2005－HPC－104（2）
2005／10／7

グリッドRPCシステムOmniRPCにおける初期データの分散管理による効率化

相田祥昭[†]　中島佳宏[†]　佐藤三久[†]
櫻井鉄也[†]　高橋大介[†]　朴泰祐[†]

OmniRPC は、グリッド環境向け並列プログラミングのための RPC システムである。グリッドを利用したアプリケーションでは，マスタとワーカで比較的大きいデータを共通に持つ必要がある場合があるが、RPC でそのようなデータ転送を行う場合、マスタからワーカごとに直接転送することになる。このため，巨大な初期データを必要とするアプリケーションでは性能向上を妨げる原因の一つとなっていた．そこで，OmniRPC においてワーカの起動とデータの転送を分離した実行モデルを提案する．ワーカ起動は OmniRPC の機構を用いて，共通に用いられるデータのための転送を別のレイヤを用いて行うようにする．これにより，データの転送に様々な方法を用いることが可能となる．このプロトタイプとして，データ転送の最適化を可能とする OmniStorage を設計，実装した．OmniStorage は，木構造にサーバを接続し，さらにそれぞれのサーバでデータをキャッシュする機能を有することによりデータ転送の効率化を図る．グリッド環境上で，RPC アプリケーションを模擬するベンチマーク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った．OmniStorage を利用することにより，OmniRPC のみを利用するよりも性能向上が得られることが分かった．

Performance Improvement by Initial Data Management on Grid RPC System OmniRPC

Yoshiaki Aida,[†] Yoshihiro Nakajima,[†] Mitsuhisa Sato,[†]
Tetsuya Sakurai,[†] Daisuke Takahashi[†] and Taisuke Boku [†]

OmniRPC is a Grid RPC system for parallel programming in a gird environmnet. In many applications such as parameter search, a large amount of common data is often needed in both master and workers. When using RPC to transfer data, the data is transferred for each worker, resulting in poor performance. To improve performance for this case, we propose a model to separate the data transfer by a data management layer from the RPC invocation. We have designed and implemented a prototype data transfer layer named OmniStorage. It enables efficient data transmission by connecting intermediate relay servers in tree network topology, and cache the data on the servers. We have evaluated the performance of our system by using a synthetic program and a real application. The results shows that OmniStorage improves OmniRPC application to achieve better performance than using only OmniRPC system.

1. はじめに

近年のインターネットの普及に後押しされた広域ネットワークインフラの発達に伴って，広域ネットワーク上での計算機資源の統合やデータの共有を可能にするグリッド技術が注目されている．グリッド上の広域に分散した資源を活用するプログラミングモデルとして，グリッド環境上における複数の計算資源を遠隔手続き呼び出し (Remote Procedure Call) によって利用する Grid RPC が，グリッド環境におけるプログラミングモデルの一つとして有望視されている．Grid RPC は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なプログラミングモデルを与える．我々は，Grid RPC のためのミドルウェアの実装の一つとして OmniRPC[1)2)] の研究開発を進めている．

Grid RPC を用いた典型的なプログラムは，遠隔呼び出しを行うマスタと，呼び出される手続をリモートの計算ノードで実行する複数のワーカで構成される．グリッド上のアプリケーションではワーカ間で比較的大きいデータを共通に持つ必要がある場合が多い．例えば，パラメータサーチではそれぞれのワーカは同じ

† 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

データを持ち，パラメータを変えて並列に同じ遠隔呼び出しを実行する．persitency を持たない遠隔呼び出しのみの場合は，呼び出しごとに共通のデータを送らなくてはならず，データのサイズが大きい場合には大きなオーバーヘッドとなる．

OmniRPC では，このようなプログラムに対し初期化機能を提供しており，この機能を用いることにより，ワーカ起動時に共通のデータの転送を行い，パラメータごとの遠隔呼び出しでは，同じデータを使うことができる．しかし，この初期化データの転送は通常の遠隔呼び出しの一部として行われており，マスタからそれぞれのワーカに転送されている．ワーカの数が多い場合，あるいはマスタといくつかのワーカの間のバンド幅が低い場合，性能を制限することになる．通常，遠隔呼び出しの引数として渡すデータはそれほど大きなものではないが，数十 MB に及ぶデータ転送を必要とするアプリケーションは存在しており，このようなアプリケーションでは転送にかかる時間が長くなることによって実行効率の低下を招いている．例えば，我々が開発した並列固有値計算プログラム[3] では，このアプリケーションは 1 個のジョブの実行時間が 1 分程度であるのに対して入力データのサイズが約 50MB と大きく，広域ネットワークでのデータの転送時間が性能向上のボトルネックとなっている．

そこで我々は，データの転送を Grid RPC の機構から分離し，データの転送を行うレイヤを用いて，共通に用いる必要があるデータの転送を効率化するモデルを提案する．マスタは，ワーカを起動する前に，共通に用いるデータを登録し，ワーカはデータ配送レイヤを用いて，そのデータを取得する．遠隔呼び出しの引数として転送する場合には個々のワーカに転送しなければならないのに比べて，データの転送方法を工夫することにより，効率化することができるようになる．

このためのプロトタイプとしてデータ配送レイヤ OmniStorage を設計，実装した．OmniStorage では，マスタとワーカの間のネットワーク上の適当な位置にデータを中継するサーバーを配置し，マスタで登録したデータをそのサーバー上にてキャッシュすることで効率化を図ることができる．すなわち，ネットワークのトポロジを考慮したデータ転送を行えるため，大規模データを必要とするアプリケーションの転送時間短縮が期待できる．

次章では OmniRPC の概要，3 章で提案システムの概要と 4 章でその実装についてそれぞれ述べる．5 章でグリッド環境上での性能評価を行い，6 章に関連研究を挙げ，終章でまとめと今後の課題について述べる．

2. OmniRPC の概要

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，

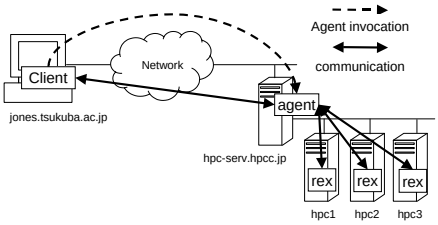

図 1 OmniRPC の概要

シームレスなマスタ/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計算機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，スレーブノードはプライベートアドレスを用いた構成も計算機資源として利用可能である．

OmniRPC の API は，基本的に Ninf[4] の API を踏襲しており，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．

また，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，実行モジュールを起動時に自動で初期化する機能を有している．この機能により，リモート実行プログラムの初期化のための大量のデータ転送や計算を，2 回目の RPC 以降省くことができる．

OmniRPC は，エージェントを使用してクライアントプログラムと複数のリモート実行プログラムとの通信を多重化し一つのコネクションで行うことができる．この通信の多重化を利用することにより，ユーザは，プライベートアドレスで構築されたクラスタや 1000 台規模のリモートホストを利用することができる．

OmniRPC の動作イメージを**図 1** に示す．この図では，通信の多重化が使用され，クラスタの計算ノードに向けて RPC 呼び出しが行われている．OmniRPC において，クライアントプログラムが起動されたときに，omrpc-agent は OmniRPC のホストファイルで指定されたホスト上で起動される．次に，クライアントプログラムの RPC 呼び出しに対して，omrpc-agent が計算ノードにあるリモート実行プログラムを起動する．

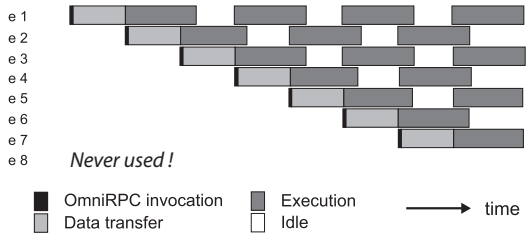

図 2 OmniRPC の問題点

3. データ配送レイヤの利用による効率化

3.1 大規模な初期データが必要な場合の問題点

パラメータサーチのような並列計算において，リモートプログラムを実行させるために必要な入力データは，複数のジョブで共通な「初期データ」とジョブ毎に異なる「パラメータ」に分けることができる．

OmniRPC では，ワーカプログラムで共通に使用される初期データを，RPC ごとに送らずにワーカプログラムの初回起動時に転送し，2 回目以降の RPC でそのデータを再利用できるモジュール自動初期化機能をもつ．しかし，ワーカプログラム起動とモジュール初期化処理はシリアライズされており，すべてのワーカの起動が完了するまでにワーカ 1 台の起動時間の台数倍もかかってしまう．転送データ量の増加に伴って初期化にかかる時間も長くなるため，結果として殆ど，場合によっては一度も使用されないノードが発生する．(**図 2**)．

3.2 データ配送レイヤ OmniStorage

前節の問題を解決するために，データの転送を Grid RPC の機構から分離し，データ転送のためのレイヤを用いて，共通に用いる必要があるデータの転送を行う．マスタは，ワーカを起動する前に，共通に用いるデータを登録し，ワーカはデータ配送レイヤを用いて，そのデータを取得する．我々はデータ配送レイヤのプロトタイプとして OmniStorage を設計，実装した．OmniStorage では，OmniRPC で記述された並列プログラムの中でデータを扱うための API 関数を記述することでこれを実現している．

図 3 に OmniStorage の API を用いた例を示す．マスタ側プログラムでは OmniRPC の非同期呼び出しの前に OmstPutData() を呼び出し，ワーカ側プログラムでは RPC の冒頭で OmstGetData() を呼び出している．データを識別するために，マスタ側とワーカ側の両方で「MyData」という，OmniStorage レイヤにおけるユニークな名前でポインタ「data」が指すデータの登録，取得を行う．データサイズの指定も行うが，これは両者で同じ値を指定する．OmniStorage はこの他にもファイルパスを指定して直接ファイル転送を行う API を持っているが，OmniStorage 上で一意な名前を用いてアクセスされることは同様である．

```
/* master program */
int main(){
  ...
  OmstPutData("MyData", data, OMST_INTEGER * DATALEN);
  for(i = 0; i < n; i++){
       req[i] = OmniRpcCallAsync("MyProcedure", i);
  }
  OmniRpcWaitAll(n, req);
  ...
}

/* worker's IDL */
Define MyProcedure(int IN iter){
  OmstGetData("MyData", data, OMST_INTEGER * DATALEN);
  ...
}
```

図 3 OmniStorage を用いたプログラミング例

OmniStorage ではクライアントからクラスタのマスタノードへの通信とマスタノードからクラスタ内部の各計算ノードへの通信を分離しデータ転送の最適化を行う．クラスタのマスタノードと各計算ノードにキャッシュを持たせて通信の局所化を行わせることによりこの機能を実現している．データのキャッシングを行うことによって，データサイズが大きくなった時にボトルネックとなるクライアントとクラスタ間の通信を一回で済ませることができる．一方でプログラムでは API を用いるだけで OmniStorage の機能を利用できるため，最適化を意識することなくプログラミングを行うことができる．

4. OmniStorage の実装

4.1 OmniStorage のシステム構成

図 4 に OmniStorage のシステム構成を示す．C はジョブを投入するクライアントホスト，W はジョブが実行されるワーカホスト，R はクライアントホストとワーカホスト間でデータの中継を行うホストで，本稿ではリレーホストと呼ぶ．リレーホストはクラスタのマスタノードに置かれ，ワーカホストはクラスタ上の各計算ノードである．図 4 に示した通り，OmniStorage のシステムは木構造のトポロジとなり循環は無く，データは常に根から葉に向かう一方向にのみ流れる．さらに，各ホストはデータを保持するためのキャッシュを持つ．

4.2 OmniStorage の動作

図 5 に OmniStorage の動作の概要を示す．以下，括弧で囲んだ数字は図中の番号に対応する．図中の破線は制御の流れ，実線はデータの流れを表す．また，ここでは Omst-server と Omst-api はそれぞれ OmniStorage のサーバプロセスと同 API の意味とする．

(1) クライアントプログラムが API を通してクライアントのキャッシュに送信するデータを登録する．
(2) OmniRPC でリモートプログラムを起動する．(OmniRpcModuleInit / OmniRpcCallAsync)

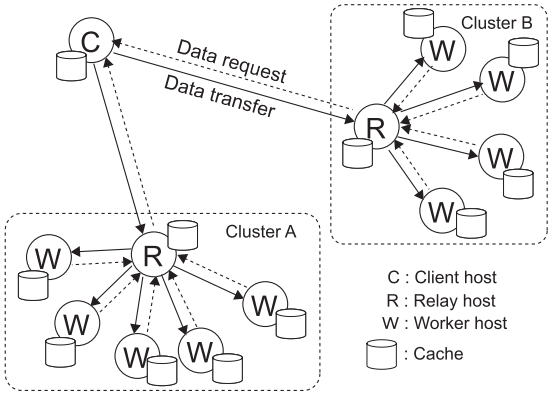

図 4 OmniStorage システム

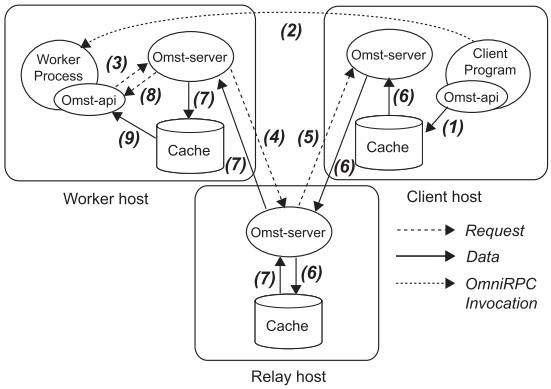

図 5 OmniStorage の動作

(3) 起動されたリモートプログラムがローカルのキャッシュを調べ，必要なデータが無ければワーカホストの Omst-server にデータを要求する．

(4) 要求を受けたワーカホストの Omst-server は上流のリレーホストにデータを要求する．

(5) 要求を受けたリレーホストの Omst-server はローカルのキャッシュを調べ，要求されたデータが無ければさらに上流のクライアントホストにデータを要求する．

(6) 要求を受けたクライアントホストの Omst-server はローカルのキャッシュから要求されたデータを返す．データはデータ要求を出したリレーホストのキャッシュに格納される．

(7) リレーホストの Omst-server がデータ要求を出したワーカホストに対してローカルのキャッシュからデータを返す．データはワーカホストのキャッシュに格納される．

(8) API にレスポンスを返す．

(9) API がキャッシュからデータを読み込み，メモリに格納する．

(1) から (9) までの一連の動作が全て実行されるのはワーカホストとリレーホストのいずれにもデータが無かった場合である．ワーカホストのキャッシュにデータがあった場合は (3) から (9) に飛ぶ．同様にワーカホストのキャッシュになくリレーホストのキャッシュにデータがあった場合は (5) から (7) に飛ぶ．すなわち，あるアプリケーションにおいて一番最初に起動されたジョブが (1) から (9) までの全てのステップを辿るが，2 番目以降のジョブでは途中にあるキャッシュが利用されるため最上位のホストまで辿ることはない．特に同一ワーカ上で 2 回目のジョブが実行されるときは API が直接ローカルのキャッシュを読みに行くため，OmniStorage システムへのアクセスは発生しない．

5. 性能評価

5.1 実験環境

実験に際して，異なるネットワークで接続された 2 クラスタを使用した．またクライアントプログラムを実行するノードとして，筑波大学と東京工業大学にある計算機，それぞれ cTsuku, cTitech を用いた．実験環境を**表 1** に示す．また，**図 6** に各ホスト間のネットワークバンド幅を示す．

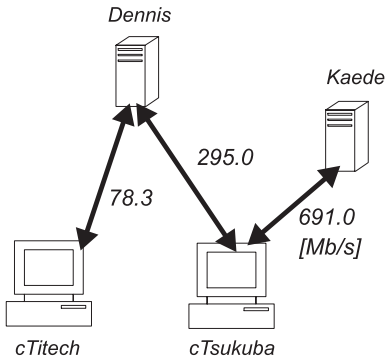

図 6 ホスト間のネットワークバンド幅

5.2 基本性能の評価

性能評価を行うために，OmniRPC を用いたデータ転送を伴う，RPC アプリケーションをモデリングするプログラムを作成した．このプログラムは，初期データのサイズと 1 回の RPC のリモートでの計算時間を変化させ，性能向上比が得られるかを調べる．なお，リモートでは実際に計算を行うのではなく，sleep システムコールを用いて計算時間に相当する時間待機させた．

比較のために，リモートプログラムで OmniStorage の API を利用し初期データを取得するバージョン（OmniRPC + OmniStorage）と，自動初期化モジュール機能で初期データを転送するバージョン（OmniRPC only）を用意した．

初期化データとして転送するデータを 1，16，128，512，1024MB と変化させ，1RPC の実行時間は 5 秒，60 秒，300 秒の 3 通りとした．ワーカとして使用するクラスタの計算ノードの数 16 個に対してジョブの個数を 32 台とした．

また実験環境には，クライアントホストとして cTitech，クラスタとして Dennis を用いた．

表 1　グリッドテストベットの計算機環境

Site	Cluster Name	Machine	Memory	Network	Node 数
HPCC.JP	Dennis	Dual Xeon 2.4GHz	1GB	1Gb Ethernet	16
筑波大学	Kaede	Dual Xeon 3.2GHz	2GB	1Gb Ethernet	64

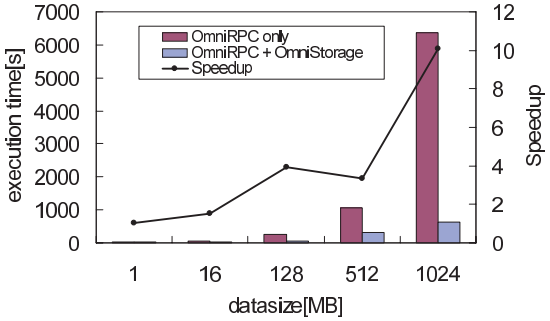

図 7　ワーカ実行 5 秒の性能

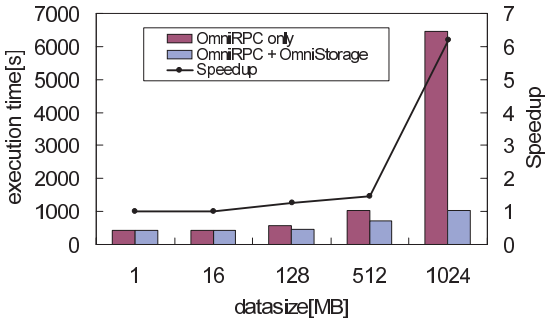

図 8　ワーカ実行 60 秒の性能

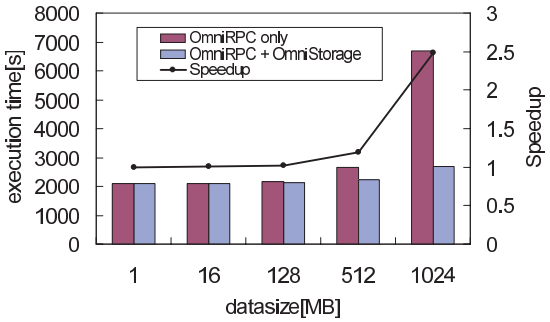

図 9　ワーカ実行 300 秒の性能

図 7,8,9 に実験結果を示す．実験により，1MB から 1024MB までの全てのデータサイズにおいて OmniStorage を使用したことによる性能向上が認められた．また，ジョブ 1 個の実行時間が短くなるほど，データサイズが大きくなるほど性能比は良くなっている．これは全実行時間に占めるデータの転送時間の割合が大きくなると転送の効率化の効果が大きくなるためであると考えられる．また，OmniRPC 単独の場合ではデータサイズがある値を超えると実行時間が大きく増加しているため，データサイズが大きい場合には大きな効果が得られることが分かった．

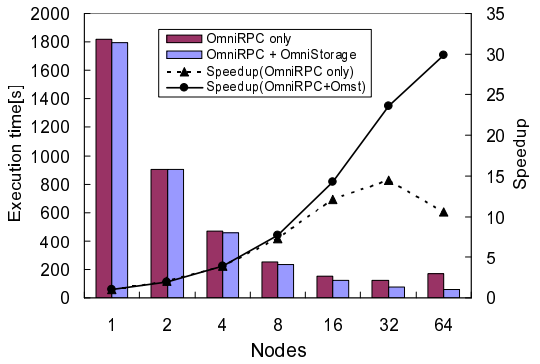

図 10　ノード数による並列固有値計算プログラムの実行時間の変化

5.3　並列固有値計算プログラムを用いた性能評価

並列固有値計算プログラム[3] を利用して，まず一つのクラスタのみを用いて，さらに次の実験では複数クラスタにまたがってジョブ投入を行い，計算ノード数の増加におけるスケーラビリティの評価を行った．なお，固有値計算ジョブの総数は 80 個とした．このプログラムは、大規模な一般固有値問題を解くもので、複素空間上の円周上の点に対応する方程式を並列に解くことにより、その円周内にある固有値を効率的に求めるプログラムである。詳しくは、櫻井らの論文[3] を参照されたい。

図 10 に，一つのクラスタ内で使用する計算ノードの数を変化させた時の全実行時間を比較した結果を示す．クライアントホストには cTsukuba，クラスタは Kaede を用いた．実験結果を示す．折れ線グラフは 1 ノードの実行時間に対しての性能向上比を表す．これより，OmniStorage は 32 台程度までの台数において性能向上が得られることが分かる．それ以上の台数で性能が頭打ちになってくるが，これは固有値計算ジョブの実行時間のばらつきによるものと考えられる．

次に 1 クラスタと 2 つのクラスタを同時に利用した場合の実行時間を比較した．クライアントホストには cTsukuba を使用し，1 クラスタの実験には Kaede の 16 ノード，2 クラスタの実験には Kaede の 16 ノードと Dennis の 16 ノードの合わせて 32 ノードを用いた．**図 11** に実験結果を示す．OmniRPC 単独では 1.06 倍程度の高速化に留まるが，OmniStorage を用いることによって 1.58 倍高速化することができた．

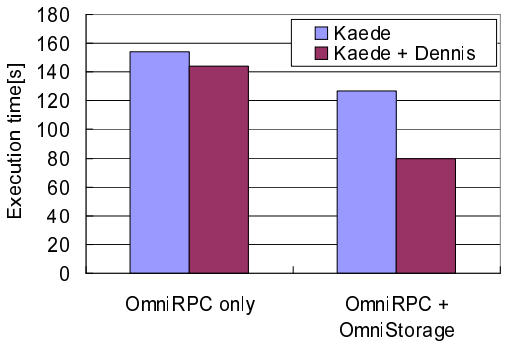

図 11 複数クラスタでの並列固有値計算プログラムの実行時間

6. 関連研究

Grid RPC システムにおいて，ノード間でデータを効率よく管理し性能向上を狙うという研究はいくつかある．

NetSolve[5] では，Distributed Strage Infrastructure と RequestSequencing の 2 つのコンポーネントを使用し，RPC 呼び出しが行われる側のプロセス間において，Persistent なデータを扱うことができる．DSI は，RPC が呼ばれる側のプログラムによって扱われるデータの配置を管理し，クライアントプログラムからワーカプログラムへのデータをキャッシュすることにより転送を効率化させている．

複数の Agent を用いて計算を行う Grid RPC システムである DIET[6] では，データ識別子を用いて Persistent データを管理し，クライアントからデータを一度送ればそれ以降は Agent 間において管理され，各 RPC が呼ばれる側のプログラムでそのデータが利用できる．データの管理は，CORBA で実装された Agent 上でデータのデータ識別子情報を管理する Logical Data Manager と実際のデータを管理する Physical Data Manager の 2 つのコンポーネントを使用することにより実現している．

7. 結論・課題

本稿では各ワーカで共通に使用するデータを分散管理しネットワークトポロジを意識したデータ転送を行うことができる OmniStorage を設計し実装した．その結果，共通データのワーカプログラムへの効率的な転送が実現され，性能評価の結果 OmniStorage を用いた方が OmniRPC 単独の場合よりも高い性能が得られることがわかった．さらにマスタ/ワーカ型のプログラミングモデルにおいて大規模データの転送を伴う場合に広域ネットワーク上の多数の計算機資源をスケーラブルに利用できる可能性を示した．

今後の検討事項としては，現在はマスタからワーカへのデータの配布機能にのみ特化しているが，これに加えて計算結果のデータを収集する機能を追加することが挙げられる．例えば最尤分子系統樹推定アルゴリズム[7] のような，入力データよりも出力データの方が大きいアプリケーションで有効であると考えられる．またデータの配送に木構造ではなく分散ハッシュテーブルを用いることも検討課題として挙げられる．さらに、OmniStorage の API は現状の実装に特化したものではないため，BitTorrent や Gfarm などの広域データ共有ソフトウェアをデータ管理レイヤとして用いることも検討している．

謝辞 実験環境を提供していただいた，東京工業大学松岡研究室に感謝致します．本研究の一部は，文部科学省科学研究費補助金 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) 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.Vol. 44, No.SIG11 (ACS 3), pp.34–45 (2003).
2) Nakajima, Y., Sato, M., Boku, T., Takahashi, D. and Gotoh, H.: Performance Evaluation of OmniRPC in a Grid Environment, *SAINT-W '04: Proceedings of the 2004 Symposium on Applications and the Internet-Workshops (SAINT 2004 Workshops)*, p.658 (2004).
3) 櫻井鉄也, 多田野寛人, 早川賢太郎, 佐藤三久, 高橋大介, 長嶋雲兵, 稲富雄一, 梅田宏明, 渡邊寿雄: 大規模固有値問題の master-worker 型並列解法, 情報処理学会 ACS 論文誌, Vol.Vol. 46, No.No. 10, pp.1–8 (2005).
4) Ninf Project: http://ninf.apgrid.org/.
5) Arnold, D., Agrawal, S., Blackford, S., Dongarra, J., Miller, M., Seymour, K., Sagi, K., Shi, Z. and Vadhiyar, S.: Users' Guide to NetSolve V1.4.1, Innovative Computing Dept. Technical Report, University of Tennessee (2002).
6) Del-Fabbro, B., Laiymani, D., Nicod, J.-M. and Philippe, L.: Data Management in Grid Applications Providers, *DFMA '05: Proceedings of the First International Conference on Distributed Frameworks for Multimedia Applications (DFMA'05)*, pp.315–322 (2005).
7) Yang, Z.: PAML: a program package for phylogenetic analysis by maximum likelihood, *Computer Applications in BioScience*, Vol.13, pp.555–556 (1997).