社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－108 (8)
2006／10／6

グリッドRPCシステムNinf-Gのリモート起動手法の改良

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

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GridRPC APIを実装したプログラミング機構としてNinf-Gを設計，実装している．Ninf-Gは基本的にGlobus Toolkitを用いて実装されているが，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，多様な実行環境に柔軟に適応することができる．Invoke Server機構の設計に関しては，すでに別稿で報告している．われわれは，Globus Toolkit 4 WS GRAM, Unicore, Condor, ssh, NAREGIミドルウェアβに対してそれぞれInvoke Serverを実装した．また，Invoke Server機構のオーバヘッドを評価するためにジョブ起動時間を測定した．その結果，Invoke Serverによって導入されるオーバヘッドはジョブ起動機構自身のオーバヘッドと比較すると十分小さいことがわかった．本稿では，より適応範囲を広げるために計画されている，より高度なモジュール化に関しても議論する．

Remote invocation method improvements in Ninf-G: a GridRPC implementation

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

GridRPC, a programming API that enables Remote Procedure Call on the Grid, is considered to suit well with the Master-Worker type computation, which is a programming paradigm that can leverage huge computation power of the Grid environment. We have been proposing an implementation of the API, called Ninf-G. While Ninf-G is implemented based on the Globus Toolkit, it also can utilize other job invocation systems via external modules called Invoke Servers. Previously, we reported on the Invoke Server mechanism in an separate article. In this paper we report our implementation of Invoke Servers for Globus Toolkit WS GRAM, Unicore, Condor, ssh, and NAREGI middle ware beta. We also report the measured job invocation cost with each Invoke Server. The results indicated that the overhead introduce by the Invocation Server mechanism is negligible compared with the job invocation cost itself. We also discuss on further improvement in terms of modularity.

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．われわれは数年にわたって，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)]を設計，実装してきた．

Ninf-Gは基本的にGlobus Toolkitを用いて実装されているが，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．Invoke Serverのプロトコルの設計に関しては，すでに別稿[3)]で報告している．

本稿では，Globus Toolkit4, Unicore, Condor, ssh, NAREGIミドルウェアβへ対応したInvoke Serverの設計と実装に関して詳述する．また，Invoke Server機構のオーバヘッドを評価するために行った各Invoke Serverによるジョブ起動時間の測定に関しても報告する．さらに，より適応範囲を広げるために計画されている，より高度なモジュール化に関しても議論する．

本稿の以下の構成を以下に示す．2節でNinf-Gのアーキテクチャと，Invoke Serverの概要を示す．3節で個々のInvoke Serverに関して詳述する．4節でいくつかのInvoke Serverを用いた際の起動時間を示す．

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

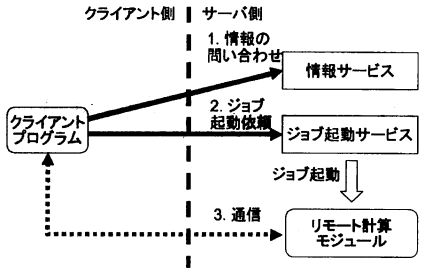

図 1 Ninf-G の概要

```
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

5 節で，Ninf-G のより高度なモジュール化について述べる．6 節で，本稿のまとめと今後の課題について述べる．

2. 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[4] は，OGF(Open Grid Forum: 2006 年 7 月に GGF から改称) 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．

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 の構成と Invoke Server

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．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．デフォルトジョブ起動機構の pre-WS GRAM を使用するためのライブラリコードは，自動的にクライアントプログラムにリンクされる．

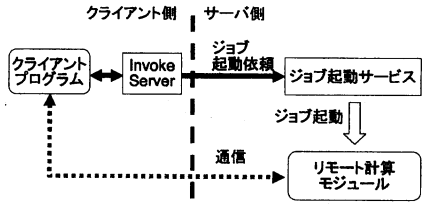

図 3 独立したジョブ起動モジュール

Ninf-G を Globus 以外のグリッドミドルウェアで運用するためには，これら 3 つの機能を実現する必要がある．ただし，この 3 つの機能のうち，情報サービスはオプショナルで，運用によっては使用する必要はない．また，通信機構は単純なライブラリでデプロイも必要なく，静的にバイナリにリンクすることも可能なためそれほど問題にならない．したがって，残るジョブ起動機構を対象グリッドミドルウェアに対応することができれば，Ninf-G をそのミドルウェア上で運用することが可能になる．

この際，ジョブ起動機構をクライアントプログラムとリンクする形で実装すると，あらたなジョブ起動機構に対応するたびに，Ninf-G のクライアントライブラリそのものに変更を加えなければならない．また，グリッドミドルウェアによっては，C 言語の API をもたないものもあり，その場合にはクライアントプログラムとリンクする形での実装は不可能となる．

このため，Ninf-G では Invoke Server と呼ぶ機構を導入した[3)]．Invoke Server は，ジョブ起動機構部分を別プロセスとしてクライアントライブラリから外に出したものである．Invoke Server とクライアントプログラムの間は，シンプルなテキストベースの通信プロトコルで通信が行われるため，スクリプト言語で Invoke Server を実装することも容易である．

3. Invoke Server

本節では，主な Invoke Server の実装の詳細を述べる．

3.1 WS GRAM 用 Invoke Server

Ninf-G には，Globus Toolkit 4 の Web サービス GRAM (WS GRAM) に対応した Invoke Server が 2 つ用意されている．ひとつは，Python で実装された GT4py，もうひとつは，Java で実装された GT4java

☆ GT version 2 以来サポートされているジョブ起動機構．GT4 の主要起動機構である Web サービス (WS) を用いた WS GRAM と区別するために，pre-WS GRAM と呼ばれている

である．前者を使用することが推奨されている．

Globus Toolkit 4には，WS GRAMに対するCのAPIが存在しない．Cで実装されたコマンドラインプログラムは存在するが，直接プログラムから使用できるAPIが設定されていない．これに対して，Java言語では，明示的に設定されたAPIが存在する．

われわれは，スクリプト言語であるPythonで記述され，C言語で書かれたコマンドラインプログラムを利用するInvoke Server GT4pyと，Javaで書かれ，Globus Toolkitの提供するAPIを直接起動するInvoke Server GT4javaを実装した．

3.2 Invoke Server SSH

Globus Toolkitは，多くのグリッドプロジェクトで利用されてはいるが，デプロイと運用にコストがかかるため，一般的なクラスタでの利用が進んでいるとは言いがたい．これにたいして，sshは事実上全ての計算機で使用できるため，sshをジョブの起動に利用することができれば，Ninf-Gの利用を促進することができる．

しかし，sshを利用した場合には，いくつかの制約が生じる．ひとつは計算モジュールとクライアント間の通信でGSIを利用した暗号化ができないことである．もうひとつは，リモート計算モジュールからGlobusのセキュリティを利用して他の資源へアクセスすることができないことである．

前者は，後述するポートフォワーディングを用いることで部分的には解決する．後者は一般的な利用の範囲では問題ないと考えられる．

3.2.1 基本構造

Invoke Server SSHはC言語で記述されている．Invoke Server SSHは，サーバに対してsshで接続を行い，サーバ上にシェル (/bin/sh) を起動する．このシェルを用いてサーバ側でコマンドを実行することで，サーバ上でのジョブを制御する．サーバ側で利用するスクリプト言語をシェルとしたのは，最大限のポータビリティを得るためである．

単一のジョブを直接フォークで起動する場合の動作を説明する．まず，ジョブをバックグラウンドで起動し，プロセスIDを取得する．ジョブのモニタリングには，`kill -0` を用いる．取得したプロセスIDに対してこの操作を行うと，プロセスが生存していれば0が，すでに終了していれば1が返されるので，ジョブの状態を知ることができる．

3.2.2 ファイルの転送

Ninf-Gは，実行ファイルのステージングと，標準出力，標準エラー出力ファイルのステージアウトをサポートしている．Invoke Server SSHはこの機能を実現するために，制御用のssh の他に，個々のファイルにつき，ひとつのファイル転送用sshを用いる．

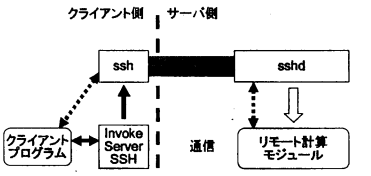

図4 sshのポートフォワーディングを用いた通信路暗号化

3.2.3 ローカルバッチキューイングシステムへの対応

グリッド環境のクラスタの多くは，ローカルバッチキューイングシステムで管理されている．Globus Toolkitには，このローカルバッチキューイングシステムを抽象化するjob managerという概念がある．job managerがローカルバッチキューイングシステムの相違を隠蔽するため，クライアント側からは，job managerを指定するだけで背後にあるシステムを意識せずに，ジョブを実行することができる．

Invoke Server SSHではGlobusを使用しないため，このjob manager機構を利用することはできない．したがって，ローカルバッチキューイングシステムに独自に対応する必要がある．

現在Invoke Server SSHは，ローカルバッチキューイングシステムとしてSun Grid EngineとPBS(TORQUEおよびPBS Professional)をサポートしている．ローカルバッチキューイングシステムを利用してジョブを実行する際には，ジョブをシェルから直接起動するのではなく，qsub, qstat, qdelコマンドをサーバ側で実行してジョブを制御する．

ジョブを実行する際にはクライアント側でスクリプトファイルを生成し，これを上述したファイル転送機構を用いてサーバ側に転送し，シェルを通じてqsubコマンドでサブミットする．ジョブの状態取得はqstatで，ジョブのキャンセルはqdelで行う．

3.2.4 ポートフォワーディングによる通信の暗号化

sshはサーバ側にポートをオープンし，そこへの接続をクライアント側からの通信として指定されたホスト，ポートへリダイレクトする機能を持つ．この通信はsshの機能によって暗号化される．

この機能を用いることで，Invoke Server SSHを利用した際にもクライアントとサーバ側サイトの間の通信を暗号化することができる．図4にこの様子を示す．この図からもわかるように，クライアント内，およびサーバ側サイト内での通信は平文となるため，Globus-IOの暗号化機能を利用した場合よりは安全性は低下する．しかしほとんどの場合はこれで十分であると考える．

3.3 Invoke Server Condor

Invoke Server Condorは，Wisconsin大学で開発されたジョブ管理機構であるCondor[7]に対して直接ジョブを投入する機構である(図5)．クライアントノー

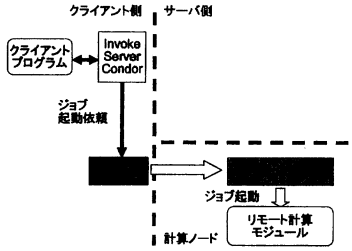

図 5 Invoke Server Condor の動作

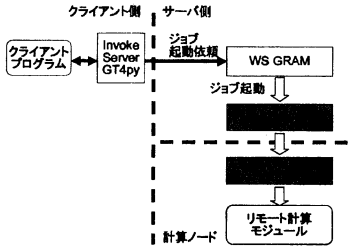

図 6 Globus Condor jobmanager

ドに Condor のジョブ投入デーモンである schedd を配置し，直接ジョブを投入する．

Globus Toolkit は，Condor に対する job manager をサポートしているため，Invoke Server Condor を用いなくても Globus 経由で Condor に job を投入することもできる (図 6)．しかしイントラネット環境で Ninf-G と Condor を利用する場合，Globus をデプロイするコストを省くことは重要だと考え，Condor に直接ジョブを投入することのできる Invoke Server Condor を実装した☆．

Invoke Server Condor は，Java で記述された Condor のコマンドラインインターフェイスのラッパ [9] を利用して，Java 言語で実装されている．Condor には C 言語による DRMMA API や SOAP API[10] が用意されているが，実装の容易さ簡潔さから Java によるラッパを選択した．

3.4 Invoke Server NAREGI ミドルウェア β

NAREGI プロジェクト [11] は，文部科学省の主導するグリッドプロジェクトで，ワークフローの実行と複数サイトにまたがる並列実行に重点をおいたミドルウェア [12] を開発している．Ninf-G はこのプロジェクトの一環として実装されている．NAREGI ミドルウェア β は，OGF(Open Grid Forum, 旧 GGF) で主導される OGSA(Open Grid Service Architecture) や各種 Web Service 標準への対応を重視して開発されている．

この NAREGI ミドルウェア β で管理された資源に対してジョブを投入するための Invoke Server を開発した．NAREGI ミドルウェア β はジョブ投入のインターフェイスとして，XML で記述されたワークフローをやりとりする Java 言語 API を持つ．Invoke Server はジョブ投入リクエストを XML に変換して投入し，インターフェイスから返却されるジョブ状態を示す XML 表現を解釈する．

4. 評　　価

各起動機構の性能に与えるインパクトを知るために性能を測定した．RPC の性能を評価する際には，RPC 関数の起動までの時間 (レイテンシ) と RPC 関数との通信速度 (スループット) の 2 点が重要となる．今回の評価対象である起動機構はレイテンシのみに関与し，スループットに影響するデータ通信路には関与しないため，今回はレイテンシのみに着目して評価を行った．

具体的には，`grpc_function_handle_init()` に引き続き，ダミーの関数呼び出し行い，これらにかかる時間を測定した．ダミーの関数は，まったく計算を行なわず，転送引数サイズも 4 バイトと小さい．

4.1 測定環境

クライアントとして東京秋葉原に設置した PC を，サーバとしてつくばに設置したクラスタを使用した．両者のスループットは，36.5 Mbit/sec 程度 (iperf を用いて計測)，ping レイテンシは 6.2ms (ラウンドトリップ) 程度である．測定時にはクライアント，サーバともに他のユーザはなく，占有環境で測定を行った．

クライアント PC は，Athlon 2GHz Dual Core, メモリ 4Gbyte, OS は Fedora Core 5, サーバクラスタは 1 台のヘッドノードと 4 台のワーカノードから構成され，各ノードは Pentium III 1.4 GHz Dual CPU, メモリ 2Gbyte, OS は RedHat 8 となっている．

Globus Toolkit は クライアント，サーバともバージョン 4.0.2 を，Condor は バージョン 6.8 を使用した．GridEngine(SGE) は 6.0 update 7 である．

参考のため各手法を用いて /bin/hostname をサブミットした際の結果がクライアント側に表示されるまでの時間を表 1 に示す．このコストには，ジョブの起動だけでなく終了のコストおよび出力された文字列の転送コストが含まれているため，後述する RPC ジョブの起動時間に比べるとかなり大きな値になっている項目もあることに注意されたい．pre-WS GRAM では globus-job-run コマンドを，WS GRAM では globusrun-ws コマンドを用いてジョブを投入した．

4.2 測定項目

各 Invoke Server とサーバ側のローカルバッチキューイングシステムの組み合わせに対して測定を行った結果を表 2 に示す．測定はそれぞれ 20 回行い，最初の

☆ 筆者らは過去に，Condor のみを対象とした RPC システム Ninf-C[8] も実装している．Ninf-C は Condor の持つ耐故障性を最大限に引き出すよう設計されているが，Condor 以外のシステムを利用することはできない．一方 Ninf-G は，複数の Invoke Server を利用することでさまざまなグリッドシステムを同時に利用することができる．

表 1 各起動方法による/bin/hostnameにかかる時間 (秒)

	Fork				Condor				SGE			
	平均	分散	最小	最大	平均	分散	最小	最大	平均	分散	最小	最大
pre-WS GRAM (GT2)	1.05	0.00	1.02	1.09	19.51	19.94	11.71	22.19	14.67	22.47	11.52	22.04
WS GRAM (GT4)	6.49	0.52	5.59	7.96	20.10	1.80	17.62	22.51	44.60	4.26	37.56	47.22
Invoke Server SSH	0.25	0.00	0.24	0.26	-	-	-	-	-	-	-	-

表 2 各起動方法による RPC の起動時間 (秒)

	Fork				Condor				SGE			
	平均	分散	最小	最大	平均	分散	最小	最大	平均	分散	最小	最大
組込み GT2	0.95	0.00	0.92	1.00	15.18	8.37	6.69	19.66	9.37	8.30	4.00	14.53
Invoke Server GT2c	0.95	0.00	0.90	1.05	14.12	7.50	6.57	19.56	8.59	9.57	3.14	13.2
Invoke Server GT4py	2.03	0.00	1.97	2.64	18.75	0.09	7.61	19.58	4.48	0.23	4.06	6.55
Invoke Server GT4java	1.32	0.00	1.25	17.73	18.79	0.18	12.05	19.84	4.59	0.35	3.73	21.49
Invoke Server SSH	0.57	0.00	0.55	0.58	-	-	-	-	12.87	1.25	10.11	15.50
Invoke Server Condor	-	-	-	-	11.57	9.65	7.01	17.05	-	-	-	-

1 回に多く存在する外乱の影響をのぞくために，最初の一回の結果をのぞいて平均，分散を集計している．

縦軸に各 Invoke Server，横軸にサーバ側でのジョブ起動機構を示している．**組込み GT2** は，Ninf-G のクライアントライブラリ本体に埋め込まれた pre-WS GRAM 用のジョブ起動機構を利用し，Invoke Server を利用していない．**GT2c** は，Invoke Server のコンセプト実証用に作成された Invoke Server で，クライアントライブラリのコードを再利用して実装されている．この 2 つの内部ロジック，使用ライブラリは完全に同じであることから，これらを比較することで Invoke Server 機構そのもののオーバヘッドを見ることができる☆．

Invoke Server SSH はサーバ側での Condor ジョブ起動に対応していないため Condor の欄が空白となっている．同様に，Invoke Server Condor は Condor 以外のジョブ起動には対応していないため，SGE, Fork の欄が空白となる．

4.3 測定結果

組込み GT2 と Invoke Server GT2c はほぼ同じ値を示していることがわかる．いくつかの結果では GT2c のほうが高速であるが，これは測定誤差の範囲と考えられる．このことから，Invoke Server という形で，ジョブ起動機構をライブラリの外に出すことコストは，事実上問題にならないと考えられる．

GT4py と GT4java を比較すると，Fork の際に Java が有意に速いことがわかる．これは，GT4py は C 言語で記述された外部コマンドを起動するため，若干のオーバヘッドがあるためだと思われる．

GT2(pre-WS GRAM) と GT4(WS GRAM) の SGE での実行結果を見ると GT4(WS GRAM) がより高速である．これは，pre-WS GRAM と WS GRAM での SGE jobmanager の動作の相違によるものと考えられる．今後詳細を調査する予定である．また，全体では ssh と Fork によるジョブ起動がもっとも高速である．

SGE でも Condor でも，実行までにかかる時間が比較的安定していない．これは連続したサブミットによってシステムに高い負荷がかかることを防ぐための，スロットリングが行われているためだと思われる．

☆ Invoke Server 名の GT2, GT4 はそれぞれ，GT2 由来の pre-WS の GRAM と，GT4 から導入された WS GRAM を使用することを示している．計測にはいずれも GT4 を用いている．

5. より高度なモジュール化

現在の Ninf-G では，モジュールとして外部に出すことのできる機構は，ジョブ起動機構のみである．情報サービス，および通信機構としては Globus の機構である MDS2 と Globus-IO が，ライブラリとしてリンクされている．次のステップとしては，環境の要請に応じて，ジョブ起動機構だけでなく情報サービスおよび通信機構においてもさまざまなグリッドミドルウェアの提供する機能を選択可能にすることで，Ninf-G の提供範囲をひろげることが考えられる．これによってより多くの環境，アプリケーションに対して，効率的な実行を行うことが可能になる．

図 7 に高度にモジュール化した Ninf-G の模式図を示す．情報サービスに関しては，ジョブ起動サービス同様に，外部モジュールを設け，それを経由して情報の問い合わせを行う．通信機構に関しては，クライアント側とサーバ側にプロキシを置き，これを経由することで通信機構の相違を吸収する．クライアントとプロキシ，プロキシとリモート計算モジュールの間の通信は平文とし，通常のソケットライブラリを用いる．使用候補となる通信機構としては，Condor の chirp ライブラリが挙げられる．

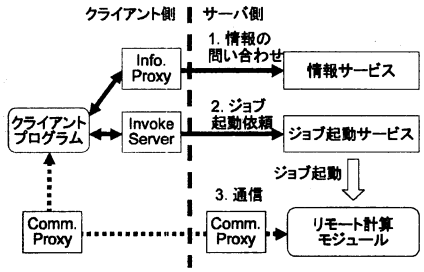

図 7 モジュール化した Ninf-G

この副次的な作用として，Ninf-G のクライアント，リモート計算モジュールと Globus Toolkit ライブラリのリンクが不要となる．現在の Ninf-G では，ジョブ起動機構として Globus Toolkit をまったく使用しない場合であっても，Globus Toolkit をインストールしリンクしなければ Ninf-G を使用することはできない．Globus Toolkit はさまざまなライブラリを含む巨大なパッケージであり，このインストールが不要になることで Ninf-G の普及促進が期待できる．

6. おわりに

本稿では Ninf-G に新たに導入されたいくつかの Invoke Server の実装を詳しく紹介し，起動時間を測定した．その結果 Invoke Server それ自身の機構によるオーバヘッドは無視できるほど小さいことがわかった．

比較的短時間でさまざまなシステムに対する実装が可能だったことは，Invoke Server のコンセプトとシンプルなプロトコルが有効だったことを示している．最近になって，Nordu Grid[13] の GridFTP を用いたジョブサブミッション機構に対する Invoke Server の実装が，第三者から提供されたことも，コンセプトの有用性を示す傍証となると思われる．

今後は，5 で述べた方針に基づき更なるモジュール化をすすめ，さまざまな環境に柔軟に対応することの可能なシステムを構築していく．

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) 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.
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) : Condor. http://www.cs.wisc.edu/condor/.
8) Nakada, H., Tanaka, Y., Matsuoka, S. and Sekiguchi, S.: "The Design and implementation of a Fault-Tolerant RPC system: Ninf-C", *"Proceedings of HPC ASIA 2004"*, pp. 9–18 (2004).
9) : Condor Java API, http://staff.aist.go.jp/hide-nakada/condor_java_api.
10) Chapman, C., Goonatilake, C., Emmerich, W., Farrellee, M., Tannenbaum, T., Livny, M., Calleja, M. and Dove, M.: Condor BirdBath: Web Service interfaces to Condor, *Proceedings of 2005 UK e-Science All Hands Meeting*, Nottingham, UK, pp. 737–744 (2005).
11) 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).
12) Matsuoka, S., Hatanaka, M., Nakano, Y., Iguchi, Y., Ohno, T., Saga, K. and Nakada, H.: Design and Implementation of NAREGI Super-Scheduler Based on the OGSA Architecture, Vol. 21, No. 4, pp. 521–528 (2006).
13) Eerola, P., Konya, B., Smirnova, O., Ekelof, T., Ellert, M., Hansen, J. R., Nielsen, J. L., Waananen, A., Konstantinov, A., Herrala, J., Tuisku, M., Myklebust, T., Ould-Saada, F. and Vinter, B.: The NorduGrid production Grid infrastructure, status and plans, *Proceedings of 4th International Workshop on Grid Computing (GRID'03), IEEE CS Press*, pp. 158–165.