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

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[†]

グリッドで並列高性能計算を行うには，資源のコアロケーションシステムが異種の複数スケジューラで管理される資源を安全に事前同時予約できることが重要である．本稿では WSRF に基づくインタフェースを提供するコアロケーションシステム GridARS を改良し，2 相コミットでの事前同時予約手続きを実現した．また，GridARS の有効性を示すために，計算資源と通信資源のコアロケーションを必要とする実アプリケーションプログラムのためのポータルを GridARS を用いて構築し，複数の異種スケジューラで管理される日米間に跨るネットワーク，計算資源を用いて実験を行った．この結果，1)GridARS を用いることで，異種スケジューラで管理される資源群をトランザクションにより安全に確保できること，2)GridARS を用いたポータルにより，容易にグリッド上での高性能計算環境をユーザに提供できること，を確認した．

Excecution Environment for Grid High Performance Computing over Widely-distributed Resources Managed by Heterogeneous Resource Schedulers with Advance Reservation

Atsuko Takefusa,[†] Hidemoto Nakada,[†] Hiroshi Takemiya,[†]
Motohiko Matsuda,[†] Tomohiro Kudoh,[†] Yoshio Tanaka[†]
and Satoshi Sekiguchi [†]

For high performance parallel computing on the actual Grids, one of the important issues is to co-allocate distributed resources that are managed by various local schedulers with advance reservation. To address the issue, we improved the GridARS resource co-allocator, so that it uses two phased commit (2pc) protocol to enable generic and secure advance reservation process. To confirm effectiveness of the GridARS, we also developed a portal system for real application programs and performed an experiment using transpacific computing and network resources. Our experiment showed that: 1) GridARS could co-allocate distributed resources managed by various local schedulers and 2) the portal system was useful to make it easy to perform high performance Grid computing for users.

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 (グリッド高性能計算) 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．

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

各組織やネットワークのドメイン内で各ユーザ (ローカルユーザ) に有効に資源を提供しつつ，グリッド高性能計算のユーザ (グローバルユーザ) の要求に応えるには，次のような要件をみたすグリッドコアロケーションシステムが求められる．

既存キューイングスケジューラとの連携 グリッド高性能計算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 通常のローカルスケジューラは 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．

2 相コミット　分散する資源を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，1 相コミットでは，分散する資源の予約時刻等を修正する際，一部の資源が修正不可能でも残りの資源では修正処理を行ってしまうため，ロールバックできなくなるという致命的な問題が発生する可能性がある．また，資源の予約や解放手続きについても，課金等が発生する場合には慎重に行わなければならなくなる．

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事前予約しなければならない

WSRF/GSI　グローバルユーザに対してセキュアで標準的なインタフェースを提供するため，また，資源管理者が各サイトのポリシに従ってグローバルユーザに資源提供するため，標準化が進められている WSRF(Web Services Resource Framework)[1] および GSI(Grid Security Infrastructure) をベースとする通信が求められる．

我々は既存スケジューラで管理される分散した計算機やネットワークを事前同時予約するグリッドコアロケーションシステム GridARS (Grid Advance Reservation-based System framework) を開発している[2]．GridARS はグローバルスケジューラ (GRS) と資源マネージャ（RM，ローカルスケジューラ）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．

本研究では，GridARS を改良して階層的な 2 相コミットによる安全な予約手続きを可能にし，上で述べた技術的要件を満たすコアロケーションシステムを構築した．また，GridARS を用いて並列アプリケーションを分散する資源上で実行するためのポータルを開発し，ユーザが安全かつ容易にグリッド高性能計算を実現できることを実証する．

改良した GridARS では，新たに実装した GridARS-WSRF インタフェースモジュールを用いて，ユーザ-GRS 間，GRS-RM 間で WSRF による 2 相コミットでの予約手続きを可能にする．GridARS-WSRF は Globus Toolkit 4(GT4)[3] で実装されているため，WSRF/GSI での標準的で安全な通信が可能である．また，GRS は分散する多様な資源を GridARS-WSRF を介してトランザクションにより自動的に事前同時予約でき，GRS 自身も資源マネージャの一つとなり得る．ユーザは，GridARS の提供するクライアントインタフェースを用いることで，容易に分散する資源を自動的に確保することができる．

開発したポータルでは，複数組織・サイトの，異なるスケジューラで管理される計算・ネットワーク資源を GridARS で事前予約し，その予約 ID を用いて GT4 の WS GRAM 経由でジョブ起動することで，SSH 等で直接ログインすることなくグリッド高性能計算を実行する．アプリケーションには，GridMPI[4] で実装された QM/MD 連成シミュレーションを用いた．また，事前予約機能のない既存キューイングスケジューラには，PluS[5),6)] と Maui[7] 外部スケジューラを用いた．

2. グリッド高性能計算の現状

グリッド高性能計算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 (グリッドチャレンジテストベッド[8])．

(2) 専用ミドルウェアでの自動予約・実行　専用ミドルウェアを用い，独自の方法で分散している計算機をシームレスにユーザに利用させる (NAREGI[9])．

(3) 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 SGE 等のキューイングスケジューラで管理されており，クラスタごとに電子メール等の予約手続きにより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 (TeraGrid[10])．

(4) 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 (GridARS[2]，CSF[11]，Moab[7]，GUR[12])．

(1) は，一般の運用で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．(2) はサイト全てに専用ミドルウェアをインストールする必要があるため配備コストが高く，また，組織間の強い連携が前提となる．(3) ではクラスタをサイト (組織) ごとのポリシ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．(4) は (3) 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にジョブを投入するため，(1)，(2) のようにジョブの実行に関するノウハウが明らかでない．よって，本研究では (4) の方式で 1 節で述べた技術的要件を全て満たすシステムを開発し，また，ポータルシステムの構築により提案システムにより容易にグリッド高性能計算が可能になることを実証する．

3. GridARS による事前同時予約

3.1 GridARS の概要

GridARS コアロケーションシステムの概要を**図 1** に

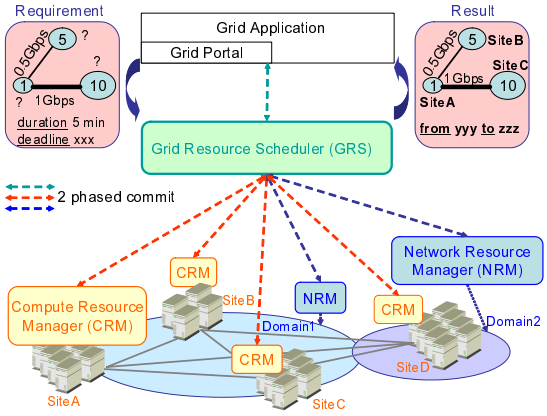

図 1 GridARS コアロケーションシステムの概要

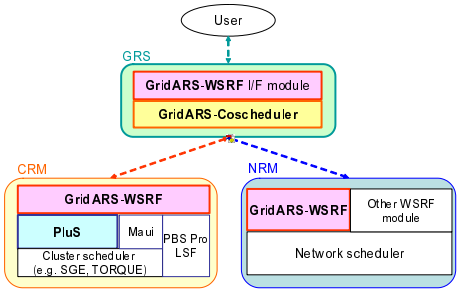

図 2 GridARS コアロケーションシステムアーキテクチャ

示す．GridARS 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) により構成される．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 が複数の CRM，NRM と連携し，事前同時予約により適切に資源を割当てる．図中のユーザ-GRS 間，GRS-CRM 間，GRS-NRM 間の破線では，本研究で開発した GridARS-WSRF を用いて 2 相コミットで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安全な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．

図 2 に GridARS のシステムアーキテクチャを示す．GRS は GridARS-WSRF と実際に事前同時予約を行う GridARS-Coscheduler で，各資源マネージャ(RM) は GridARS-WSRF とローカル資源スケジューラで構成される．ただし，各 RM が GridARS が採用している WSDL(Web Services Description Language) によりサービスを構築している場合，GridARS-WSRF を用いなくても GRS と連携することができる．GridARS では NRM に対して GNS-WSI インタフェース[13] を採用している．

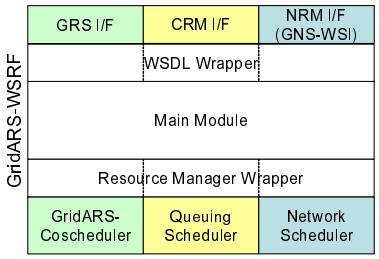

図 3 GridARS-WSRF アーキテクチャ

3.2 GridARS-WSRF

GridARS-WSRF は GT4 で実装されており，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モジュールを提供する．また，GSI と gridmapfile による認証・認可を行う．GridARS-WSRF は，WSDL ラッパ，メインモジュール，資源マネージャラッパで構成される (**図 3**)．

WSDL ラッパでは，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，WSDL の長所を殺してしまうことになる．よって，WSDL は資源ごとに定義したものを採用した．

メインモジュールでは，ユーザと 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．ユーザ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ユーザに予約オペレーションの成功を知らせ，資源マネージャには予約手続きをする．資源マネージャが予約準備完了状態になった後に，ユーザからの予約完了オペレーションで予約手続きを完了する．

既存のキューイングスケジューラ等の資源マネージャは，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 内と資源マネージャ内の状態が一致するようにする．また，ユーザは同様に GridARS-WSRF 内で管理される資源の状態を，ポーリングにより確認する必要がある．

資源マネージャラッパでは，GridARS-Coscheduler および各ローカル資源スケジューラに対して Java の API を提供している．この API を実装することで，既存資源マネージャに 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を可能にする．

```
EndpointReferenceType rsvEPR =
  GrsClient.create(GRS_FACTORY_URI);
EndpointReferenceType cmdEPR =
  GrsClient.reserve(rsvEPR, REQUIREMENTS);
// 資源の予約手続きが完了するまでポーリング
GrsClient.commit(cmdEPR);
```

図 4 事前予約手続き

```
EndpointReferenceType cmdEPR =
  GrsClient.modify(rsvEPR, REQUIREMENTS);
// 予約修正手続きが完了するまでポーリング
GrsClient.commit(cmdEPR);
```

図 5 予約資源の修正手続き

```
EndpointReferenceType cmdEPR =
  GrsClient.release(rsvEPR);
// 資源解放手続きが完了するまでポーリング
 GrsClient.commit(cmdEPR);
```

図 6 予約資源の解放 (キャンセル) 手続き

3.3 クライアントインタフェース

GridARS は WSRF に基づいており，GRS インタフェースの WSDL を元にクライアントスタブライブラリを生成・利用することができるが，ユーザの負荷が大きい．よって，Java クライアント API，コマンドインタフェース，シェルインタフェースを提供している．いずれも，WS-Addressing 仕様で決められているエンドポイントリファレンス (EPR) を用いて，各予約インスタンスを識別する．EPR にはサービスの URI と GRS 内でのサービスインスタンスの識別子が含まれている．

Java クライアント API の主な関数を以下に示す．

`create` 予約サービスインスタンスの生成
`reserve` 資源の予約手続き
`modify` 予約した資源の修正手続き (時刻，資源数，帯域等)
`release` 資源の解放手続き
`commit` 予約／修正／解放手続きの完了
`abort` 予約／修正／解放手続きの破棄

`reserve/modify/release` は 2 相コミットにおける仮予約/修正/解放手続きであり，これだけでは要求する手続きは完了しない．各手続きに対してその処理を完了させる場合は `commit`，破棄する場合は `abort` を呼び出す．ポーリングや 2 相コミットを隠蔽した Java クライアント API も提供している．

図 4，**図 5**，**図 6** に予約・修正・解放手続きの擬似プログラムを示す．図 4 の `GRS_FACTORY_URI` には，GridARS GRS で予約手続きインスタンスを生成する Factory サービスの URL を入力する．また，図 4，図 5 の `REQUIREMENTS` には，要求する資源情報が渡される．資源の表現では，計算資源は JSDL 仕様 ver. 1.0 を事前予約用に拡張したもの，ネットワーク資源は GNS-WSI を複数のドメインに対応できるように拡張したものを用いている．

擬似プログラムでは，`create` で事前予約要求に対して 1 つの EPR(`rsvEPR`) が生成される．また，`reserve/modify/release` の各呼び出しごとにそのコマンドの EPR(`cmdEPR`) が生成される．この `cmdEPR` が `commit/abort` の引数として渡される．

コマンドインタフェースおよびシェルインタフェースでは，手動または Java 以外の言語からの予約手続きをサポートする．それぞれ Java クライアント API の各関数に対応するコマンドを提供している．

4. グリッド高性能計算のポータル構築事例

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資源およびネットワークを GridARS で事前予約し，GT4 の WS GRAM 経由でジョブ起動して予約資源を利用するポータルシステムを構築した．アプリケーションプログラムには，GridMPI[4] で実装された QM/MD 連成シミュレーションを用いた．

4.1 QM/MD 連成シミュレーション

対象とするアプリケーションは，尾形らにより開発された量子力学/分子動力学 (QM/MD) 連成シミュレーションコード[14] を機能拡張して，Nudged Elastic Band 法 (NEB 法) に基づく化学反応計算を行うようにしたものである．NEB 法は，化学反応の開始および終了時の系の状態からその間の状態を推測し，各状態を最適化することにより位相空間内における化学反応経路を特定する．そのために，反応中の系の状態を位相空間内における点 (イメージ) として考え，それらの間をばねにより束縛した状態で，各点の取りうるエネルギー値を同時に最小化することを試みる．NEB 法を用いることにより，反応過程を表す各点の状態は並列に求められるため，モンテカルロ法等による従来手法より計算時間の短縮される．

開発したコードは，MD シミュレーションコード，QM シミュレーションコード，およびそれらのシミュレーションを制御するコントローラから構成される．コントローラは反応中の各点に対応する系における原子分布を推測し，MD,QM 両シミュレーションに与える．コントローラから与えられた系を構成する原子データに基づき，MD, QM 両シミュレーションコードは連携しながら系の持つエネルギーを計算する．コントローラは，両シミュレーションの計算結果を基に，よりエネルギーの低い系の原子分布を推測する．この過程を繰り返すことにより，反応経路が計算される．

シミュレーションにおいて，反応過程をあらわす複数の系の状態計算は並列に行われる．さらに，各点の計算自体も細粒度並列化されている．これらの並列計算は MPI により実装されている．今回の実験におい

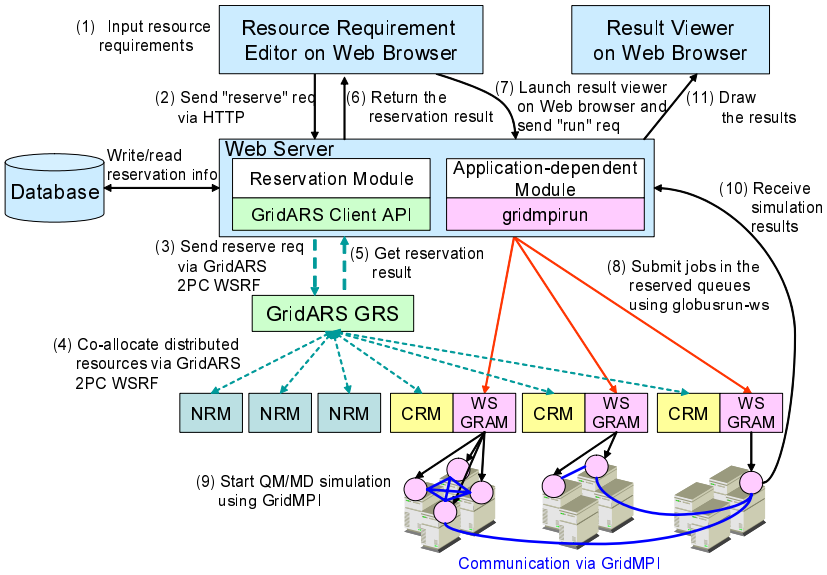

図 7 アプリケーションポータルの概要

ては，GridMPI を用いることにより本コードを Grid 環境上で実行可能とした．MPI で実装されていることから，本コードの Grid 環境上での実行においては，各クラスタに割り付けられたプログラムのコアロケーションが必須である．

実験においては，シリコン (Si) で構成された基盤中に酸素原子 (O) が混入した系を考え，Si-O 結合の変化により酸素原子が拡散する様子を計算した．系全体は 64001 個の原子から構成され，QM シミュレーションにより詳細解析される領域は酸素原子を含め 14 個の原子から構成されている．

4.1.1 GridMPI と gridmpirun ジョブ起動

GridMPI はグリッド環境で高性能な MPI アプリケーションを実行するために開発された MPI 実装である．特に，複数アーキテクチャの混在する環境への対応と TCP/IP での通信性能に重点を置いている．

GridMPI では，クラスタ用 MPI 実装に対して Interoperable MPI[15] (IMPI) 標準で定義される MPI 間相互通信プロトコルの拡張を行っている．IMPI 標準ではプロセス起動と通信プロトコルが規定されている．GridMPI ではクラスタ等の内部通信には独自プロトコルを使用し，クラスタ間通信に IMPI プロトコルを使用する．IMPI 標準ではプロセス起動を単純にするため，IMPI サーバーというプロセスを導入している．各クラスタは，IMPI サーバーを介して IP 接続を行うために必要になるアドレス・ポート情報の交換を行うように定義されている．

IMPI 標準に従う MPI アプリケーションの起動は，IMPI サーバーの起動と各サイトでの MPI プロセス起動の複数ステップからなり，煩雑である．そのため，GridMPI ではシェル・スクリプト `gridmpirun` を提供している．`gridmpirun` では，GT2，GT4 あるいは ssh を介した簡便な MPI プロセスの起動が可能である．`gridmpirun` は IMPI サーバーの起動後，コンフィギュレーション・ファイルの記述に従って RSL ファイルを生成し，`globusrun-ws` 等を用いて各サイトで MPI プロセスの起動を行う．今回のアプリケーションでは，予約情報に基づいてコンフィギュレーション・ファイルを生成し `gridmpirun` を呼び出している．

4.2 ポータルの概要

図 7 にアプリケーション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ており，ポータルのフロントエンドは Java Applet と Java Script で実装している．ブラウザからの資源の予約要求に応じて，GridARS を用いた資源の事前予約手続きと，`gridmpirun` コマンドによる GT4 WS GRAM を介した予約キューへのジョブ投入を行う．実行の流れを以下に述べる．

(1) ユーザが資源要求入力画面 (**図 8**) から資源に関する要求を入力する．

(2) ユーザが Reserve ボタンを押すと，事前予約 HTTP インタフェースを介して資源予約要求をウェブサーバに送信する．

(3) ウェブサーバが GridARS クライアント API を用いて WSRF で GridARS GRS に事前予約要求を送信する．また，GRS での事前同時予約の準備が完了したら，予約完了要求を送信する．

(4) ユーザの要求に応じて CRM，NRM に対して事前予約要求を送る．各 RM での予約準備が完了した後，ウェブサーバから `commit` が送られると，各 RM での資源の事前予約を完了させる．

(5) GridARS GRS での資源の事前同時予約が成功

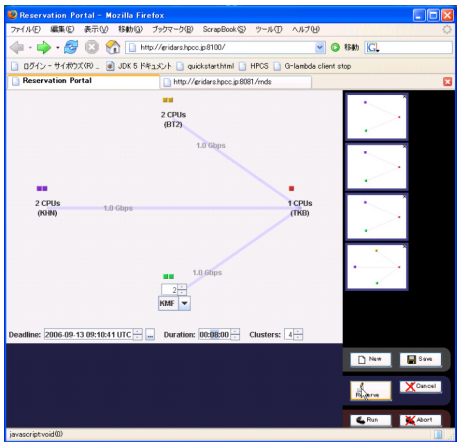

図 8　資源要求入力画面

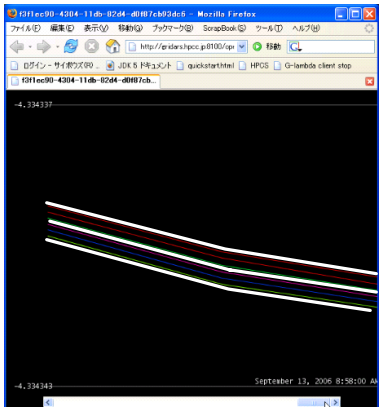

図 9　シミュレーション結果出力画面

すると，ウェブサーバが予約した資源に関する情報を取得し，データベースに格納する．

(6)　ウェブブラウザ側に，予約情報を返す．

(7)　ユーザが予約を確認した後 Run ボタンを押すと，シミュレーション結果出力画面ブラウザ (**図 9**) を開き，ジョブ起動要求を送信する．

(8)　ウェブサーバが予約資源に関する情報を渡して `gridmpirun` コマンドを実行すると，`gridmpirun` は IMPI サーバの起動し，`globusrun-ws` を用いて WS-GRAM 経由で各サイトのキューイングスケジューラにジョブを投入する．

(9)　予約時刻になると，各キューイングスケジューラが投入されたジョブを割当てられた計算機上で同時に起動し，QM/MD 連成シミュレーション計算を開始する．

(10)　コントローラに集められたシミュレーション結果をウェブサーバが随時取得する．

(11)　取得した結果をシミュレーション結果出力画面ブラウザ上に表示する (図 9)．

4.2.1　事前予約 HTTP インタフェース

本ポータルシステムでは，事前予約 HTTP インタフェースを定義して，通常の HTTP リクエスト (GET/POST) としてウェブブラウザクライアントとの事前予約に関する情報交換を行う．主な事前予約 HTTP インタフェースを**表 1** に示す．URL には呼び出し時にアクセスする URL(/の前はポータルの URL を指定)，機能，入力･出力パラメータは各インタフェースの機能，入力値，戻り値を示している．EPR は各予約の識別子となっている．

4.2.2　ウェブサーバの実装

本ポータルでは，軽量な Java 組み込み HTTP サーバである OOWeb[16] を用いた．OOWeb では Java オブジェクトをウェブページとしてマッピングすることができる．これにより，GridARS Java クライアント API を用いて事前同時予約することができる．

また，ウェブサーバは事前予約モジュールとアプリケーション依存モジュールで構成されている (図 7)．事前予約モジュールでは，GridARS クライアント API 経由で事前予約に関する手続きを行う．アプリケーション依存モジュールでは，取得した予約情報を元に `gridmpirun` を実行する．事前予約モジュールは各アプリケーションに依存することなくグリッド環境の事前予約に利用できる．よって，アプリケーション依存モジュールを入れ換えることで，容易の他のグリッド高性能計算のためのポータルを構築するができる．

4.2.3　セキュリティ

ユーザが資源要求入力画面を開く際に認証手続きを行うと，ウェブサーバがそのユーザ名とパスフレーズを用いて MyProxy[17] からログインしたユーザの証明書を取得することができる．GridARS 内では証明書を委譲するため，ウェブサーバがログインしたユーザの証明書を用いて事前予約手続きをすれば，そのユーザの権限で分散する資源の事前予約が可能となる．また，同様にジョブの投入も行えるため，ポータルからのシングルサインオンが実現できる．

4.3　本ポータルを用いた実験

異なるスケジューラで管理される分散するクラスタ計算機に対して，本ポータルからの事前同時予約およびジョブ投入する実験を 2006 年 9 月 11-13 日に G-lambda プロジェクト[18] と EnLIGHTened プロジェクト[19] が合同で行った．実験環境は以下の通りである．

- サイト数：9(国内 7 拠点，米国 2 拠点)
- CRM のキューイングスケジューラの構成：GridARS-WSRF, PluS + SGE[20] または Maui + TORQUE[21]
- 計算ノード：Linux 系 OS，X86
- ネットワーク：GNS-WSI インタフェースを提供する 4 つの NRM と連携

表 1　主な事前予約 HTTP インタフェース

URL	機能	入力パラメータ	出力パラメータ
/reserve	資源予約	資源予約要求	SUCCESS，EPR，資源予約結果 / FAILURE
/cancel	予約した資源の解放	EPR	SUCCESS / FAILURE
/list	要求／予約のリストの取得	N/A	保存されている要求／予約名のリスト
/load	要求／予約のロード	要求／予約名	要求／予約情報
/save	要求／予約のセーブ	要求／予約名，要求／予約情報	N/A

実験では，要求時刻の 3 分後☆に分散する資源を同時に 10 分間事前予約で確保し，予約完了後にジョブを投入して QM/MD 連携アプリケーションを実行した．通常の利用よりも短い期間で予約・ジョブ投入・実行を繰り返し行ったが，安定して動作した．また，この際 GridARS の GRS が 4 つの CRM，4 つの NRM に対して同時に `reserve/commit` を行っていたが，図 7 の (2) から (6) までの操作が 15 秒程度，(3)，(4) の手続き (GridARS での事前予約完了までの時間．結果の取得は含まない) で 8 秒程度 ($RsvTotal$) であった．

予約完了までのオーバヘッド $RsvTotal$ は `create`，ユーザ-GRS 間 `reserve`，GRS の状態を調べるポーリング，ユーザ-GRS 間 `commit` からなり，このポーリング期間では GRS と 8 つの RM 間での `reserve` と RM の状態を調べるポーリングが行われている．WSRF/GSI のオーバヘッドを a，各ポーリング回数を n 回 (遅延を含む資源予約準備にかかる手続きの最長時間により，n の値が変わる)，ポーリング間隔を b，`create` を含む GRS でのその他の処理を c とすると，$RsvTotal$ は次のようになる．

$$\begin{aligned} RsvTotal &= a + (a + a \times n + b \times (n-1)) \\ &\quad + a + c \\ &= 3a + (a+b)n - b + c \end{aligned} \tag{1}$$

ホスト内での WSRF/GSI のオーバヘッドは 0.6[sec][2)]，実験では $b = 1$，$n = 4$ だったので，6.2+c[sec] となっていた (RM 間の遅延を考慮すると，実際は 6.2[sec] より大きい)．よって，総 RM 数が 8 の場合においても 8[sec] 程度で事前予約手続きが完了していることから，2 相コミットにより安全かつ効率的に事前同時予約手続きが行えることを実証した．

また，日米間に跨るネットワーク・計算資源を用いた QM/MD 連成シミュレーションの実行も，ユーザは図 7 の (6) の処理のみで自動的に行うことができた．

5. 関 連 研 究

Moab[7)] 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・レポーティングツール，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ができる．

CSF[11)] 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 の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．独自に定義する `agreement`XML ファイルまたは `RSL` で書かれた XML ファイルにユーザが時刻やホストの情報を指定し，`csf-rsv-create` コマンドで計算機を事前予約する．また，`csf-job-submit` コマンドで割当て先の ResourceManagerFactoryService を指定してジョブを投入する．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．

GUR[12)] 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 の定義するメタジョブファイルに時刻や計算機に関する要求を記述し，`gur.py --submit` で事前予約，`gur.py --run` でジョブ投入する．GUR-Catalina 間の通信は GSI-enabled SSH で行う．

表 2 に GridARS と他のコアロケーションシステムとの比較結果を示す．表中のローカルスケジューラは構成されるローカルスケジューラを表し，ヘテロは既存キューイングスケジューラに対応していることを示す．GS-LS 間通信方法はグローバルスケジューラ（またはユーザ）とローカルスケジューラとの通信方法を現している．また，対象資源の項目では，スケジュールの対象となる資源 (実現しているもの) を表す．自動コアロケーション，2 相コミット，オープンソースの項目では，その有無を○，×，または△で示している．

6. まとめと今後の課題

本研究では，GridARS システムを改良して GridARS-WSRF による 2 相コミットでの事前同時予約手続きを実現し，グリッド高性能計算のための技術的要件を

☆ 実験時の資源マネージャ側のガードタイム (予約時刻に資源が確保できることを保証する) が最大 150[sec] であったため，3 分前とした．

表 2 自動予約手続き+既存キューイングスケジューラからなるコアロケーションシステムの比較 (GS: グローバルスケジューラ, LS: ローカルスケジューラ, C: 計算機, N: ネットワーク

	ローカル スケジューラ	自動コアロケーション	GS-LS 間 通信方法	ジョブ 投入	対象 資源	2 相 コミット	オープン ソース
GridARS	PluS+ヘテロ	○	WSRF/GSI	WS GRAM	C, N	○	○
Moab	Moab LS+ヘテロ	○	-/GSI	Pre WS GRAM	C	×	×
CSF	LSF	×	WSRF/GSI	独自 WSRF サービス	C	×	△ (GS のみ)
GUR	Catalina+ヘテロ	○	SSH(/GSI)	SSH(/GSI)	C	×	○

満たすコアロケーションシステムを開発した．また，GridARS を用いて MPI で実装される並列アプリケーションのポータルを構築し，複数の異種スケジューラで管理される日米間に跨るネットワーク，計算資源を用いて実験を行った．この結果，GridARS により異種の複数スケジューラで管理される資源をトランザクションにより安全に確保できることを示した．また，GridARS を用いたポータルにより，グリッド高性能計算環境を容易にユーザに提供できるを確認した．

今後の課題として，以下があげられる．まず，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，ポータルについては，ポートレットの適用が考えられる．ポートレットは自由なポータルの配置を可能にするため，これによりモニタリング等の様々な関連ツールの統合が容易に行える．

参 考 文 献

1) OASIS Web Services Resource Framework (WSRF) TC: Web Services Resource 1.2 (WS-Resource) Committee Specification (2006).
2) 竹房ほか：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).
3) 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).
4) 石川，松田，工藤，手塚，関口：GridMPI – 通信遅延を考慮した MPI 通信ライブラリの設計，情報処理学会研究報告 2002-HPC-95-17 (2003).
5) 中田ほか: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).
6) 中田ほか：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).
7) Moab Grid Scheduler (Silver) Administrator's Guide version 4.0: `http://www.clusterresources.com/products/mgs/docs/`.
8) 合田ほか：グリッドチャレンジテストベッドの構築と運用〜グリチャレ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).
9) 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).
10) TeraGrid: `http://www.teragrid.org/`.
11) Community Scheduler Framework: `http://sf.net/projects/gcsf`.
12) 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.
13) 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).
14) Ogata, S., Shimo, F., Kalia, R., Nakano, A. and Vashisha, P.: Hybrid Quantum Mechanical/Molecular Dynamics Simulations on Parallel Computers: Density Functional Theory on Real-space Multigrids, *ComputerPhysics Communications*, p.30.
15) W. L. George and J. G. Hagedorn and J. E. Devaney: IMPI: Making MPI Interoperable, *Journal of Research of the National Institute of Standards and Technology* (2000).
16) OOWeb: `http://ooweb.sourceforge.net/`.
17) Novotny, J., Tuecke, S. and Welch, V.: An Online Credential Repository for the Grid: MyProxy, *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press*, p.104 (2001).
18) G-lambda プロジェクト: `http://www.g-lambda.net/`.
19) EnLIGHTened プロジェクト: `http://enlightenedcomputing.org/`.
20) Grid Engine: `http://gridengine.sunsource.net/`.
21) TORQUE Resource Manager: `http://www.clusterresources.com/resource-manager.php`.