グリッド RPCシステム Ninf-G の可搬性および適応性の改善

中田秀基[†] 朝生正人[†,††] 谷村勇輔[†]
田中良夫[†] 関口智嗣[†]

Ninf-G はグリッド上で RPC(Remote Procedure Call) を実現するシステムである．従来の Ninf-G は，さまざまなジョブ起動機構に対応することができるが，通信機構に Globus Toolkit の Globus-IO を用いるため，常に Globus Toolkit とリンクする必要がある．このため，1)Globus Toolkit のインストールがユーザに対する導入障壁となる，2)Globus Toolkit に依存するため可搬性が限定される，3)Globus-IO 以外の通信レイヤを利用することができない，といった問題がある．現在実装中の Ninf-G Ver.5 では，これらの問題点を解決するために，リモート通信部を通信プロキシと呼ぶ独立した別プロセスとすることで，コア部分を外部ライブラリ非依存とした．これによって，ユーザに対する導入障壁が解消し，可搬性が向上すると同時に，他の通信レイヤの利用が可能となる．さらに，通信プロキシを利用することで，クライアントとリモート実行ノードが直接通信できない環境においても利用することが可能になる．本稿ではこの Ninf-G Ver.5 の設計について詳述する．さらに，通信プロキシを導入することで予想されるオーバヘッドを見積もるための予備的評価を行う．評価の結果，別プロセス化によるオーバヘッドは無視できないものの，得られるメリットを勘案すれば受け入れることができる範囲であることがわかった．

Portability and Adaptability Improvements in Ninf-G: a GridRPC implementation

Hidemoto Nakada ,[†] Masato Asou ,[†,††] Yusuke Tanimura,[†]
Yoshio Tanaka [†] and Satoshi Sekiguchi [†]

Ninf-G is a system that enables RPC (Remote Procedure Call) on the Grid envirionment. While the existing version of Ninf-G can work with various job invocation mechanisms from various Grid Middleware stacks, it always uses Globus-IO from Globus Toolkit as the communication layer. It means that the modules have to be linked with the Globus Toolkit libraries. As a result, the following issues arise: 1) Globus Toolkit installation could be a barrier for intial users, 2) Globus Toolkit restricts portability of Ninf-G, 3) it is difficult to use communication layer other than Globus-IO. The next incarnation of Ninf-G, the Ver. 5, will solve these issues by introducing external processes for communication, called 'Communication Proxies'. All the Globus dependent portion of the codes is included in the proxies, making the core modules free from dependency. This design enabled to lower the barrier for initial users, make Ninf-G more portable, and make it possible to use other commnication layers Furthermore, the Communication Proxy makes it possible to use Ninf-G where the Client and the remote execution node cannot communicate each other directly. In this paper, we show the detailed design of the Ninf-G ver.5. We also show a result of an experiment to estimate the anticipated overhead comes from introducing the proxies. The result showed that the overhead is substantial but acceptable.

1. はじめに

グリッド上で RPC(Remote Procedure Call) を実現する GridRPC は，グリッド上のプログラミングモデルとして有望なマスタ・ワーカ計算をサポートするのに適した計算機構である．われわれは, この GridRPC API を実装したプログラミング機構として Ninf-G[1),2)] を数年にわたって設計，実装してきた．

Ninf-G はジョブの起動機構を Invoke Server と呼ぶ外部モジュールとしてプラグインすることで，多様な環境に柔軟に適応することができる [3),4)]．しかし，根幹となる通信部分において Globus Toolkit[5),6)] の提供する Globus-IO を利用しており，クライアントやリモートライブラリに Globus-IO のライブラリをリンクしなければならない．このため，起動機構として Globus Toolkit の GRAM を利用しない場合であって

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

も，Ninf-G を利用するためには，Globus Toolkit をインストールする必要がある．

これには 3 つの問題点がある．1 つはユーザベースを制約してしまうことである．Globus Toolkit のインストールはそれほど難しくはないが，パッケージサイズは大きく，独自にコンパイルする場合にはコンパイルにも長大な時間がかかる．特に単一クラスタ内で運用する GridRPC アプリケーションを開発する際には，Globus Toolkit の提供する機能は意味をなさない．にもかかわらず Globus Toolkit のインストールが必要であることはユーザにとって Ninf-G インストールの大きな，障壁となる．

もう 1 つの問題点は，プラットフォームを制約してしまうことである．Globus Toolkit は精力的にさまざまなアーキテクチャ，OS にポーティングされているものの，すべてのアーキテクチャ，OS で稼働するわけではない．Globus Toolkit に依存していることによって，Ninf-G が稼働するプラットフォームが制約されてしまう．さらに，もう 1 点は Globus-IO 以外の通信レイヤを用いることができないことである．たとえば Ninf-G がサポートしている起動機構の一つに Condor[7] があるが，Condor では Chirp[8] と呼ばれる通信レイヤを提供している．しかし，Chirp を Ninf-G で利用することはできない．

この問題を解決すべく，我々は新たに Ninf-G Ver.5 (以下 Ninf-G5) を設計, 開発している．Ninf-G5 は，Invoke Server のコンセプトを押し進め，リモート通信部分を通信プロキシとして別プロセスとすることでコアとなるクライアント，リモートライブラリから，各グリッドミドルウェアの通信機構に依存する部分を排除する．これにより，各グリッドミドルウェアの提供するセキュアな通信機構の恩恵を受けつつ，コア部分の可搬性を向上させることが可能となる．また，通信プロキシを利用する副次的な効果として，クライアントとリモート計算モジュールが直接通信できない環境においても，Ninf-G を利用することが可能になる．

本稿の以下の構成を以下に示す．2 節で従来の Ninf-G の概要の説明と問題点の指摘を行う．3 節で新たな Ninf-G5 の設計について述べる．4 節では，あらたに導入する通信プロキシによるオーバヘッドを見積もるための実験を行う．5 節で関連研究について述べ，6 節で，本稿のまとめと今後の課題について述べる．

2. 従来の Ninf-G の概要

Ninf-G は RPC(Remote Procedure Call) 機構をグリッド上で実現する GridRPC システムである．GridRPC[9] は，OGF(Open Grid Forum) で標準化が進められている API 規格で，Ninf-G はこの API 規格に準拠している．

図 1 に Ninf-G の動作概要を示す．Ninf-G は大きく分けてクライアントとリモート計算モジュールの二つのプログラムから構成される．クライアントは，サーバ上のリモート計算モジュールに計算を依頼し，結果を受け取る．ひとつのクライアントから，複数のリモート計算モジュールを同時に利用することで，並列実行を容易に実現することができる．

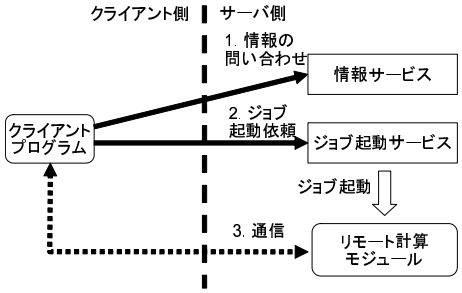

図 1 Ninf-G の概要

```
grpc_function_handle_t handle;
grpc_error_t res;
    ...
res = grpc_function_handle_init(&handle,
   "server.example.org", "test/func");
res = grpc_call(&handle, 100, A, B);
```

図 2 クライアントプログラムの例

2.1 クライアントプログラム

Ninf-G のクライアントプログラムの例を図 2 に示す．ハンドルと呼ばれる構造を，サーバと実行プログラム ID を指定して作成し，それに対して`grpc_call` で引数を指定して計算を依頼する．ユーザが引数のマーシャリングを明示的に行う必要がないのが，GridRPC の特徴である．

2.2 Ninf-G の構成

Ninf-G が必要とするグリッド関連機能は，リモート計算モジュールのインターフェイス情報を取得するための情報サービス，対象サーバでリモート計算モジュールを起動するジョブ起動機構，クライアントとリモート計算モジュール間で通信を行うための通信機構，の 3 つである．

従来の Ninf-G は Globus Toolkit(以下 GT)[5),6)] の使用を前提としており，情報サービスとして MDS2，デフォルトのジョブ起動機構として pre-WS GRAM 通信機構として Globus-IO を利用している．これらの機能はクライアントライブラリにリンクされている．Globus-IO は，リモート計算モジュール用のライブラリにもリンクされる．

Globus Toolkit への依存を解消し，可搬性と適応性を改善するためには，これらの機能を Ninf-G のコア部分からとりだし，外部モジュールとする必要がある．このうち情報サービスに関しては，ローカルファイルを利用するモジュールで，ジョブ起動機構に関しては Invoke Server[3),4)] と呼ばれる外部モジュールを

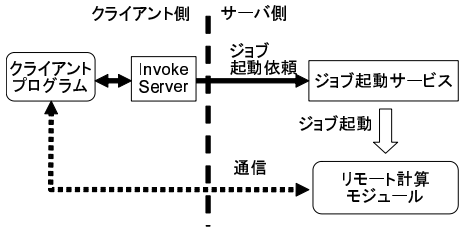

図 3 Invoke Server の概要

用いることで，コア部分から排除することが，すでに可能となっている．図 3 に Invoker Server の概要を示す．したがって，残る通信機構の外部化を行えば，可搬性と適応性を改善することができる．

2.3 起動と通信

Ninf-G における起動と通信は，下記のように行われる．

- クライアントは，接続用のポートをオープン．
- クライアントが，ジョブ起動機構を経由して，リモート計算モジュールを起動．この際に，クライアントの IP アドレスとポートを引数として渡す．
- リモート計算モジュールは，引数で指定された IP アドレスとポートに対して接続を行う．

リモート計算モジュールが接続を主導するため，リモート計算モジュールがプライベートアドレスしか持たない計算ノードで起動される場合にも，NAT などでクライアントへの接続性が保たれていれば，Ninf-G を利用することができる．

2.4 Globus-IO

Globus-IO は，Globus Toolkit が提供する通信ライブラリで，GSI (Grid Security Infrastructure) と呼ばれる，プロキシ証明書を利用したシングルサインオンを実現する認証機能や，SSL に準じたセキュアな通信を提供する．

このライブラリは C 言語で記述されているが，エキセントリックなスレッドモデルに基づいている．Globus-IO が設計されたのは 1998 年頃であり，このころにはまだネイティブスレッドをもつ OS が一般的ではなかった．このため，Globus チームは，独自の擬似スレッド機構を実装したが，このスレッド機構は I/O の際に他のスレッドに制御を譲る機能を持たなかった．このため，Globus-IO の提供する関数群は，イベント駆動モデルで良く用いられる，コールバック関数を指定するノンブロッキング関数となっている．現在は，通常の pthread 上の実装も提供されているが，関数の仕様は変更されていない．このため，Globus-IO を利用する際には，通常のスレッドモデルと同様に，排他制御を明示的に意識しつつ，イベント駆動モデル的な記述を行わなければならない．

ここで重要なのは，Globus-IO の提供するインターフェイスが，通常のソケットライブラリとは本質的に異なる点である．このため，単純なラッピング関数で両者を抽象化することができない．また，Globus-IO を利用する際には Globus の提供するスレッドモデルで全てのプログラムを記述する必要がある．

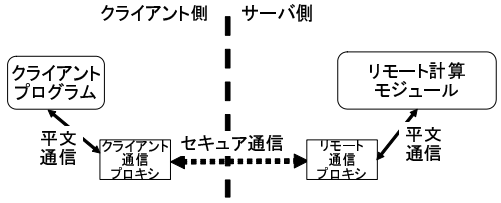

図 4 通信プロキシ機構の概要

3. Ninf-G5 の設計

3.1 要　　請

ここで，Ninf-G5 における通信機能への要請を整理する．

- 特定のグリッドミドルウェアに依存せずに Ninf-G のクライアントおよびリモート計算ライブラリをコンパイル，リンクできること．
- 必要に応じて，個別のグリッドミドルウェアの持つ通信機能が利用できること．

3.2 設　　計

上述の要請を満たすべく，通信プロキシを導入する．通信プロキシは，通信を中継するプロセスで，クライアント側とリモート計算モジュール側の双方で稼働する．図 4 に通信プロキシの概念を示す．通信プロキシは必要に応じて個別のグリッドミドルウェアの提供するライブラリとリンクされる．クライアント側の通信プロキシをクライアント通信プロキシ，リモート計算モジュール側の通信プロキシをリモート通信プロキシと呼ぶ．

クライアント通信プロキシはクライアントと，リモート通信プロキシはリモート計算モジュールと，それぞれ通常のソケット通信を行う．通信プロキシ同士は，個別のグリッドミドルウェアの提供する通信レイヤで通信を行う．したがって，クライアント，およびリモート計算モジュールは，通常のソケット通信のみをサポートすればよく，特定のグリッドミドルウェアに非依存とすることができる．

クライアント通信プロキシは，基本的に 1 つのクライアントプログラムに対して 1 つだけ起動される．これに対して，リモート通信プロキシは個々のリモート通信モジュールに対してそれぞれ起動する．一般に一つのクライアントは複数のリモート通信モジュールと通信するので，一つのクライアント通信プロキシが複数のリモート通信プロキシと通信することになる．

3.3 通信プロキシの起動と通信路の確立

通信プロキシの起動方法にはいくつかのバリエーションが存在するが，ここではもっとも基本的な起動方法について説明する．他の起動方法は次節以降で述

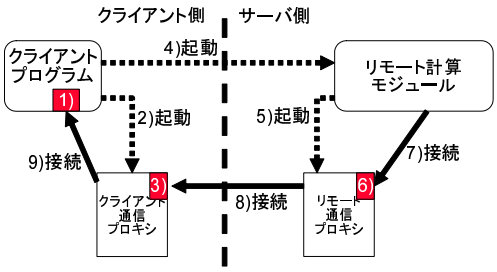

図 5 通信プロキシによる通信路の確立

べる．

クライアント通信プロキシは，クライアントプログラムから起動される．リモート通信プロキシは，リモート計算モジュールから起動される．クライアントとクライアント通信プロキシ，リモート計算モジュールとリモート通信プロキシはそれぞれ，制御用のパイプで接続される．このパイプは，親プロセスで作成したものを fork 後に stdin/stdout/stderr にバインドし，その後通信プロキシを exec することによって作成する．したがって，通信プロキシプロセスから見ると，制御用のパイプは標準入出力，標準エラーに割り当てられることになる．

通信プロキシによる通信路確立の様子を，図 5 に示す．

(1) クライアントが通信ポートを作成する
(2) クライアントが，クライアント通信プロキシを起動する．クライアントの通信ポートを通知する．
(3) クライアント通信プロキシは，受信用ポートを作成し，ポート番号をクライアントプログラムに通知する．
(4) クライアントは，ジョブ起動機構経由でリモート計算モジュールを起動する．この際に，クライアント通信プロキシのポート番号を通知する．
(5) リモート計算モジュールは，リモート通信プロキシを作成し，クライアント通信プロキシのポート番号を通知する．
(6) リモート通信プロキシは受信ポートをオープンし，そのポート番号をリモート計算モジュールに通知する．
(7) リモート計算モジュールは，リモート通信プロキシに接続する．
(8) リモート通信プロキシは，クライアント通信プロキシに接続する．
(9) クライアント通信プロキシは，クライアントに接続する．

3.4 通信プロキシによる非対称通信路への対応

上述した起動手法では，通信プロキシはクライアントもしくはリモート計算モジュールと同一ノード上で稼働することとなっている．しかし，通信は単なるソケット通信であるから，同一ノード上で稼働させる必要は必ずしもない．通信プロキシをファイアウォール上で稼働させることによって，ファイアウォールに代表される非対称通信路に対応することが可能となる．

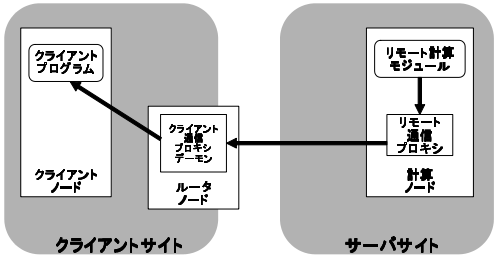

図 6 プライベートアドレスクライアントへの対応

3.4.1 クライアント通信プロキシデーモンによるプライベートアドレスクライアントへの対応

従来の Ninf-G では，通信路の確立がクライアントノードへのコネクトバックで行われるため，クライアントノードがリモート計算モジュールから直接通信可能でなければ，利用することができない．したがって，NAT によって外部へのアクセス可能ではあるがプライベートアドレスしか持たないノードからは利用することができない．

これに対して，ルータ計算機上で事前にクライアント通信プロキシを起動しておくことによって，このような環境でも Ninf-G を利用することが可能となる．このようなクライアンと通信プロキシをクライアント通信プロキシデーモンと呼ぶ．図 6 にこの様子を示す．

3.4.2 中継プロキシによる非 NAT プライベートアドレスクラスタへの対応

多くのクラスタでは，計算ノードがプライベートアドレスしか持たない構成が一般的である．2.3 項で述べたように，このようなクラスタであっても，計算ノードから外部ネットワークへのアクセスが NAT によって確保されていれば，Ninf-G を利用することができる．しかし，クラスタによっては，計算ノードからの NAT による外部アクセスすら許さない場合がある．このようなクラスタでは従来の Ninf-G を利用することはできない．

このような場合にも，リモート通信プロキシをクラスタのゲートウェイとなるノード上で起動することで接続性を確保することができる．このようなプロキシを中継プロキシと呼ぶ．中継プロキシは事前にログインして手動で起動するか，グリッドミドルウェア経由で起動する．図 7 にこの様子を示す．

3.5 セキュリティに関する考察

通信プロキシを導入することで，不可避的に新たなセキュリティリスクが生まれる．クライアントがオープンするポートとリモート通信プロキシがオープンするポートは，通常の平文ソケットであり，暗号的に認証していない．しかし，これらのソケットはノード内

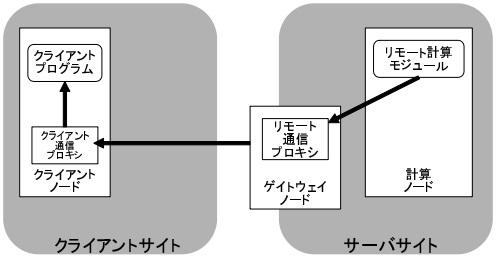

図 7 非 NAT プライベートアドレスクラスタへの対応

通信にのみ用いられるため，他ノードからの攻撃は考慮する必要はなく，比較的安全であると考えられる．

中継プロキシは，リモート計算モジュールと別のノードで稼動するため，危険度は多少高くなる．しかしこの場合も，受信ポートはクラスタ内部ネットワークに対してだけオープンすればよく，危険度は低い．

さらに，Ninf-G5 では，クライアントとリモート計算モジュールの間で，通信路確立後に，共通鍵を用いた認証を行う．共通鍵はクライアント側で生成され，ジョブ起動時にファイルとしてサーバ側に転送される．このファイルのパーミッションを所有者のみリード可能にしておくことで一定のセキュリティレベルが達成できると考えられる．

4. 実　　験

通信プロキシを挿入することによって予想される通信速度の低下の大きさを知るために，実験を行った．実験には単純なピンポンを行うモジュールを作成し使用した．モジュールは，クライアント，サーバ，クライアントプロキシ，サーバプロキシの 4 つで，それぞれ Globus-IO で通信を行う．通信モードとして通常のソケット通信に相当する通常モードと，SSL 通信と等価な暗号化を行う GSI モードを利用することができる．

実験に用いた機器の設定を図 8 に示す．実験では，下記の 4 つ条件で比較を行った．

- **NONE**
  クライアントとサーバが直結して通常通信．
- **SECURE**
  クライアントとサーバが直結して暗号通信．
- **PROXY**
  クライアントプロキシ，サーバプロキシを経由して通常通信
- **PROXY_SECURE**
  クライアントプロキシ，サーバプロキシを経由して通信．クライアントプロキシ，サーバプロキシ間は暗号通信．その他は通常通信．

実験はそれぞれ数十回行い，最速値を採用した．

図 9 に，100base/TX で接続した場合の実験結果を示す．この 2 つの計算機間にはハブとスイッチがいくつか入っている．横軸は転送したデータ量，縦軸はスループット (Mbyte/s) である．プロキシを利用した場合でも直結通信の 8 割程度の速度が出ていることがわかる．同様に，図 9 に 1000base/T で接続した場合の実験結果を示す．この 2 つのノードはスイッチを介して直結されている．このような環境では，プロキシをはさむことによるオーバヘッドは大きく，スループットが半減していることがわかる．

図 8 実験環境

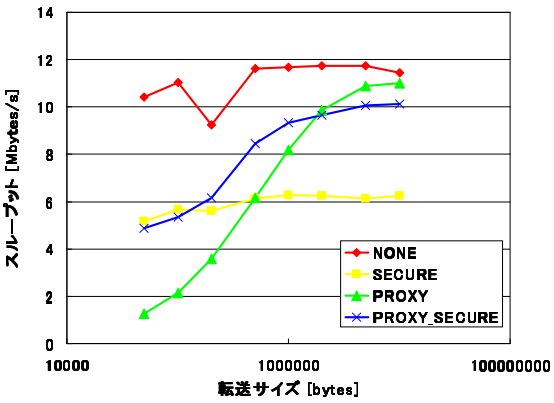

図 9 100base/TX での実験

両者において，セキュリティを用いる場合には，プロキシを経由した方が場合によって高速であるという結果がでているが，これはプログラムのチューニングの問題であると思われる．

5. 関 連 研 究

OmniRPC[10] は，Ninf-G と類似した API を持つ RPC 処理系である．OmniRPC は，通常のソケット通信と Globus-IO の双方をサポートしているが，Globus Toolkit がない環境では，Globus-IO なしでビルドすることができる．しかし，他の通信レイヤに対応するためには，ライブラリのソースコードを直接変更する必要があると思われる．

6. お わ り に

GridRPC システム Ninf-G5 の通信プロキシ機構について述べた．通信プロキシ機構を用いることで，Ninf-G5 のコア部分がグリッドミドルウェア対して非

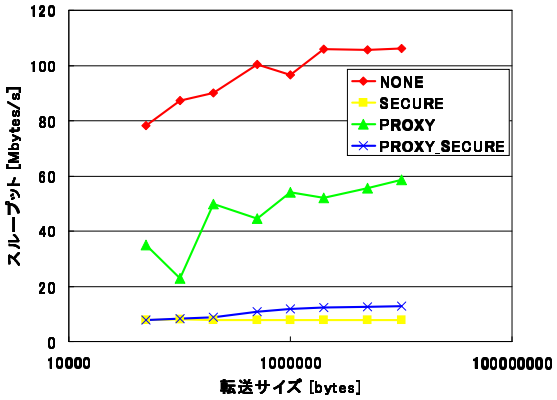

図 10 1000base/T での実験

依存となり，導入障壁の低下と可搬性の向上，他の通信レイヤの採用が可能となる．

予備評価として通信プロキシを用いる構成での通信速度を測定したところ，通信プロキシのオーバヘッドは高速ネットワーク環境においては無視できないものの，得られるメリットと勘案して受け入れることが可能であると考えられる．

Ninf-G5 は本年度中のリリースを目指して，現在実装中である．

今後の課題としては以下が挙げられる．

- 今回示した実験で用いたプログラムは，簡単なテスト用のプログラムであり，Ninf-G 本来の通信パターンを完全に反映したものではない．Globus-IO 用 通信プロキシの実装を完了し，実際の Ninf-G の通信パターンでのオーバヘッドを計測する必要がある．
  また，今回の実験結果には不可解な点がいくつか見られたが，これはチューニングの余地があることを示していると考えられる．実装完了後，チューニングを進めていく必要がある．
- 通信プロキシ機構の有用性を確認するためには，他のグリッドミドルウェアの通信機構用の通信プロキシを実装する必要がある．Condor の Chirp を用いた通信プロキシを実装する予定である．
- Ninf-G5 はさまざまなコンフィギュレーションでの実行を念頭に置いてはいるが，リモート計算モジュールを実行する計算機が TCP/IP を利用できることを仮定している．しかし，この仮定はすべての計算機で真ではない．例えば，Cray XT3[11] では，各計算ノード上のプロセスが利用できるのは特殊なファイルベースの通信ライブラリだけであり，TCP/IP 通信を利用することはできない．このような環境でも Ninf-G5 を利用できるようにするためには，通信ライブラリをさらに抽象化し，リモート計算モジュールが TCP/IP 通信以外の通信を利用できるようにする必要がある．

謝　　辞

設計，実装にご協力いただいた，産総研 Ninf 開発チームの皆様に感謝します．

本研究の一部は文部科学省「経済活性化のための重点技術開発プロジェクト」の一環として実施している超高速コンピュータ網形成プロジェクト (NAREGI：National Research Grid Initiative) によるものである．

参 考 文 献

1) Nakada, H., Tanaka, Y., Matsuoka, S. and Sekiguchi, S.: *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd, chapter Ninf-G: a GridRPC system on the Globus toolkit, pp. 625–638 (2003).
2) Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T. and Matsuoka, S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *Journal of Grid Computing*, Vol. 1, No. 1, pp. 41–51 (2003).
3) 中田秀基, 田中良夫, 関口智嗣: GridRPC システム Ninf-G における UNICORE および GT4 によるジョブ起動, 情報処理学会ハイパフォーマンスコンピューティングシステム研究会, Vol. 2005-HPC-102, pp. 45–55 (2005).
4) 中田秀基, 田中良夫, 関口智嗣: GridRPC システム Ninf-G のリモート起動手法の改良, 情報処理学会ハイパフォーマンスコンピューティングシステム研究会, Vol. 2006-HPC-108, pp. 43–47 (2006).
5) Globus Project. http://www.globus.org.
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) Thain, D., Tannenbaum, T. and Livny, M.: Condor and the Grid, *Grid Computing: Making the Global Infrastructure a Reality* (Berman, F., Fox, G. and Hey, T.(eds.)), John Wiley & Sons Inc. (2002).
8) Chirp. http://www.cs.wisc.edu/condor/chirp/.
9) Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C. and Casanova, H.: GridRPC: A Remote Procedure Call API for Grid Computing, submitted to Grid2002.
10) Sato, M., Boku, T. and Takahasi, D.: OmniRPC: a Grid RPC System for Parallel Programming in Cluster and Grid Environment, *Proceedings of CCGrid2003*, pp. 206–213 (2003).
11) Cray XT4 and XT3 Supercomputers, http://www.cray.com/products/xt4.