複数グリッドジョブ実行システムの計算資源を統合・利用するGrid RPCシステムの設計と実装

中島佳宏[†] 佐藤三久[†] 相田祥昭[†]
高橋大介[†] 朴泰祐[†] Franck Cappello[††]

OmniRPCはグリッド環境での並列プログラミングのためのGrid RPCシステムである．我々は，従来ワーカとの直接通信で行っていたOmniRPCの遠隔呼び出しをドキュメントベースの通信にすることにより，遠隔のグリッドジョブ実行システム上でのジョブとして実行できる機構を設計した．これにより，OmniRPCの対象とする計算資源を複数のグリッドジョブ実行システムで管理されている計算資源に拡大できる．また，これまでバッチシステムとして利用されてきたグリッドジョブ実行システム上の資源を，遠隔手続き呼び出しのプログラミングモデルで記述されたプログラムから利用可能になる．グリッドジョブ実行システムにおいてOmniRPCから利用するために必要な汎用なインタフェースについて検討，設計を行った．提案するシステムをXtremWeb，CyberGRIP，CondorとOpen Source Grid Engineの4つのグリッドジョブ実行システム上に実装し，予備的な性能評価を行った．提案システムにより，複数のグリッドジョブ実行システムが提供する計算資源を統合・利用可能にし，遠隔手続き呼び出しによるプログラミングモデルを用いて統一的に利用できるフレームワークを提供する．

Design and Implementation of Grid RPC System Integrating Computing Resources on Multiple Grid-enabled Job Execution Systems

Yoshihiro Nakajima,[†] Mitsuhisa Sato,[†] Yoshiaki Aida,[†]
Daisuke Takahashi,[†] Taisuke Boku[†] and Franck Cappello[††]

OmniRPC is a Grid RPC system for parallel programmings in a grid environment. In this paper, we propose an extension of OmniRPC to make use of computing resources managed by several grid-enabled job execution systems. In order to submit a RPC related job to grid-enabled job execution systems, the proposed system decouples the computation in a remote node from the Grid RPC mechanism and uses document-based communication rather than connection-based communication. By this system, we can exploit not only remote servers and clusters but also computing resources provided with grid-enabled job execution systems on different sites. We designed a general interfaces to construct OmniRPC system on each different grid-enabled job execution systems and adapted the proposed system to four grid-enabled job execution systems: XtremWeb, CyberGRIP, Condor and Open Source Grid Engine. We present implementations for the four grid-enabled job execution system and report a preliminary performance of those implementations by using two application. Our model can provide a framework of a parallel programming model by using remote procedure calls bridging between large scale computing resource pools.

1. はじめに

インターネットをはじめとする広域ネットワークの進歩により，広域ネットワーク上の計算機資源やデータの共有，並列分散コンピューティングを支援するグリッド技術が注目されている．

我々は，これまでグリッド環境上の複数の計算資源を遠隔手続き呼び出し（RPC: Remote Procedure Call）を用いて，簡単に並列分散プログラミングを行えるGrid RPCミドルウエアOmniRPC[1)]の開発を行ってきた．Grid RPCは，グリッド環境におけるプログラミングモデルの1つとして有望であり，特にパラメータサーチアプリケーションやタスク並列アプリケーションに対して有効である．

† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†† INRIA, France

近年，企業などで，製薬のための分子構造設計や電子回路設計のシミュレーションなど，莫大な量の計算を迅速に処理するニーズが非常に高くなっている．このために，グリッド環境上で大量の計算ジョブを実行する XtremWeb [2]，Condor [3]，CyberGRIP [4] などジョブ実行を行うグリッドジョブ実行システム（GJES）が開発されている．しかし，これらのシステムは，依然としてファイルを仲立ちとしたジョブバッチシステムであり，ジョブのメインの計算を行うプログラムと，そのジョブの結果のデータの解析を行うプログラムを，それぞれ作成しなくてはならない．さらに，これらのミドルウエアを透過的に利用するためのプログラミングモデルはまだ確立されていない．

そこで本稿では，OmniRPC において従来コネクションベースで行っていた RPC をドキュメントベースにすることによって，GJES のリモートジョブとして実行できるようにし，各拠点にある GJES で管理されている計算資源を統合し RPC スタイルで記述されたプログラムから活用可能にする．提案するシステムは，GJES をバックエンドのシステムとして活用し，その上に OmniRPC を実装することによって実現する．本稿では，複数 GJES の計算資源を統合・活用できる OmniRPC を構築するために必要な拡張や GJES を利用するためのインタフェースの設計について述べ，その設計に基づき，実際に 4 つの GJES に適応させる．これら実装について述べ，予備的な性能評価を行う．

本研究の貢献は，以下のとおりである．

- いろいろなジョブ実行システムで管理される計算資源を，RPC という計算モデルで書かれたプログラムから透過的に利用できるようにした．
- システム異常などによるワーカプログラム実行中断に対し耐故障性を保証した RPC システムの提供．
- 計算実行の RPC の粒度に応じて，クライアント側の設定ファイルの変更のみで，グリッド環境上でのマルチクラスタから GJES まで実行環境の計算資源を活用できる．
- RPC のスケジューリングをクライアント側で行うことができる．
- GJES で管理される計算リソースをリモートから利用可能にする．

2 章で OmniRPC の概要を述べ，3 章で複数 GJES 上で動作する Grid RPC システムの設計について述べる．4 章で 4 つの GJES への適応事例をのせ，5 章で簡単な性能評価を行う．終章でまとめと今後の課題を述べる．

2. OmniRPC システム

OmniRPC は，クラスタから広域ネットワークで構成されたグリッドに至る様々な計算機環境において，シームレスなマスター/ワーカ型の並列プログラミングを可能にする Grid RPC システムである．

OmniRPC で想定しているグリッド環境は，インターネット上で複数のクラスタが接続され，それらのクラスタを相互利用するような環境である．また OmniRPC は，現在のクラスタ環境に多く見られる，クラスタのマスタノードだけがグローバル IP を持ち，ワーカノードはプライベートアドレスを用いた構成も計算資源として利用可能である．このため OmniRPC では認証として，グリッド環境でよく用いられる Globus Toolkit [5] の GSI 認証のほかに，広く使われている SSH を利用可能になっている．

OmniRPC の API は，基本的に Ninf [6] の API を踏襲している．グリッド環境において典型的な並列アプリケーションであるパラメータ検索などのアプリケーションを効率的にサポートするために，OmniRPC では，リモート実行モジュール（REX）の起動時に実行される初期化手続きを定義することによって，REX を起動時に自動で初期化する機能を有している．我々は，この機能を自動初期化実行モジュール機能と呼ぶ．さらにワーカプログラム側の計算データの状態を保持する Persistency をサポートしている．この Persistency をサポートした API を利用することにより，効率的なプログラミングが可能となる．また，非同期呼び出しの API を用いることにより並列プログラミングを行うことができる．

OmniRPC はエージェントを使用して REX のプロセス管理を行う．また，エージェントを利用し，クライアントプログラムと複数の REX 間の通信を多重化して 1 つのコネクションで通信を行うことができる．

OmniRPC の動作イメージを**図 1** に示す．この図で

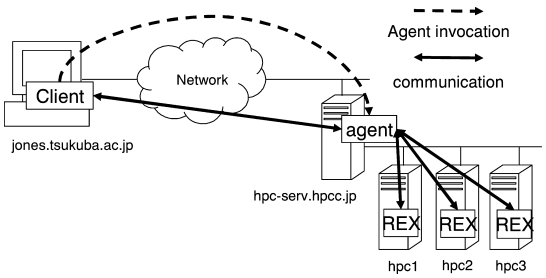

図 1 OmniRPC システムの概要
Fig. 1 An overview of OmniRPC system.

```
#include "OmniRpc.h"

main(int argc, char **argv) {
  /* declaration */
  double A[10][N][N], B[10][N][N], C[10][N][N];
  OmniRpcRequest req[10];
  OmniRpcInit(&argc, &argv);

  /* calls matrix multiplication function named "dmmul",
     C[i] = A[i]*B[i]  */
  for(i = 0; i < 10; i++)
    req[i] = OmniRpcCallAsync("dmmul",N, A[i], B[i], C[i]);
  /* waits the finishes of all RPCs */
  OmniRpcWaitAll(10, req);

  OmniRpcFinalize();
  ...
}
```

図 2 遠隔地で行列計算を行う OmniRPC のクライアントプログラムソース例
Fig. 2 A client program source code calculates matrix multiplication in remote nodes.

```
Module matrix;

Define dmmul(IN int n, IN double a[n][n],
          IN double b[n][n], OUT double c[n][n])
         "call function of matrix multiplication"
Calls "C" mmul(n,a,b,c);
```

図 3 行列計算を行う OmniRPC REX のプログラムソースの例
Fig. 3 An OmniRPC REX program source code.

は通信の多重化を行い，遠隔のクラスタのワーカノードに向けて RPC が行われている．

OmniRPC を用いて遠隔地で行列演算を行うプログラムのクライアントと REX のコード例をそれぞれ**図 2**，**図 3** に示す．

3. 複数 GJES の計算資源を統合・活用する Grid RPC システムの設計

3.1 対象とする GJES

本研究が対象とする GJES は，XtremWeb や Grid MP [7] に代表される遊休計算機を利用して独立したジョブを処理するシステムや，Condor，CyberGRIP，Grid Engine [8),9)] に代表される並列ジョブにも対応したバッチキューイングシステムである．これらの GJES で，GJES のサーバプログラムに対してクライアントプログラムからジョブ実行の依頼や計算資源の予約を行う．また GJES のサーバでジョブのキューイングが行われ，サーバプログラムがジョブを計算資源に割り当て，その計算ノード上でジョブを実行させる．ジョブの実行終了後，GJES が実行結果を計算ノードから回収する．

これらの GJES での基本的な単位は独立したジョブであり，ユーザにより投入されたジョブは GJES に参加している計算ノードに適宜割り当てられ実行される．ジョブを構成する入出力データはファイルを介してアクセスされる．

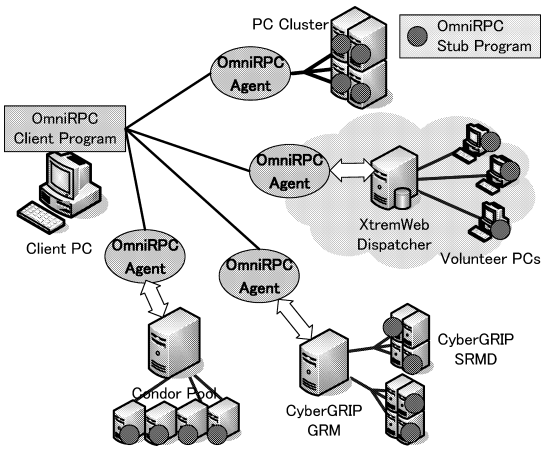

図 4 提案システムの概要
Fig. 4 An overview of the proposed system.

ジョブの遠隔計算機での実行のモデルには，CyberGRIP，Condor のようにサーバがジョブを遠隔の計算機に直接割り当て実行させるいわゆる Push モデルと，XtremWeb のように計算資源上で実行されているデーモンが，それぞれの GJES のサーバにジョブを取りにいく形態，いわゆる Pull モデルの 2 種類がある．

また，いくつかの GJES は，ハードウェア，ソフトウエアやネットワークの異常により中断されたジョブを再キューイングし，別の計算資源にジョブを再スケジューリングする機能を有する．

3.2 設 計 目 標

提案システムは，プログラムソースの変更なしに遠隔手続き呼び出しで記述されたグリッド向けアプリケーションの実行環境を，これまで利用可能だった複数クラスタ環境だけではなく，各拠点にある GJES で管理されている計算資源を統合し活用可できる環境にまで拡張できるようにする．提案するシステムの概略図を**図 4** に示す．

本システムでは以下の項目を目標とする．

- GJES に対する RPC による並列プログラミングモデルの提供
- 耐故障性を備えた Grid RPC システムの構築
- 各拠点の GJES が提供する計算資源の利用
- プログラムソースの変更なしで，複数クラスタから複数 GJES の計算資源へ実行環境を対応可能にする
- 新しい GJES に対応可能なインタフェースの提供
- REX ファイルの計算資源へのデプロイの自動化

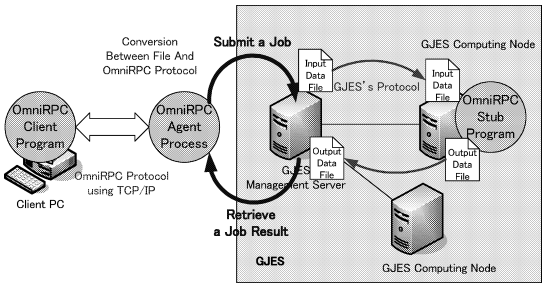

図 5 GJES に対する OmniRPC の実行モデル
Fig. 5 The OmniRPC execution model for GJES.

3.3 複数 GJES の計算資源を統合・活用する Grid RPC システムの設計

従来の OmniRPC は，クライアントプログラムが使用する計算機において REX を起動し，その REX にネットワーク接続を行う．そして，RPC 入力データを REX に転送し，計算を行う形態をとる．つまり OmniRPC は Push モデルのシステムである．また，計算中には，クライアントプログラムと REX 間はコネクションを維持し，このコネクションを使って計算の依頼，結果の転送を行っている．

GJES に対する Grid RPC の実行モデルを**図 5** に示す．RPC を複数 GJES 上で行うためには，コネクションにより行っていた計算の依頼と結果の転送を，ファイルを介して，すなわちドキュメントベースの通信により行うことが考えられる．あるリモート関数の遠隔呼び出しを行う場合は，その RPC の入力データをファイルに格納し，必要なリモート関数を含む REX とともにジョブとして GJES に投入する．GJES によって，そのジョブが計算ノードに割り当てられ，ジョブの実行が行われる．このジョブの計算結果，つまり RPC の出力データがファイルに出力され，そのデータを RPC の出力データとして変換しクライアントプログラムにデータ転送することにより，RPC が完了する．

OmniRPC システムでは，エージェントを用いることにより，遠隔の実行モジュールの起動，レジストリの管理，ワーカとの通信の中継を行わせることができる．本稿では，現状の OmniRPC のシステムを大幅に変更することなく複数の GJES に対応するために，このエージェントに GJES のサーバプログラムへのブリッジを行う機能を付加し実現することにした．

提案システムにおける耐故障性については，REX が実行される計算資源におけるシステム異常等によるジョブの中止について注目する．これは，REX の実行の負荷により，計算ノードがダウンしてしまう場合が我々の実験で多かったためである．また，の OmniRPC のアプリケーションの実行[10] では，クライアントプログラムとエージェントを動作させていた計算ノードのダウンはなかったため，クライアントプログラムと OmniRPC のエージェントについては，耐故障性を考慮しないことにした．

ワーカプロセス側の耐故障性を実現するにあたり，GJES の機能により以下の 2 つの対応が考えられる．

- GJES 自身が耐故障性機能を有する場合
  REX を実行する計算資源においてシステム異常等により中止されたジョブを，再スケューリングし，他の計算資源で処理を行う耐故障性を備えているため，エージェントから GJES のサーバに投入されたジョブの実行は必ず終了する．このため，エージェントは単にジョブが終了するまでポーリングすればよい．これにより，OmniRPC システム全体としては耐故障性を保証できる．
- GJES が耐故障性機能を持たない場合
  エージェントが，GJES サーバに投入したジョブの状態を定期的にチェックする．ジョブを実行する計算資源のシステム以上により異常終了した場合には，エージェントが GJES サーバからジョブの消去を行い，再度同じジョブを投入する．これをジョブが正常終了するまで続けることにより，OmniRPC システム全体として耐故障性を保証できる

ただし，提案するシステムでは，従来の OmniRPC システムで用いていたコネクションを用いるのに比べ，断続的な通信かつ，ドキュメントベースで処理を行うため，それぞれの GJES による通信性能・ジョブ起動のオーバヘッドによって性能が劣ることは避けられないと思われる．しかし，これはアプリケーションの通信と計算の比率に依存する問題であり，粗粒度の並列性のあるアプリケーションであれば十分な性能が得られる．このようなアプリケーションにとって，従来のグリッド環境に加え，GJES での大量の計算資源が利用できることは十分な意義があると考える．

また提案するシステムでは，システムの機構上 OmniRPC で使用できた Persistency は基本的にサポートできない．しかし，これまで提案してきた OmniRPC 自動初期化実行モジュール機能に関しては，提案システムにおいてプログラミングセマンティックスのみサポートする．このセマンティックスを実現するため，OmniRPC のエージェントは REX の初期化用のデータをローカルに保存，かつ RPC 実行時にこの RPC 用の入力パラメータデータと保存されている

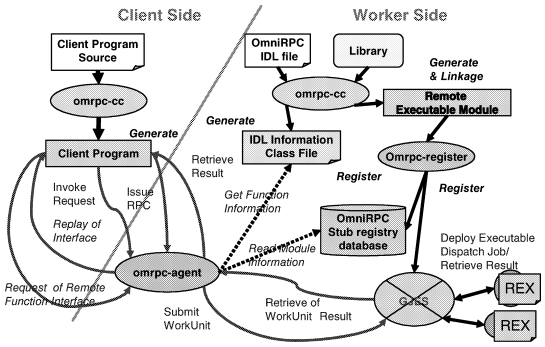

図 6 プログラム作成のフロー
Fig. 6 A flow diagram of program developmenet by the proposed system.

```
<?xml version="1.0" ?>
<OmniRpcConfig>
 <Host name="dennis.hpcc.jp" user="ynaka" />
  <Agent invoker="ssh" mxio="on" path="/omrpc"/>
  <JobScheduler type="xtremweb"
     maxjobs="32" config="/etc/xwrc"/>
 </Host>
 <Host name="alice.hpcc.jp" user="msato" />
  <Agent invoker="ssh" mxio="on" path="/omrpc"/>
  <JobScheduler type="cybergrip" maxjobs="32" />
 </Host>
</OmniRpcConfig>
```

図 7 複数の GJES を利用する際の設定ファイル
Fig. 7 An exmaple configuration file for multiple GJESs.

初期化用のデータとともにジョブとして GJES に投入する．ただし，提案システムは 1RPC ごとに GJES の計算資源で動作している REX プロセスが終了するため，提案システムにおいてこの機能を利用することによる初期化のためのオーバヘッドの削減はできない．しかし，初期化のためのデータをローカルにキャッシュすることで，効率化は可能になる．

以上のことを考慮し，提案するシステムでは OmniRPC に以下の拡張を行う．

- OmniRPC のエージェントが各 GJES のサーバと OmniRPC のクライアントプログラム間のプロトコル変換を行う機能
- OmniRPC の REX でデータ入出力をファイルから行う機能
- リモート関数のインタフェース情報の管理
- 新しい GJES に容易に対応可能なインタフェースの提供

図 6 に提案するシステム上での，プログラミング作成時のフローを示す．ユーザは基本的に OmniRPC と同じようにクライアントプログラムと IDL（Interface Description Language）で REX プログラムを記述する．そして OmniRPC が提供するツールを用いてコンパイルや REX の登録などを行う．

3.4 複数 GJES の計算資源の統合

提案システムにより，各拠点にある GJES を統合し，それらの GJES によって管理されている計算資源を活用できるようにする．ユーザが OmniRPC の計算資源として GJES を利用するには，各拠点で管理者が用意した GJES の操作やジョブ実行要求を行うインタフェースの実装と，設定ファイルの変更のみ行う．

図 7 に提案システムにおいて，複数のジョブ実行システムを同時に利用するときの設定ファイル例を示す．この例では，**dennis.hpcc.jp** にある **XtremWeb** と **alice.hpcc.jp** にある **CyberGrip** をバックエンドシステムとして利用する．**Host** 要素の中にある **JobScheduler** の **type** において使用する GJES を指定する．また，**maxjobs** で GJES に同時にジョブを投入できる最大数を設定する．提案システムでは，この GJES のジョブ投入スロットリングによって，クライアントのプログラムがスロットリングの制限値を超えて RPC を行う際には，次の RPC 実行がブロックされるようになる．これによって，一度に大量の RPC の計算の処理をするジョブが GJES に投入されるのを防ぐことができる．

3.5 複数 GJES の統合可能なインタフェースの設計

前節で述べた OmniRPC の実行モデルにおいては，エージェントはクライアントプログラムと GJES のサーバ間でのプロトコル変換を行う．そこで，GJES 操作とジョブの実行マネージメントを行う汎用的な `BatchSystem` クラスと，ジョブ実行要求と計算資源要求，ジョブの入出力データを管理する `WorkUnit` クラスの 2 つを作成し，エージェントが利用することによって複数の GJES に対応できるようにする．

`BatchSystem` クラスは，3 つのキュー（投入する `WorkUnit` の管理，実行中の `WorkUnit` の管理，結果の `WorkUnit` の管理）と，それぞれ GJES に `WorkUnit` を投入するスレッド，結果を取得するスレッドを持つ．基本的に，ジョブを投入する場合には，このクラスに対して行い，結果の取得もこのクラスを通して行うようにする．ただし，各 GJES 依存のジョブの投入・状態取得・結果の取得・削除の操作については GJES ごとにインタフェースを実装したクラスを提供することにする．この `BatchSystem` から利用される操作インタフェースを `BatchSystemInterface` と呼ぶ．

`WorkUnit` クラスは，使用するアプリケーション名，RPC の入力・出力データを保持するファイル名，REX

を実行させるに必要な転送するファイル名などを保持する．ジョブ実行要求や計算使用要求とデータのファイルヒエラルキはシステム固有であり，これらの要求を生成する操作と，RPC のデータを書き込むストリームを取得する操作，RPC の結果のデータへの入力ストリームを取得する操作が必要である．

各 GJES に対応させるには，この `WorkUnit` クラスを継承し，システム固有のジョブ要求や計算使用要求とデータのファイルヒエラルキの要素を保持させる．

これら `BatchSystemInterface` と `WorkUnit` の実装を行うことにより，GJES 上で OmniRPC を動作させることができる．

3.5.1 GJES の操作インタフェース

`BatchSystemInterface` クラスは各 GJES ごとに異なる操作を抽象化したものである．実装しなくてはならない必要な操作は，システムを使うためのセットアップと終了処理，ジョブ要求の投入・状態取得・結果の取得・削除と REX に対応する `WorkUnit` の作成である．

具体的には以下にあげる 6 つのメソッドをシステムごとに実装する．

- *Initialize()*
  GJES を使用するためにアカウントやセッションを設定する．
- *Finalize()*
  GJES の使用を終了する．
- *createWorkUnit(REX rex)*
  指定した REX をそれぞれの GJES に投入するために，それぞれ GJES 固有のジョブを管理する `WorkUnit` を継承したクラスのインスタンスを生成する．
- *unregister(WorkUnit u)*
  GJES のサーバに登録された WorkUnit を削除する．
- *retrieveCompleteWorkUnit(WorkUnit u)*
  終了したジョブの出力ファイルやログファイルを GJES サーバより取得する．
- *getWorkUnitStatus(WorkUnit u)*
  指定された `WorkUnit` に対応するジョブの状態を取得する．

3.5.2 GJES 上のジョブの記述

`WorkUnit` クラスは，GJES のジョブを抽象化したものである．ユーザは，GJES 固有のジョブ要求や計算資源使用要求とデータのファイルヒエラルキの要素を管理するインタフェースを実装する．

そのクラスの中で，主に以下の 3 つのインタフェースを実装する．

- *prepareToSubmit()*
  GJES 上にジョブを投入する前に，抽象化された `WorkUnit` を GJES 独自のジョブ実行要求フォーマットへ変更する．
- *getOutputStream()*
  ジョブの実行時に必要な OmniRPC クライアントから送られてくるデータを書き込む，入力データファイルへの OutputStream を取得する．
- *getInputStream()*
  ジョブの実行で生成された，OmniRPC クライアントへ送る出力データファイルからの InputStream を取得する．

4. 複数 GJES への適用

以下では，適応事例として用いた GJES と，それらに対する OmniRPC の実装について説明する．

GJES として XtremWeb，CyberGrip，Condor，Open Source Grid Engine 上に提案するシステムを適応させる．それぞれ OmniRPC を適応させたシステムを OmniRPC/XW，OmniRPC/CG，OmniRPC/C，OmniRPC/GE と呼ぶ．また，各ミドルウエアの差異を**表 1** に示す．

XtremWeb は，インターネットに接続された計算機やイントラネット内部にある計算資源プールを活用し，大規模分散処理を目的とした中央集権的な管理・ジョブスケジューリング・結果収集を行うグローバルコンピューティングミドルウエアである[11]．

CyberGRIP は富士通研究所が開発した，既存のバッチシステムを仮想的に 1 つのバッチシステムとして利

表 1 各 GJES における差異
Table 1 Specification difference between GJESs.

項目	XtremWeb	CyberGRIP	Condor	OSGE
アカウンティング	あり	なし	なし	なし
利用のためのセッション管理	あり	なし	なし	あり
ジョブ要求の指定	XtremWeb API	CyberGRIP の GRM スクリプト	スクリプト	DRMAA API
実行プログラムの指定	登録されたアプリケーション名	実行バイナリのパス	実行バイナリのパス	実行バイナリのパス
サーバの使用要求	特になし	OS，アーキテクチャの指定	Universe の指定	特になし
ジョブの ID の生成	クライアント	ジョブ投入時にサーバ	ジョブ投入時にサーバ	ジョブ投入時にサーバ
ジョブの投入・取得時のデータ形式	Zip 形式のアーカイブ	特になし	特になし	特になし
サーバからの結果の回収	クライアントが明示的に行う	システムが転送	システムが転送	システムが転送
ジョブの実行形態	ワーカがサーバに問合せ	サーバからワーカに依頼	サーバからワーカに依頼	サーバからワーカに依頼
サーバへのアクセス方法	Java API	C library	Command	DRMAAlibrary (Java)

用できるようなフレームワークを提供し，大量の計算ジョブを処理する GJES である[4]．

Condor は，Wisconsin-Madison 大学で開発されている，計算機の空き時間を利用しハイスループットコンピューティングを目的とするジョブスケジューリングシステムである．

Open Source Grid Engine（OSGE）は，ジョブのキューイングやスケジューリングを行い，投入されたジョブやユーザの要求に対応して，利用可能なリソースを動的に提供するシステムである．また，Global Grid Forum の DRMAA（Distributed Resource Management Application API）WG[12] で仕様が策定された汎用的に DRM システムを利用するためのライブラリを提供している．

4.1 実　　装

4.1.1 OmniRPC/XW

GJES のサーバの操作において，XtremWeb では独自のアカウントでのセッション管理が必要である．そのため *Initialize()*，*Finalize()* によってユーザ認証およびセッション操作を行うようにした．また，アプリケーションやデータファイルは Coordinator にホスティングされている．このため，*retrieveCompleteWorkUnit()* において，明示的に Coordinator から実行結果を含むデータファイルのダウンロードを行うようにした．さらに，Coordinator でホスティングされている REX の識別子は，REX のバイナリ名と別々で管理されているため，*createWorkUnit()* で対応をとるようにした．

XtremWeb でのジョブの操作については，XtremWeb のクライアントプログラム側でジョブの識別子を作成し，その識別子を介して操作を行う．そこで，*prepareToSubmit()* で，ジョブ投入に必要な識別子の生成と実行プログラムに与えるコマンド引数を指定する．さらに，XtremWeb では入出力データファイルのファイルヒエラルキを Zip アーカイブ形式で扱う．そのため，*getInputStream()* や *getOutputStream()* も Zip ファイルを介した操作で行うようにした．

4.1.2 OmniRPC/CG の実装

ミドルウエアの操作において，CyberGRIP はファイルの転送をシステムが自動的に行う．このため OmniRPC/CG では，ジョブ投入とジョブの状態取得だけを実装した．しかし，CyberGRIP では，ジョブ要求の操作において，ジョブ実行要求と計算機の使用要求の両方が必要となる．さらに，アプリケーションのコマンドオプションはジョブ要求に記述できないため，シェルスクリプトを作成しその中でアプリケーションとコマンド引数を記述することにより実現した．また，シェルスクリプトをジョブ要求での実行ファイルとして指定する．入出力ファイルへのストリーム操作に関しては，通常のファイル操作で行うことが可能であるため特別な操作は行っていない．

4.1.3 OmniRPC/C の実装

ミドルウエアの操作において，シェルコマンドを利用し Condor のサーバの操作を行う．Condor も同じくジョブ入出力ファイルの転送をシステムが自動的に行うため，OmniRPC/CG ではジョブ投入とジョブの状態取得操作だけを実装した．ジョブ実行要求に関しては，独自の Condor スクリプトをファイルに生成する操作を実装した．入出力ファイルへのストリーム操作に関しては，通常のファイル操作で行うことが可能であるため特別な操作は行っていない．

4.1.4 OmniRPC/GE の実装

OSGE が提供している Java 版の DRMAA ライブラリを使用し OSGE の操作を行う．DRMAA では，セッション管理が必要であるため，Initialize()/Finalize() でセッションの開始・終了を行う．ジョブ実行要求に関しては DRMAA のライブラリで指定する．通常のファイルが使用できるため，ファイルストリームに関わる特別な操作は行っていない．

5. 予 備 評 価

実装したシステムに対してモデルプログラムと実アプリケーションを用いて性能評価を行う．

5.1 実 験 環 境

筑波大学にある 2 つのクラスタを使用し実験を行った．使用した計算機の概要を**表 2** に示す．この実験では各クラスタを占有して実験を行っている．まず Dennis クラスタのマスタノードと各クラスタ間のネットワークの性能を明らかにするために *iperf*，*ping* を使用して測定を行った．その結果を**表 3** に示す．GCC version 3.4.4，Sun Java 2 Platform Standard Edition version 1.4.2_10 を使用してプログラムをコンパイルしている．また GJES はそれぞれ，XtremWeb version 1.5.0（バックエンド SQL システムとして MySQL Version 4.1.12 を利用），CyberGRIP version 2.2（ロー

表 2 グリッドテストベットの計算機環境
Table 2 Experimental platform in the grid test-bed.

Cluster Name	Machine	Network	#nodes
Dennis	Dual Xeon 2.4 GHz, 1 GB	1 GbE	10
	Dual Xeon 3.06 GHz, 1 GB	1 GbE	6
Alice	Dual Athlon 1800+, 1 GB	100 MbE	9

表 3 Dennis のマスタノードと各クラスタ間のネットワーク性能
Table 3 Network performance between dennis cluster's master node and each cluster's node.

Cluster	Round-Trip Time (ms)	Throughput (Mbps)
Dennis	0.23	879.31
Alice	0.18	94.12

表 4 各 GJES のスケジューリング設定
Table 4 Scheduling configuration of each GJES.

Items	XtremWeb	CyberGRIP	Condor	OSGE
Polling Interval (s)	1	10	5	15

カルバッチシステムとして JTX を使用)，Condor version 7.10.7，Open Source Engine Version 6.0u6 を使用した．

提案システムは以下の 2 つのスケジューリングのレイヤが存在する．

- OmniRPC クライアントプログラムが RPC を各 GJES へ割り当てるためのスケジューリング
  OmniRPC のクライアントプログラムは各 GJES への RPC の割当てをラウンドロビンで行う．また前述のとおり，OmniRPC エージェントが各 GJES に同時に割り当てる RPC の上限数を，OmniRPC の設定ファイル中の `maxjobs` で指定する．もし，`maxjobs` より多い数の RPC を GJES に同時に割り当てようとすると，RPC は GJES に割り当てられた RPC の実行が終了するまでブロックされる．
- GJES における OmniRPC エージェントから投入されたジョブを各計算資源へ割り当てるためのスケジューリング

本実験中の前者のスケジューリングの設定に関して，RPC の各 GJES への割当てはラウンドロビンを利用する．また，GJES の計算資源ノードがジョブの実行終了後連続してジョブ実行を行えるように `MaxJob` の値は GJES が管理する計算ノード数の 2 倍にした．本実験での後者のスケジューリングの設定に関して，各 GJES のデフォルトの設定を利用することとした．ただし，各 GJES の 1 ノードに割り当てるジョブ数は 1 とする．ここで各 GJES のスケジューリングポリシを変更しても大きく性能が変化することはないと考える．**表 4** に，使用した各 GJES の設定を示す．ただし，XtremWeb のワーカデーモンの再接続時間は 15 秒，Coordinator の MySQL への接続は 1 秒とした．また OmniRPC エージェントが各 GJES へ操作をする間隔は 1.0 秒にしている．

性能評価は，同じ評価項目を 3 回連続で行い，その中で最良の結果をのせている．

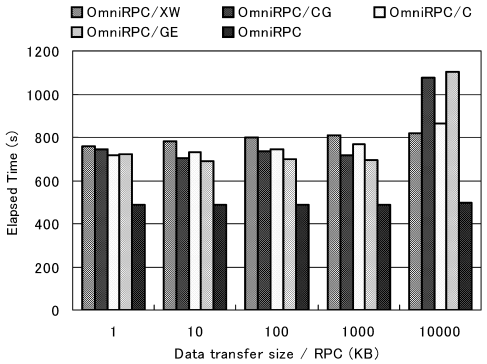

図 8 1RPC の実行時間が 60 秒のときの実行時間
Fig. 8 Elapsed time in the case of 60 seconds per RPC.

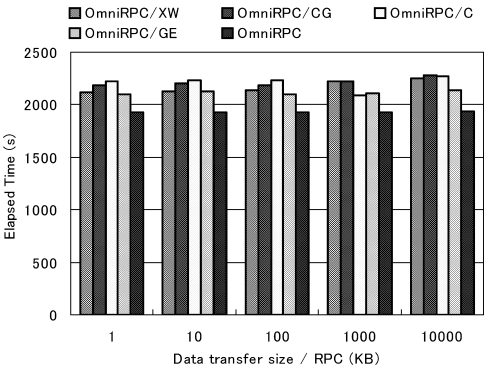

図 9 1RPC の実行時間が 240 秒のときの実行時間
Fig. 9 Elapsed time in the case of 240 seconds per RPC.

5.2 RPC アプリケーションをシミュレートするモデルプログラムによる性能評価

実装した OmniRPC/{XW,CG,C,GE} 上で，RPC アプリケーションをシミュレートするモデルプログラムを用いて，遠隔計算機における 1RPC の実行時間とデータ転送量を変化させ，実行時間と逐次対性能比を調べる．

この性能評価では dennis クラスタを使用し，各 GJES のサーバ，クライアントプログラムと OmniRPC エージェントは dennis クラスタのマスタノードで実行させる．

1RPC の遠隔計算資源での実行時間を短い 60 秒と少し長い 240 秒として，データ転送量を 1 KB から 10 MB まで変化させて，各 128 回 RPC を実行する実験を行った．1RPC の実行時間が 60 秒・240 秒のときの，モデルプログラムの実行時間をそれぞれ**図 8**，**図 9** に示す．ただし，比較のために OmniRPC 単体を用いた結果ものせる．逐次性能比は 1CPU で計算したときの実行時間を基準に算出している．

1RPC の実行時間が 60 秒のときには，OmniRPC/

XW が 11.1 倍，OmniRPC/CG が 11.0 倍，OmniRPC/GE が 11.3 倍，OmniRPC/C が 10.7 倍の対逐性能比が得られた．240 秒のときには，OmniRPC/XW が 14.5 倍，OmniRPC/CG が 14.1 倍，OmniRPC/GE が 14.7 倍，OmniRPC/C が 14.7 倍の対逐次性能比が得られた．しかし，提案システムは，1RPC の実行時間が 60 秒の場合は平均 3 割，240 秒の場合は 1 割ほど OmniRPC 単体を利用するよりも性能が低い．ここで，クライアントプログラムから 1RPC を実行するのに必要なオーバヘッドのほとんどは GJES におけるジョブスケジューリングである．エージェントが行う RPC データとファイルとの相互変換などに必要な処理が占める割合は，5%以下である．この GJES のジョブスケジューリングにかかるオーバヘッドが性能を下げる主な要因となっている．このように，直接計算資源を利用する場合に比べて，本システムでは GJES を介して計算を実行することに起因するオーバヘッドが存在するが，提案システムは妥当な性能向上が得られていると考えられる．

また，1RPC の実行時間が 240 秒の場合は，RPC のデータ転送量が 1 KB から 10 MB まで同じような性能が得られているのに対し，60 秒の場合には 10 MB の場合には性能が極端に悪化している．1RPC の実行時間が短いときに提案システムでは，GJES 側においてジョブ終了のサーバへの登録やデータ転送が集中してしまうことが多く，特に転送量が多い場合にはその傾向が多く見られるようになる．しかし，提案システムにおいて，1RPC で 10 MB のデータ転送と約 1 分の処理を行うような粒度のアプリケーションは想定していないため，問題ないと考えられる．

5.3 実アプリケーションによる性能評価

複数の GJES を利用して計算を行う事例を用いて性能評価を行う．使用したアプリケーションは，最尤法で分子系統樹推定を行うプログラム集 Phylogenetic Analysis by Maximum Likelihood（PAML）[13] の中から *codeml* を並列に実行可能にさせたものを用いた．*codeml* プログラム中で，OmniRPC を用い処理する全遺伝子データセットを，分割し決まった本数の遺伝子データを複数の計算資源で並列処理するように変更した．

計算機環境として，Dennis では OSGE，Alice では Condor を OmniRPC のバックエンドシステムとして使用する．1,000 本の遺伝子データを 5 本ずつ分割し計 200 回 RPC 行うことで計算を行う．1RPC に必要なデータ転送量は，入力パラメータ用に約 100 KB，出力パラメータ用に 30 KB である．リモートでの 1RPC 計算時間は Dennis のノードで約 5 分，Alice のノードで約 10 分である．

表 5 OmniRPC 版 PAML codeml の実行時間
Table 5 Execution time of PAML OmniRPC version in each setting.

使用したクラスタ（ノード数）	実行時間 (s)
Dennis (1, 2.4 GHz node) original program	71844.3
Alice (9)	14914.8
Dennis (16)	3796.7
Dennis (16) + Alice (9)	3473.4

PAML *codeml* の実行結果を**表 5** に示す．性能比較のためにオリジナルの *codeml* プログラムを Dennis 2.4 GHz のノードで計算したときの時間をのせる．

提案システムを用いて 2 つのクラスタを利用したときに，オリジナルの *codeml* プログラムを Dennis 2.4 GHz のノードで計算した場合よりも最大で 20.68 倍高速に尤度計算を行うことができた．しかし，2 つのクラスタを利用した場合，Dennis クラスタ単体で利用する場合より性能向上比は小さい．これは，Dennis クラスタに比べて Alice クラスタは性能が低く，1RPC あたりの実行時間は Alice の計算ノードの方が長くなることが原因となり，ジョブの計算資源への割当てのロードインバランスが起こっている．そのため，遅い計算ノードのジョブ実行の終了を待ち合わせる時間が長くなり，計算資源を有効に利用できていない．この実験では，7 分以上もただ 1 つの計算機のみでジョブ実行が行われて，ほかの計算機はそのジョブ実行終了の待ちをしていたことがあった．これが，性能向上を妨げる原因になっている．

提案システムの性能は OmniRPC 単体で利用するよりも性能は低くなる．しかし，リモート実行時間での増加にともないその差は少なくなる．我々が行ってきたグリッド上での実験において[10]，1RPC の実行時間は 10 分から 1 時間近くになることも多く，このような場合に提案システムを利用した際の性能は OmniRPC 単体を利用したときと同じような性能が得られることが予想される．さらに，提案システムではワーカプログラムの耐故障性の機能も利用できる．このため，提案システムにおける欠点である性能低下にくらべて，耐故障性能や広域ネットワーク環境上に存在する異なる拠点の計算機を利用できる利点を考えれば，提案システムは十分意義があると考える．

5.4 遠隔計算機におけるワーカプログラムの耐故障性

提案システムの遠隔計算機上におけるワーカプログラムの耐故障性を検証するために，以下の 2 つの方法を用いる．

- リモートプロセスを Kill する．
- リモート計算機での GJES の各デーモンを異常終了させる．

以上の 2 つを，各提案システムに行ったところ，OmniRPC エージェントプロセスがジョブの異常終了を検知し，同じ計算内容を再投入したことを確認できた．結果として，提案システムでは，計算を停止することなくプログラムを正常に最後まで実行可能なことを確認した．

6. 関連研究

Djilali は，XtremWeb 上で遠隔手続き呼び出しを用いることでバッチシステム上にプログラミングモデルを構築した[14]．しかし，クライアント側は RPC スタイルでプログラミングが可能であるが，リモート側は，ファイルからデータを入力し計算しその結果をファイルに出力するプログラムを別に作成する必要がある．このため，ユーザには統一的なプログラミングモデルは提供されていない．

ジョブをバッチシステムレベルで相互にやりとりし，それぞれの GJES が管理している計算機資源を活用するという研究はいくつかある．Condor-G [15] は Globus によるドメインの資源管理プロトコルと，Condor によるドメイン内の管理を行うことによって複数のドメインにある資源を仮想的な単一ドメインとして扱うことが可能になっている．

中田らは，Condor の上に耐故障性を重視した RPC システム Ninf-C を実装した[16]．Ninf-C では，チェックポイントとマイグレーション機能をサポートしているため計算途中のジョブの停止，他の計算機へのジョブの移動を行い対故障性を保証する．我々の提案するシステムでは Condor 以外のシステムも利用できるが，Ninf-C では Condor 以外のミドルウエア上では動作しない．また，入力データセットによって RPC の粒度が異なることがある，そのような場合に対し我々が提案するシステムでは，細粒度のときにはマルチクラスタ環境を利用し，荒粒度のときには GJES 環境で動作させるといったことが，設定ファイルの変更のみで行うことができる．しかし，Ninf-C の設計にはこのような場合が考慮されていない．さらに，Ninf-C では，Condor-G によって，Globus によって管理されたグリッドの計算資源にアクセスできるが直接はアクセスできないのに対し，我々のシステムではそれが可能である．

7. おわりに

遠隔手続き呼び出しのプログラミングモデルを用いて複数 GJES 上の計算資源を統合・利用可能にするシステムを設計し実装した．それぞれの GJES において OmniRPC から利用するために必要な汎用なインタフェースについて検討，設計を行い，4 つの GJES に提案するシステムを適応させた．提案システムにより，複数の GJES が管理する計算資源上で，遠隔手続き呼び出しによるプログラミングモデルを用いて統一的に利用できるフレームワークを提供した．

今後の課題としては，各拠点の計算資源の利用状況に基づきそれぞれの GJES の計算資源に RPC を割り当てる動的な RPC スケジューリングの機能について検討を行う．

謝辞 富士通研究所の CyberGrip 開発チームには，ソフトウエアを提供していただき，また本システムの実装についてのご助言をいただきました．Université de Paris Sud，LRI の Samir Djilali 氏と LAL の Oleg Lodygensky 氏，France INRIA の Gilles Fedak 氏には OmniRPC/XW 実装についてのご助言をいただきました．本研究の一部は，文部科学省科学研究費補助金基盤研究（A）課題番号 172002「大容量分散コンピューティングのための大規模スケーラブル P2P グリッド基盤の研究」，特別研究員奨励費課題番号 177324，および，日仏共同研究プログラム（SAKURA），日仏共同研究 FJ Grid による．

参考文献

1) 佐藤三久，朴　泰祐，高橋大介：OmniRPC：グリッド環境での並列プログラミングのための Grid RPC システム，情報処理学会論文誌：コンピューティングシステム，Vol.44, No.SIG11 (ACS 3), pp.34–45 (2003).
2) Germain, C., Néri, V., Fedak, G. and Cappello, F.: XtremWeb: Building an Experimental Platform for Global Computing, *GRID '00: Proc.1st IEEE/ACM International Workshop on Grid Computing*, pp.91–101 (2000).
3) Litzkow, M.J., Livny, M. and Mutka, M.W.: Condor — A Hunter of Idle Workstations., *ICDCS*, pp.104–111 (1988).
4) Miyazawa, K., Kadooka, Y., Yamashita, T., Suzuki, T. and Tago, Y.: Development of Grid Middleware CyberGRIP and Its Applications, *1st IEEE International Conference on e-Science and Grid Computing*, *PSE Workshop* (2005).
5) Foster, I. and Kesselman, C.: Globus: A Meta-

computing Infrastructure Toolkit, *The International Journal of Supercomputer Applications and High Performance Computing*, Vol.11, No.2, pp.115–128 (1997).
6) Ninf Project. http://ninf.apgrid.org/
7) United Devices: Grid MP. http://www.ud.com
8) Open Soruce Grid Engine. http://gridengine.sunsource.net/
9) Gentzsch, W.: Sun Grid Engine: Towards Creating a Compute Power Grid., *CCGRID*, pp.35–39 (2001).
10) Nakajima, Y., Sato, M., Goto, H., Boku, T. and Takahashi, D.: Implementation and performance evaluation of CONFLEX-G: grid-enabled molecular conformational space search program with OmniRPC, *ICS '04: Proc. 18th annual international conference on Supercomputing*, pp.154–163 (2004).
11) Fedak, G., Germain, C., Neri, V. and Cappello, F.: XtremWeb: A Generic Global Computing System, *CCGRID '01: Proc.1st International Symposium on Cluster Computing and the Grid*, p.582 (2001).
12) Distributed Resource Management Application API Working Group. http://www.drmaa.org/
13) Yang, Z.: PAML: A program package for phylogenetic analysis by maximum likelihood, *Computer Applications in BioScience*, Vol.13, pp.555–556 (1997).
14) Djilali, S.: P2P-RPC: Programming Scientific Applications on Peer-to-Peer Systems with Remote Procedure Call, *CCGRID '03: Proc. 3rd International Symposium on Cluster Computing and the Grid*, p.406 (2003).
15) Frey, J., Tannenbaum, T., Livny, M., Foster, I. and Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids, *Cluster Computing*, Vol.5, No.3, pp.237–246 (2002).
16) 中田秀基，田中良夫，松岡 聡，関口智嗣：耐故障性を重視した RPC システム Ninf-C の設計と実装，先進的計算基盤システムシンポジウム（SACSIS2004）(2004).

(平成 17 年 10 月 4 日受付)
(平成 18 年 1 月 29 日採録)

中島 佳宏（学生会員）
昭和 55 年生．平成 15 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学中．グリッドコンピューティング等に関する研究に従事．

佐藤 三久（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 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学システム情報工学研究科教授．同大学計算科学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．IEEE，日本応用数理学会各会員．

相田 祥昭
昭和 57 年生．平成 17 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学中．グリッドコンピューティングに興味を持つ．

高橋 大介（正会員）
昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 11 年同大学情報基盤センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．平成 16 年筑波大学大学院システム情報工学研究科講師．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度，平成 15 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．

朴　泰祐（正会員）

昭和 36 年生．昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，平成 16 年同大学大学院システム情報工学系助教授，平成 17 年同教授，現在に至る．超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，クラスタコンピューティング，グリッドに関する研究に従事．平成 14 年度および平成 15 年度情報処理学会論文賞受賞．日本応用数理学会，IEEECS 各会員．

Franck Cappello

Franck Cappello holds a Research Director position at INRIA. He leads the Grand-Large project at INRIA and the Cluster and Grid group at LRI. He has initiated the XtremWeb (Desktop Grid) and MPICH-V (Fault tolerant MPI) projects. He is currently the director of the Grid5000 project, designing, building and running a large scale Grid experimental platform. He has authored more than 50 papers in the domains of High Performance Programming, Desktop Grids, Grids and Fault tolerant MPI. He is editorial board member of the "international Journal on GRID computing" and steering committee member of IEEE HPDC and IEEE/ACM CCGRID. He is the general chair of HPDC 2006.