分散資源の利用を可能にする。

図中のユーザ-GRS 間、GRS-CRM 間、GRS-NRM 間の破線では、WSRF に基づく 2 相コミットでの事前予約手続きがなされる。これにより、GRS から分散する各資源の予約手続きをトランザクションで実現できる。また、階層的な 2 相コミット構造を提供しているため、GRS 自身も資源マネージャの一つとなりうる。すなわち、スケーラビリティのために GRS を階層的に構成したり、他のグループのグリッドスケジューラとの連携も容易に行える。

提案する GridARS の GRS は、ユーザの要求に従って事前同時予約を行う GridARS-Coscheduler と、2 相コミットを行う GridARS-WSRF インタフェースモジュールからなる。また、各 RM はインタフェースモジュールと既存資源スケジューラで構成される。

3.1 GridARS-Coscheduler

図 3 に示すように、GridARS-Coscheduler はコアロケータとプランナからなる。コアロケータは、GridARS-WSRF を介してユーザの計算機・ネットワーク等の資源と予約時刻に関する要求を受取り、そ

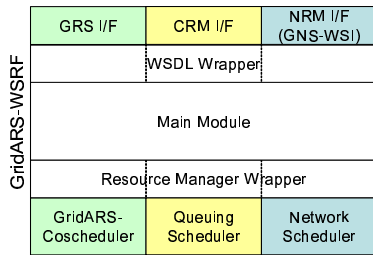


図 4 GridARS-WSRF アーキテクチャ

Fig. 4 GridARS-WSRF System Architecture.

れをプランナに渡す。プランナは、受け取った抽象的な予約要求に対して、各 RM に問い合わせた該当時刻の資源情報と照らし合わせ、具体的な資源 (すなわち RM) 情報と時刻を含めた要求リストを作成し、コアロケータに返す。ここで、各 RM の予約情報を情報サービスで一元管理し、情報サービスから予約情報を取得する方法も考えられるが、特に商用の資源プロバイダでは資源の利用情報を公開できないため、GridARS では各 RM に直接問い合わせる方法をとる。また、プランナは目的に応じて入れ替えることができる。

コアロケータは、プランナから受け取った具体的な予約要求に従って、トランザクション処理により指定された RM に対して同時に予約手続きを行う。手続きの詳細は 4 節で述べる。また、コアロケータは予約手続き後、確保した資源の状態を定期的に監視する。

3.2 GridARS-WSRF

GridARS-WSRF はポーリングベースで 2 相コミット事前予約をするためのインタフェースモジュールを提供する。ポーリングベースの場合、クライアント、サーバ間の通信回数が増える、状態変化の検知が遅れるなどの欠点があるが、クライアントがファイアウォール内にいるといった非対称なケースでも通信が可能である。また、ウェブサービスでサーバからクライアントに通知するための WS-Notification 仕様⁹⁾も提案されているが、これを用いる場合もクライアントから定期的にサーバの状態を調べることが求められる。よって、GridARS ではポーリングを基本とし、WS-Notification に基づくインタフェースは補助的に用いる。

GridARS-WSRF は、WSDL ラップ、メインモジュール、資源マネージャラップで構成される。図 4 に GRS, CRM, NRM での例を示す。WSDL ラップでは、多様な資源のための WSDL の差異を吸収する。事前予約の手続き (予約・修正・解放) は各資源とも共通であるが、予約する資源のパラメータ情報は異なる。多様な資源を考慮して WSDL を統一する方法、

パラメータ部分は文字列として扱い、受け取った後に文字列を解析する方法も考えられるが、前者は統一 WSDL の策定に膨大な時間がかかる、また両者とも異なる資源の予約を受け付けてしまうため、WS インタフェースレベルで誤った要求を排除することができない。よって、WSDL は資源ごとに定義したものを採用する。

メインモジュールでは、サービスのクライアントと GridARS-WSRF 間、GridARS-WSRF とローカル資源スケジューラ間でポーリングベースの 2 相コミットによる予約・修正・解放手続きを実現している。クライアントから予約オペレーションが実行されると、ノンブロッキングでクライアントに予約オペレーションの成功を知らせ、資源スケジューラに対して予約手続きを行う。資源スケジューラが予約準備完了状態になった後に、クライアントからの予約完了オペレーションで予約手続きを完了させる。分散システムにおいて、予約手続きのように応答までに時間がかかる場合、手続きの途中でサーバまたはクライアント側に何らかの原因によりダウンする可能性もあるため、ノンブロッキングでの手続きをサポートすることが重要である。ノンブロッキングとすることで、サーバがダウンした場合はクライアントがサーバのエラーを検知できる、クライアントがダウンした場合はクライアントが復旧後に引き続き手続きが行えるといった利点がある。

また、既存資源スケジューラは、エラー発生や資源状態の変化などの情報通知をしないため、メインモジュールから定期的な状態確認と状態遷移を検知するためのポーリングを行う。これにより、GridARS-WSRF 内と資源スケジューラ内の状態が一致するため、クライアントが資源の状態を知ることができる。

資源マネージャラップでは、GridARS-Coscheduler および各ローカル資源スケジューラに対してラップ API を提供している。この API を実装することで、既存資源スケジューラに WSRF/GSI ベースのインタフェースを提供することができる。また、GridARS に組み込むことで、他のサイト/資源との事前同時予約が可能になる。

4. GridARS-WSRF の詳細設計と実装

G-lambda で提案する事前予約インタフェース GNS-WSI¹⁰⁾ を基に、GridARS-WSRF の WSRF ベースの 2 相コミット事前予約プロトコルを設計した。GridARS-WSRF は以下のサービスからなる。

ReservationFactoryService 各資源予約のための初期登録を受け付ける。利用可能な資源情報の問い合わせ

せに対する応答も行う。

ReservationService 資源予約・予約修正・予約資源解放手続きと、現在の予約状況や予約資源の状態の管理を行う。

ReservationCommandService 2相コミットのため、仮予約・修正・解放手続きの状態管理と、各手続きの確定/破棄手続きを行う。

ReservationService に対する各予約要求のサービスインスタンスを **ReservationResource**, **ReservationCommandService** に対する各手続きのインスタンスを **ReservationCommandResource** と呼び、実際には各 **Resource** が上記の処理を行う。

4.1 サービスオペレーション

表 1 に各サービスにおける予約関連オペレーションの一覧を示す。 **ReservationFactoryService** は予約情報を管理する **ReservationResource** の生成と、利用可能な資源を問い合わせるためのオペレーションを提供する。 **create** オペレーションは、生成したリソースに対する参照である **EPR(Endpoint Reference)** を返す。ここで返される **EPR** を **rsvEPR** とする。

ReservationService には資源の予約/修正/解放と、予約資源情報の取得、割当て結果取得のためのオペレーションがある。予約および修正手続きでは、予約時刻に関する要求(利用時間, デッドラインなど)と、資源に関する要求を受け取る。予約開始時間はユーザが指定することもできる。また、資源に関する要求には、クラスタ数, 各クラスタの 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。このとき、実際の予約処理はまだ行われず、予約手続きを管理する **ReservationCommandResource** の生成とその参照である **EPR (cmdEPR とする)** を返す。 **reserve**, **modify**, **release** は操作のトリガに過ぎず、各手続きの進行状況は **ReservationCommandResource** のリソースプロパティである **ReservationCommandStatus** に反映される。

ReservationCommandService では、予約/修正/解放手続き状況の取得と、各手続きの確定/破棄手続きのためのオペレーションを提供する。これにより、**WSRF** での 2 相コミット予約手続きを実現している。

4.2 状態遷移と事前予約プロトコル

個々の資源要求に関する情報は、**ReservationResource** で管理する。**ReservationResource** には **ReservationStatus** というプロパティがあり、これにより予約資源の遷移状態を管理する。**ReservationCommandResource** では、**ReservationResource** から発行された各予約手続きを管理する。**ReservationCommand-**

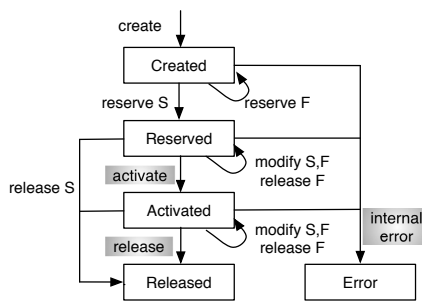


図 5 ReservationStatus の状態遷移
Fig. 5 The ReservationStatus transition process.

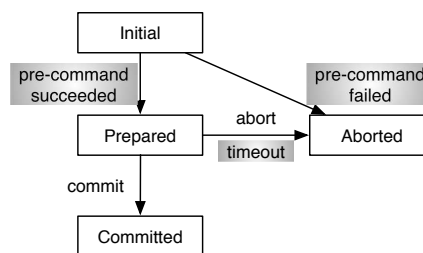


図 6 ReservationCommandStatus の状態遷移
Fig. 6 The ReservationCommandStatus transition process.

dResource の **ReservationCommandStatus** プロパティでは、予約手続きの状態遷移を管理する。

図 5 に **ReservationStatus** の状態遷移を示す。各状態は以下のとおりである。

- Created ReservationResource 生成完了
- Reserved 資源予約完了
- Activated 予約資源が利用可能
- Released 予約資源解放
- Error エラー発生

図 5 中の **create**, **reserve**, **modify**, **release** はクライアント側から発行する表 1 のオペレーションを表し、**S** および **F** は各手続きの成功, 失敗または破棄手続きをした場合を示す。灰色の四角で囲まれた箇所は、サーバ側での資源の状態変化を表す。

図 6 に **ReservationCommandStatus** の状態遷移を示す。各状態は以下のとおりである。

- Initial 仮手続き未完了
- Prepared 仮手続き完了
- Committed 手続き完了
- Aborted 仮手続き失敗または手続き破棄

図 6 中の **commit**, **abort** はクライアントが発行する手続き、灰色の四角で囲まれた箇所は、サーバ側での手続きの状態変化を表す。**ReservationCommandStatus**

表 1 予約関連サービスオペレーション

Table 1 Service operations related to reservation, modification, and release.

オペレーション名	機能	入力	出力
ReservationFactoryService(ReservationFactoryService の URL にアクセス)			
create	ReservationResource を生成する	なし	rsvEPR
getAvailableResources	利用可能な資源の情報を取得	資源の条件	利用可能な資源情報
ReservationService(rsvEPR を用いてアクセス)			
reserve	資源予約手続き	予約時間, 資源に関する要求	cmdEPR
modify	予約資源の変更	予約時間, 資源に関する要求	cmdEPR
release	予約資源の解放	なし	cmdEPR
getResourceProperty(ReservationStatus)	予約資源の状況を取得	リソースプロパティ名	予約資源状況
getResourceProperty(GridResources)	予約された資源情報を取得	リソースプロパティ名	資源情報
ReservationCommandService(cmdEPR を用いてアクセス)			
commit	予約/修正/解放手続きの確定	なし	なし
abort	予約/修正/解放手続きの破棄	なし	なし
getResourceProperty (ReservationCommandStatus)	仮予約/修正/解放手続き状況を取得	リソースプロパティ名	仮手続き状況

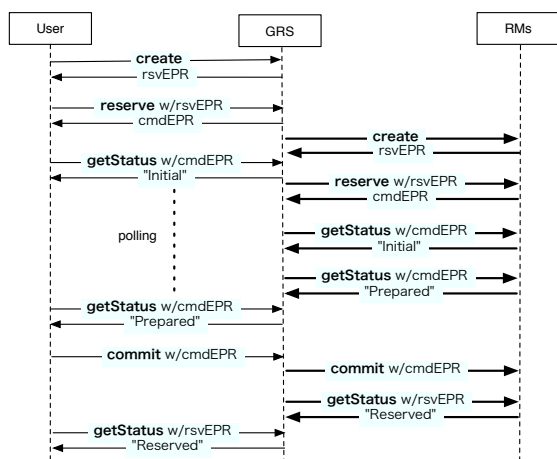


図 7 事前予約プロトコルシーケンス

Fig.7 Protocol sequence of advance reservation process.

が Prepared になった後、クライアント側から手続きの確定・破棄を行う。

4.3 事前予約プロトコルシーケンス

ユーザから GRS を介して各 RM に対して事前予約手続きを行う際のプロトコルシーケンスを図 7 に示す。3.2 節で述べたように、各手続きはノンブロッキングでポーリングベースで行う。

事前予約では、ユーザの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 RM と連携し、適切な資源を事前予約する。ユーザが GRS の ReservationFactoryService で提供する create オペレーションを呼び出すと、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 EPR(rsvEPR) が返される。次に、ユーザがその rsvEPR を用いて reserve オペレーションを呼び出す

と、GRS が資源のコアロケーションを開始する。

GRS は複数 CRM および NRM の ReservationFactoryService に対して getAvailableResources オペレーションで、要求された時間帯の利用可能な CPU 数や帯域等の資源情報を得る。その結果から、要求された資源を保有する複数 RM に対して、トランザクション処理で同時に事前予約手続きを行う。図 7 中で GRS, RM 間の太字の矢印はトランザクション処理を行っていることを意味する。GridARS-WSRF を用いた GRS と各 RM 間の手続きは、ユーザと GRS 間と同様に行うことができる。GRS はまず create で各 RM で ReservationResource を生成し、rsvEPR を受け取る。次に、その rsvEPR を用いて reserve を呼び出し、指定した時間帯の指定した CPU 数または帯域を要求する。その際、各 RM はこの予約要求に関する情報を管理する ReservationCommandResource を生成し、cmdEPR を返す。

すべての RM の ReservationCommandStatus が Prepared となると、GRS は GRS の ReservationCommandStatus も Prepared に変更し、ユーザからの commit または abort 要求を待つ。その後、ユーザは Prepared 状態を検知すると、GRS に対して commit を送り、GRS がこれを受け取った後に各 RM に対して commit 要求を送信する。

全ての資源の事前予約が完了すると GRS および各 RM の ReservationStatus が Reserved に、失敗すると Error になる。ユーザは getResourceProperty オペレーションで各ステータス情報を取得することで、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取得することができる。

4.4 GridARS-WSRF の実装

GridARS-WSRF の一実装として、Globus Toolkit4 (GT4) を用いた GridARS-WSRF/GT4 を開発した。GridARS-WSRF/GT4 では、GT4 の提供する GSI (Grid Security Infrastructure) を用いたユーザの認証・認可が可能である。ユーザの認証には PKI に基づく証明書を用いる。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。また、GRS では、GT4 のデレゲーション機能を用いて要求したユーザの権限で各 RM に対して予約手続きを行うことができる。

GridARS では NRM のインタフェースとして、G-lambda プロジェクトで提案している GNS-WSI (Grid Network Service - Web Services Interface)^{(11),(12)} を採用している。G-lambda は産総研、KDDI 研究所、NTT、NICT の共同プロジェクトであり、グリッドミドルウェアやアプリケーションに対してネットワーク資源を事前予約で提供するためのウェブサービスインタフェース GNS-WSI を既定している。これにより、ネットワーク資源との連携が可能となる。また、CRM に対しては、キューイングスケジューラにジョブを投入する際の記述方法として標準化が進められている JSDL (Job Submission Description Language)⁽¹³⁾ を事前予約用に拡張したものを、各手続きのパラメータとして採用した。

5. 評価実験

提案する 2 相コミット事前予約プロトコルの有効性を示すため、同時予約処理に要する所要時間と、1 相コミット処理でのロールバック失敗率を調査した。

5.1 同時予約処理性能

GridARS のトランザクションによる事前予約手続きの実用性を示すため、GridARS を用いて、1 つの GRS から 1-8 つの RM に対して同時予約手続きを行った際の所要時間を調査した。実験では、クラスタ内の 1 ノードにユーザと GRS、8 ノードにそれぞれ RM を 1 つずつ用意し、いずれも 1 つのネットワークスイッチで接続されている。GRS のネットワークインタフェースは 100Mbps、全 RM は 1Gbps であり、GRS のホスト構成は Xeon 2.4GHz 2 基、4GB メモリ、CentOS4.3、全 RM のホスト構成は Pentium 4 2.8GHz、1GB メモリ、CentOS4.3 となっている。クラスタにおける GRS と RM のホスト間は 200us 程度であるが、実環境でのコアロケーションをエミュレートするため、一部またはすべての RM の各オペレー

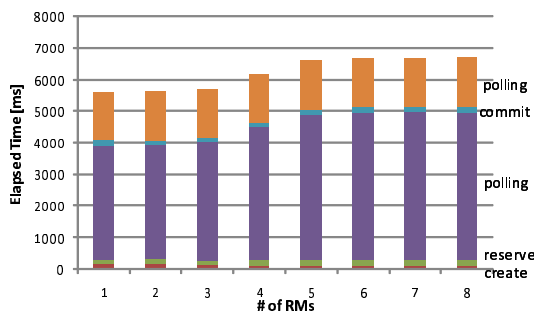


図 8 資源同時予約手続きの所要時間 (遅延なし).

Fig. 8 Elapsed time of simultaneous resource reservation processes (no additional latency).

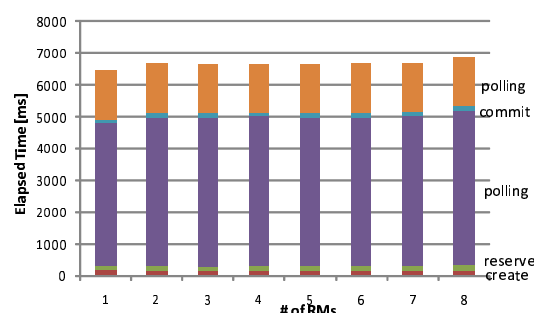


図 9 資源同時予約手続きの所要時間 (1 つめの RM のみ遅延あり).

Fig. 9 Elapsed time of simultaneous resource reservation processes (additional latency on the path to RM1).

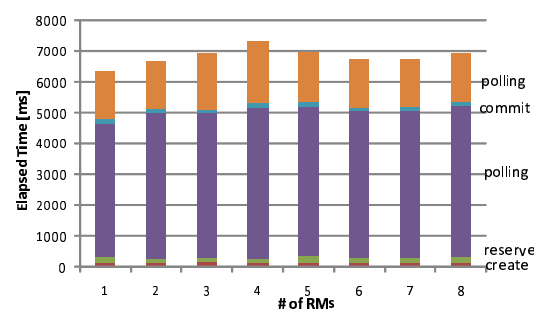


図 10 資源同時予約手続きの所要時間 (全 RM で遅延あり).

Fig. 10 Elapsed time of simultaneous resource reservation processes (additional latencies on the paths to all RMs).

ションに対して往復 186ms の遅延を設定した。これは、6 節の実験で GRS が配置された秋葉原と RM の一つが配置された米国ローリー (ノースカロライナ) との間の遅延と同じである。各 RM では、いずれも予約手続きに 2sec、コミット手続きに 1sec かかるものとした。

実験結果を図 8、図 9、図 10 に示す。図 8 は遅延を設定しなかった結果、図 9 は 1 つめの RM にのみ遅延

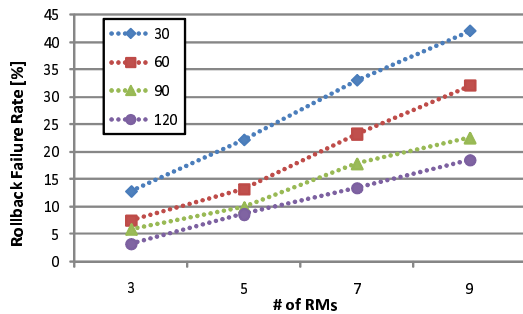


図 11 1 相コミットでのロールバック失敗率

Fig. 11 The failure rate for rollback by one-phase commit.

を設定した結果、図 10 は 8 つの RM とも遅延を設定した結果を示す。グラフの横軸は同時に手続きする RM の数、縦軸は所要時間を ms で表し、その内訳を右側に示している。create/reserve/polling/commit/polling は図 7 のユーザからの各手続きに対応している。グラフの値はそれぞれ 10 回の計測の平均値であり、いずれも GSI で通信している。

3 つのグラフを比較すると、図 8 に対して図 9 と図 10 はいずれも同程度所要時間が長くなっている。特に、RM 数が少ない場合では全 RM が仮予約手続き完了となるまでのポーリングの時間が長くなっている。すなわち、トランザクションでは一番遅い RM の影響が所要時間に大きく反映してしまうことが示されている。一方、いずれのグラフとも、同時に手続きする RM 数が増えると GRS での負荷が高くなって所要時間が長くなるものの、最長でも 7.3 秒程度であった。よって、日米間のように広域に分散する RM が複数存在するような環境でも、トランザクションによる予約手続きは $O(1)$ であり、GridARS の予約手続きは有効であることが示された。

5.2 1 相コミットでのロールバック失敗率

提案する 2 相コミットの予約修正手続きでは、完了手続きが終わるまで予約資源を確保するため、図 1 で指摘したような予約修正失敗後のロールバックの失敗は起こりえない。この実験では、1 相コミットで予約修正手続きが失敗した場合、どの程度の確率でロールバックが失敗するかを複数 RM が存在する環境で調査し、2 相コミットプロトコルがいかに有効であるかを示す。実験は分散シミュレータを開発して行った。

シミュレーション環境は以下の通りである。

- 予約可能な時間帯は 80 時間
- 各ユーザは 1 時間単位で予約する
- サイト数は 3, 5, 7, 9
- 各サイトのノード数は 8

他のユーザの予約手続きに関する設定を以下に示す。

- 予約の平均投入間隔は 30, 60, 90, 120 秒でポアソン到着
- 予約時間は 80 時間の中から 1-3 時間をランダムに決定
- 予約ノード数は 1, 2, 4 ノードのいずれかをランダムに決定

このような環境で、全サイトの全ノードを 9 時間占有する予約をあらかじめ入れておき、1 分ごとに 8 時間前後に変更する修正要求を送る。変更失敗した場合は 6 秒後にロールバックを試みる。各 10 分間のシミュレーション時間内に発生するロールバック失敗数を 1000 回の試行から測定し、その割合を示す。

図 11 に 1 相コミットでのロールバック失敗率を % で示す。横軸には RM 数を示す。グラフの 4 つの折れ線は、他のユーザの予約の平均投入間隔を示しており、上から 30, 60, 90, 120 秒となっている。平均投入間隔が短いほど、単位時間あたりに多くの予約が投入される。

図 11 から、予約の平均投入間隔が短くなるほど、また同時に手続きするサイト数が増えるほど、ロールバック失敗率が高くなるのが分かる。一番条件の良い RM 数 3、投入間隔 120 秒の場合でも 3.3%、条件の悪い RM 数 9、投入間隔が 30 秒では 42.1% にも達していた。よって、複数の RM と連携して資源を同時確保する場合、2 相コミットによる手続きが必須であることが示された。

6. GridARS を用いた応用事例

日米間に跨る異なる組織・サイトの、異なるスケジューラで管理される計算およびネットワーク資源を、GridARS を用いて各 RM と連携して事前予約し、予約資源上で並列アプリケーションを実行する実証実験を行った¹²⁾。この実験は利用するネットワークが複数組織（ドメイン）に分かれている場合、ドメイン間を連携させてドメインを跨るネットワークの帯域を予約して提供する世界で初めての試みとして、G-lambda と米国 Enlightened Computing プロジェクトが GLIF2006¹⁵⁾ および SC06¹⁶⁾ で共同して行った。本実験から、GridARS がネットワーク資源との連携を実現している点、5 節で行った 2 相コミットによる事前予約手続きを実際に日米間に跨る複数組織から提供される RM との間で行える点、そのような環境でも安定して動作する点を示す。

6.1 ポータルの概要

実験では、ポータルシステムを構築し、ポータルか

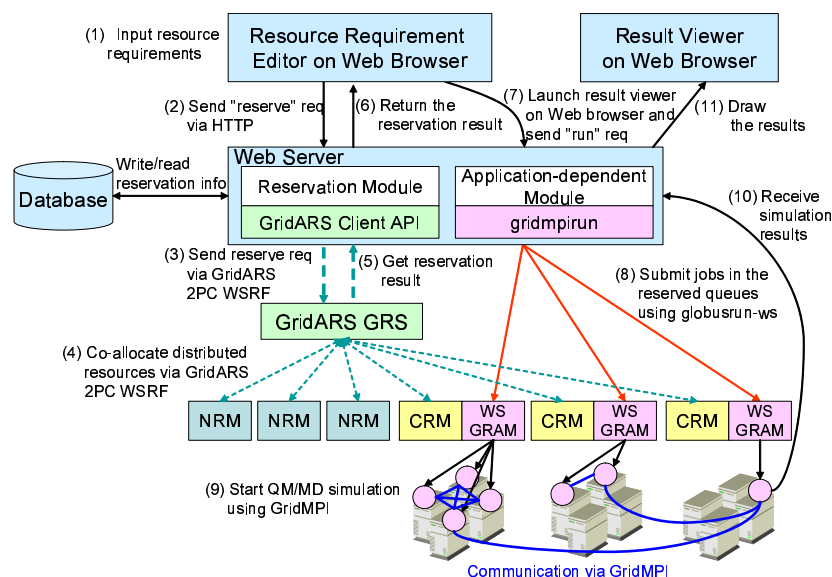


図 12 ポータルの概要

Fig. 12 Overview of the portal system.

ら GridARS を用いて複数の資源スケジューラが管理する計算・ネットワーク資源を事前予約し、GT4 の WS GRAM 経由で予約資源上にジョブを投入し、予約時刻に自動的に並列アプリケーションを実行した。アプリケーションプログラムには、GridMPI¹⁴⁾ で実装された QM/MD 連成シミュレーションを用いた。QM/MD 連成シミュレーションは、量子力学・分子動力学を用いて化学反応過程を推測するもので、反応過程の各点を並列に実行することができる。GridMPI は、グリッド環境で利用可能な MPI の参照実装である。

図 12 にポータルの概要を示す。事前予約のための HTTP インタフェースを定義し、ポータルのフロントエンドは Java Applet と JavaScript で実装した。ブラウザからの資源の予約要求に応じて、ウェブサーバを介して GridARS を用いた事前予約手続きを行う。GridARS の GRS が日米に分散する CRM, NRM と連携して適切な資源を確保する。次に、GridMPI の提供する gridmpirun コマンドにより GT4 WS GRAM を介してあらかじめ予約したキューへのジョブ投入を行う。各資源スケジューラが予約時刻に自動的に資源を利用可能とする。ポータルからの予約手続きでは GridARS のインタフェース、アプリケーションの実行では WS GRAM を利用しているため、SSH 等で直接ログインすることなく、予約した時刻に広域に分散する資源上で並列プログラムを同時に起動して実行することができる。

6.2 日米間資源予約実証実験

実証実験の環境は以下の通りである。

- サイト (クラスタ) 数: 10 (国内 7 拠点, 米国 3 拠点)
- CRM の構成: GridARS-WSRF/GT4 と Plus^{17),18)} + SGE¹⁹⁾ または Maui²⁰⁾ + TORQUE²¹⁾
- ネットワーク管理ドメイン数: 4 (国内 3, 米国 1)
- NRM の構成: KDDI 研究所, NTT, Enlightened, 産総研がそれぞれ実装した 4 つの NRM. Enlightened と産総研の NRM では GridARS-WSRF/GT4 を利用。

米国側の CRM, NRM は、Enlightened のコアロケーションシステム HARC²²⁾ を介して各手続きが行われるが、日本側の CRM, NRM と同じインタフェースを持つラップを介しているため、同様に GRS からトランザクションによる予約手続きが行える。

図 13 に実験の予約資源モニタ画面と並列アプリケーションの実行結果 (右下) を示す。実験では、繰り返し日本側 3 拠点と米国側 1 拠点の 4 つの CRM とその間のネットワークを管理する 4 つの NRM (国内 3, 米国 1) に対して同時に予約要求を送り、要求時刻の 3 分後に分散する資源を 10 分間確保し、予約完了後にジョブを投入して予約時刻に QM/MD 連成シミュレーションを実行した。実証実験のため、通常の利用よりも短い期間で予約・ジョブ投入・実行を繰り返したが、デモ期間中 GridARS に起因するトラブルが発生することなく、安定して予約手続きを行うこと

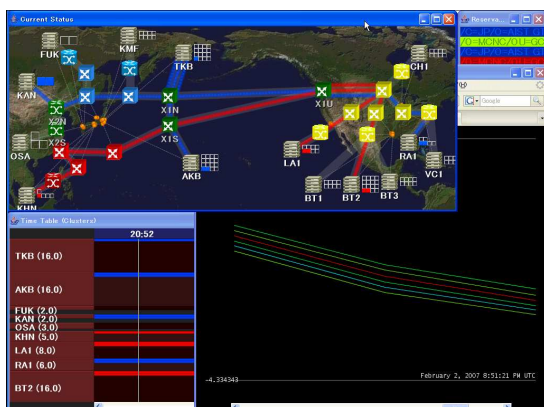


図 13 予約資源モニタ画面とシミュレーション結果出力画面

Fig.13 The reservation resource monitor service display and the simulation results output at the experiment.

ができた。また、日米間における実験での予約手続きも RM 内での処理が 5.1 節の設定より多少長かったものの、8 秒程度で完了し、5.1 節で示した予約手続きの有効性を確認した。本実験から、GridARS ではネットワーク資源との連携を実現し、日米間に分散する複数組織から提供される RM に対する事前同時予約手続きも安定して行えることが実証された。

7. 関連研究

Moab²⁰⁾ では、既存キューイングスケジューラと Cluster Workload Manager(Maui の後継) 外部スケジューラで管理される計算機を、Grid Workload Manager でコアロケートする。現在 Moab は商用 Moab Grid Suite として、モニタリング・レポートツール、エンドユーザ用ポータルなどとともに提供されている。通常、管理者ユーザのみが予約処理を行えるが、エンドユーザも専用ポータルから計算機の予約・ジョブ実行をすることもできる。

CSF²³⁾ では、商用キューイングスケジューラ LSF で管理される計算機に対して、メタスケジューラ経由でジョブを投入する。CSF バージョン 4.0.x では、GT4 を用いて WSRF インタフェースを提供する。LSF は事前予約機能があるため、ユーザが各サイトの計算機を時刻を指定してコアロケーションすることは可能であるが、CSF 自体はコアロケーションを行わない。LSF では Java ベースのウェブアプリケーションクライアントである CSF4 Portlet も提供している。

GUR²⁴⁾ は既存キューイングスケジューラに事前予約機能を付与する Catalina 外部スケジューラと連携し、順次予約手続きをしてユーザの要求する資源を事

前予約する。旅行代理店のように分散する資源を提供するグリッドスケジューラである。GUR-Catalina 間の通信は SSH または GSI-enabled SSH で行う。

Enlightened で用いている HARC (Highly-Available Robust Co-scheduler) では、Acceptor と呼ばれるコアロケータと資源マネージャ間で Paxos コミットプロトコル²⁵⁾ を用いており、Acceptor 側の耐故障性を高めている。HARC では各要求を XML で記述し、そのドキュメントを Acceptor、資源マネージャ間で HTTP で授受している。

NAREGI プロジェクト^{26),27)} では、複数サイトの計算資源をコアロケートするスーパースケジューラを開発している。スーパースケジューラは WSRF に基づく 1 相コミットの予約インタフェースを提供しており、ワークフローとの連携も特徴的である。現状ではネットワーク資源との連携は行われていない。

以上から、WSRF で 2 相コミットによるトランザクション予約手続きをサポートし、かつ、実際にネットワーク資源との連携も行っているものは GridARS のほかにない。

8. まとめと今後の課題

本研究では、分散した多様な資源を事前同時予約するグリッドコアロケーションフレームワーク GridARS を提案し、多様な資源の確保のための汎用的な事前予約プロトコルの設計とその参照実装を行った。GridARS では、コアロケーションを行う GridARS-Coscheduler と WSRF での 2 相コミット予約手続きを実現する GridARS-WSRF により、メタコンピューティングのための技術的要件を満たすコアロケーションシステムを構築することができる。

GridARS を用いた評価実験では、GRS と 8 つの RM を用いて、WSRF/GSI での 2 相コミットの基礎性能を示した。遅延のある環境でも 7.3 秒程度で 8 つの RM に対して同時予約処理を行うことができ、その有効性を示した。1 相コミットでのロールバック失敗率のシミュレーションによる調査では、RM 数が少なく予約投入頻度が低い場合でも 3.3% ロールバックが失敗してしまい、分散する資源のコアロケーションでは 2 相コミットが必須であることを示した。また、GridARS の応用事例では、既存資源スケジューラと連携して日米間に跨る計算・ネットワーク資源を自動的に確保し、MPI アプリケーションを実行する実験について紹介し、GridARS の実用性を示した。

今後の課題として、以下があげられる。まず、事前予約ベースのグリッドコアロケーションシステムは複

数提案されているが、その利用方法は統一されていない。よって、多様な資源の事前予約手続きや表現方法に関する標準化が重要である。また、デバッグや監視のため、各ユーザの確保した資源の情報を安全に提供する情報サービスも必要である。

謝辞 G-lambda プロジェクトの皆様には感謝いたします。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) 合田憲人ほか: グリッドチャレンジテストベッドの構築と運用～グリッドチャレンジテストベッドの作り方～, 情報処理学会研究報告 2006-HPC-107, pp. 49-54 (2006).
- 2) TeraGrid: <http://www.teragrid.org/>.
- 3) 竹房, 中田, 工藤, 田中, 関口: 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, 先進的計算基盤システムシンポジウム SACSIS2006 論文集, pp.93-100 (2006).
- 4) 竹房, 中田, 武宮, 松田, 工藤, 田中, 関口: 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, HPCS2007 2007 年ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, pp.135-142 (2007).
- 5) OASIS Web Services Resource Framework (WSRF) TC: Web Services Resource 1.2 (WS-Resource) Committee Specification (2006).
- 6) 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).
- 7) G-lambda プロジェクト: <http://www.g-lambda.net/>.
- 8) EnLIGHTened Computing プロジェクト: <http://enlightenedcomputing.org/>.
- 9) OASIS Web Services Notification (WSN) TC: Web Services Base Notification 1.3 (WS-BaseNotification) Public Review Draft 02 (2005).
- 10) Takefusa, A., Hayashi, M., Hirano, A., Okamoto, S., Kudoh, T., Miyamoto, T., Tsukishima, Y., Otani, T., Nakada, H., Tanaka, H., Taniguchi, A. and Sameshima, Y.: GNS-WSI2 Grid Network Service - Web Services Interface, version 2, OGF19, GHPN-RG (2007).
- 11) 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(2006), pp.868-875 (2006).
- 12) 竹房, 林, 築島, 岡本, 柳田, 宮本, 平野, 鮫島, 中田, 谷口, 工藤: ミドルウェア連携による計算・ネットワーク資源の日米間グリッドコアロケーション実験, 情報処理学会研究報告 2007-HPC-109, pp.281-286 (2007).
- 13) A. Anjomshoaa and F. Brisard and M. Drescher and D. Fellows and A. Ly and S. McGough and D. Pulsipher and A. Savva: Job Submission Description Language (JSDL) Specification v1.0 (2005).
- 14) GridMPI: <http://www.gridmpi.org/>.
- 15) GLIF: Global Lambda Integrated Facility: <http://www.glif.is/>.
- 16) SC06: <http://sc06.supercomputing.org/>.
- 17) 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).
- 18) 中田, 竹房, 岸本, 大久保, 工藤, 田中, 関口: 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, 情報処理学会研究報告 2006-HPC-107, pp.55-60 (2006).
- 19) Grid Engine: <http://gridengine.sunsource.net/>.
- 20) Moab Grid Scheduler (Silver) Administrator's Guide version 4.0: <http://www.clusterresources.com/products/mgs/docs/>.
- 21) TORQUE Resource Manager: <http://www.clusterresources.com/resource-manager.php>.
- 22) HARC: The Highly-Available Robust Co-allocator: <http://www.cct.lsu.edu/~maclaren/HARC/>.
- 23) Community Scheduler Framework: <http://sf.net/projects/gcsf>.
- 24) Yoshimoto, K., Kovatch, P. and Andrews, P.: Co-scheduling with User-Settable Reservations, *Job Scheduling Strategies for Parallel Processing*, Springer Verlag, pp. 146-156 (2005). Lect. Notes Comput. Sci. vol.3834.
- 25) Gray, J. and Lamport, L.: Consensus on Transaction Commit, Msr-tr-2003-96, Microsoft Research (2004).
- 26) Matsuoka, S., Shimojo, S., Aoyagi, M., Sekiguchi, S., Usami, H. and Miura, K.: Japanese Computational Grid Research Project: NAREGI, Vol.93, No.3, pp.522-533 (2005).
- 27) 中田, 佐藤, 佐賀, 畑中, 佐伯, 松岡: NAREGI ミドルウェア β -gLite 間における相互ジョブ起

動実験, 情報処理学会研究報告 2006-HPC-109 (2007).

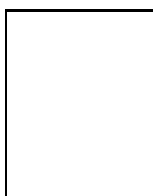
(平成?年?月?日受付)

(平成?年?月?日採録)



工藤 知宏 (正会員)

1991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, 1997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, 2002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。情報処理学会, IEEE CS 各会員。



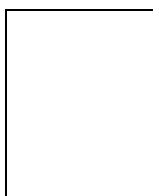
竹房あつ子 (正会員)

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年より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研究員。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。



田中 良夫 (正会員)

昭和40年生。平成7年慶応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 最優秀論文賞, 2006年度情報処理学会論文賞。ACM 会員。



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員。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員。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。



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員。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。情報処理学会, 日本応用数理学会, ソフトウェア科学会, SIAM, IEEE 各会員。