Ninf-G2: 大規模環境での利用に即した 高機能，高性能 GridRPC システム

田中良夫[†] 中田秀基[†,††]
朝生正人[†††] 関口智嗣[†]

我々は大規模グリッド環境での利用に即した高機能かつ高性能な GridRPC システム Ninf-G2 の開発を進めている．数台のクラスタにより構成される 1000 プロセッサ規模のグリッド上で GridRPC を用いて大規模アプリケーションを効率良く実行するためには，リモートライブラリの呼び出しや引数情報の取得を効率良く行なうといった性能的な問題への対処と，アプリケーションの実行状態の監視や中間結果の視覚化，サーバ属性の個別指定などの機能的要求への対応が必要となる．本稿においては，Ninf-G2 の設計および実装に際して行なっている予備評価について報告する．

Ninf-G2: A High Performance GridRPC system for Large Scale Grid Environments

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

The GridRPC is a programming API proposed for a standard at GridRPC WG in the Global Grid Forum by which a large scale application program should be written for Grids. Although Ninf and Ninf-G, reference implementations of GridRPC, have been installed in various sites to accommodate large scale applications, from those preliminary experiences we have realized improvements in order to achieve higher performance and better usability of the systems. For example, we have reduced overheads in starting up remote procedures across networks and retrieving computed results from them. Also, we have introduced new features of monitoring of outstanding RPCs, visualization of intermediate status, and methods to describe server-dependent attributes. In this article, the design of Ninf-G2 is described with preliminary results of performance evaluation.

1. はじめに

GridRPC はグリッドにおける遠隔手続き呼び出し (Remote Procedure Call, RPC) によるプログラミングモデルのひとつである．グリッド技術の標準化をすすめる Global Grid Forum (GGF)[1] における GridRPC Working Group (GridRPC WG)[2] においては GridRPC API の標準化に関する議論が進められ[3]，GridRPC がグリッドにおける標準的なプログラミングモデルの 1 つとして広く利用される事を目指した活動も行なわれている．

遠隔手続き呼び出しを用いた広域分散コンピューティングに関する研究は，1994 年頃の Ninf プロジェクト[4] や NetSolve プロジェクト[5] などに始まる．当初は遠隔手続き呼び出しを用い，手元のコンピュータと遠隔地のスーパーコンピュータを組み合わせて大規模な科学技術計算を実行するという単純なクライアント／サーバ型の計算モデルが想定されており，例えば初期の Ninf プロジェクトにおいても，そのような利用に適した仕様および実装方法が研究されてきた．しかし，その後高性能計算のプラットフォームとしてクラスタシステムが急速に普及し，それに伴ってタスク並列なアプリケーションのようにマスター／ワーカー型の計算モデルが必要とされてきた．このため広域に分散配置された複数のクラスタ群上で実行するようなアプリケーションを実装する手段として GridRPC が用いられるようになってきた．

すでに Ninf-G[6]，NetSolve[7]，DIET[8]，OmniRPC[9] といった GridRPC を実装したシステム (GridRPC システム) の研究開発が行なわれており，細胞生理学向けのモンテカルロシミュレータ[10]，グリッド上の専用計算機と汎用並列計算機群とを利用した物理現

† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
†† 東京工業大学
Tokyo Institute of Technology
††† 株式会社 創夢
SOUM Corporation

象の複合シミュレーションシステム[11] や国際的なグリッドテストベッド上で行なった気象シミュレーション[12] などの GridRPC の実用例が報告され，GridRPC の性能評価や実用性の検証が行なわれている．しかしながら，より大規模な環境，例えば数サイト〜十数サイトに配置された数十〜数百プロセッサ程度のクラスタにより構成される総数 1000 プロセッサ規模のグリッドにおいて，アプリケーションを効率良く実行するためには，現状のシステムでは機能面，性能面において問題があることも報告されている[13]．

本稿においては，これらの問題に対応した GridRPC システム Ninf-G2 の設計および実装に際して行なっている予備評価について報告する．次節では，Ninf-G2 の設計に際して考慮した項目を示す．3節では Ninf-G2 の設計および実装のベースとなる Ninf-G のアーキテクチャを紹介し，4節および 5節では Ninf-G2 の設計，実装の方針および実装に向けて行なっている予備評価について述べ，最後に現状と今後の計画を述べる．

2. Ninf-G2 の設計方針

Ninf-G2 はグリッドに関する特別な知識を持たずとも GridRPC を利用してグリッド上で動作するアプリケーションを簡単に開発できる GridRPC システムである．Ninf-G2 を用いて開発したアプリケーションを，数サイト〜十数サイトに配置された数十〜数百プロセッサ程度のクラスタにより構成される総数 1000 プロセッサ規模のグリッド上で効率良く実行できる事を目標としている．GridRPC においては，遠隔手続き呼び出しを用いて手元のコンピュータ (**クライアント**) から遠隔地のコンピュータ (**サーバ**) 上の遠隔手続き呼び出し可能なライブラリ (**リモートライブラリ**) を呼び出すことにより，分散配置された複数の計算資源に対して関数単位で計算を振り分ける事により並列分散計算を実現することができる．GridRPC API は遠隔手続き呼び出しを行なうための API を提供しており，アプリケーション開発者は通常の関数呼び出しと同じ方法で遠隔手続き呼び出しを行なう事ができる．大規模環境での利用に即したシステムとするために，Ninf-G2 においてはリモートライブラリの呼び出しや引数情報の取得を効率良く行なうといった性能的な問題への対処と，アプリケーションの実行状態の監視や中間結果の視覚化，サーバにより異なる属性の個別指定などの機能的要求への対応が必要となる．本節では Ninf-G2 の設計に際して特に考慮した項目を示す．

大規模クラスタ利用におけるオーバーヘッドを抑える

数百〜数千プロセッサを利用して計算を行なう場合，計算の依頼およびその初期化にかかるオーバーヘッドをできる限り小さくする必要がある．初期化のオーバーヘッドには認証行為，計算ノードでのプロセス生成および引数情報の取得などが含まれる．

複数のサーバを利用する機能を充実させる

サーバごとにプロセスが利用するポート番号，利用可能なプロトコル，認証方法などの属性が異なる可能性がある．複数のサーバを利用してアプリケーションを実行する場合，それらサーバごとに異なる属性を指定する，およびその非均質性を隠蔽する機能が必要となる．

大規模アプリケーションに対応する

大規模なアプリケーションを長時間にわたって実行する場合，ハートビート，モニタリング，デバッグ，中間結果の可視化などの機能が必要となる．

データ送受信を効率化する

クライアントとサーバ間のネットワーク性能が低いグリッド環境においても効率良く動作するよう，クライアントとサーバ間での冗長なデータ送受信を省く機能の提供など，データ送受信のオーバーヘッドを削減する機能・実装が必要となる．

システムを肥大化させない

アプリケーションの実行には様々な機能が必要となるが，それらすべてをシステムに組み込むとシステム自体が肥大化し，結果として使いにくいものとなったり不安定さを招くことになる．スケジューリング，ブローカリングおよび障害からの復旧等の耐故障性といった機能は Ninf-G2 自体には組み込まず，他のモジュールとリンクしてそれらの機能を利用するようにする．障害が発生した場合にはエラーコードを適切に返すにとどめる．

3. Ninf-G のアーキテクチャ

本節では Ninf-G2 の設計および実装のベースとなる Ninf-G の概要を述べる．

Ninf-G は GGF GridRPC WG において議論が進められている「標準 GridRPC API」の参照実装であり，グリッドのミドルウェアとして広く利用されている Globus Toolkit[14] の API を利用して実装されている GridRPC システムである．Ninf-G Version 1.0 は 2002 年 11 月にリリースされ，その後約半年間で 600 を越えるダウンロードがあり，現在各国で利用されている．Ninf-G を用いることにより，プログラマは Globus Toolkit が提供する低レベルな API を用いた複雑なプログラミングを行なうことなく，ローカルな関数呼び出しと同様なインタフェースを用いた簡単なプログラミングを行なうことができ，同時にグリッドに関する特別な知識を持たずとも Globus Toolkit を介してグリッドの標準技術にのっとったグリッドアプリケーションを開発することができる．Ninf-G は Globus Toolkit の各コンポーネントを利用して，次のような動作を実現している (図 1)．なお，Globus Toolkit の各コンポーネントの詳細については文献[14] を参照されたい．

(1) リモートライブラリの呼び出し情報 (パス情報や引数情報など) は MDS を利用して登録・検索を行なう．

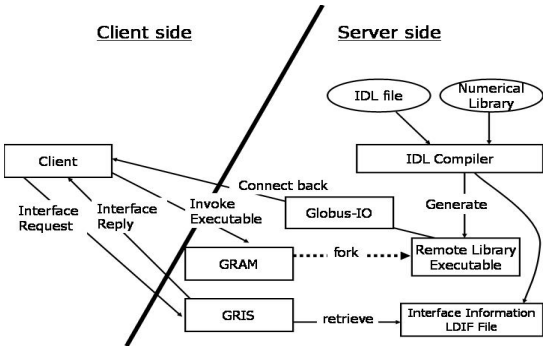

図 1 Ninf-G のアーキテクチャ

(2) リモートライブラリの起動は GRAM を介して行なう．これにより，GSI に基づく認証およびサーバ上でのキューイングシステムの利用が可能となる．

(3) クライアントとサーバ間のデータ送受信は Globus IO を利用する．

GridRPC API を用いてリモートライブラリを呼び出す場合には，まず *grpc_function_handle_init()* あるいは *grpc_function_handle_default()* を用いてリモートライブラリとの接続を抽象化する関数ハンドルを作成し，以後その関数ハンドルを用いて *grpc_call()* または *grpc_call_async()* による同期／非同期型リモートライブラリ呼び出しを行なう事になる．関数ハンドルの作成がリモートライブラリ呼び出しの初期化処理にあたり，ここで MDS の検索による情報取得および GRAM を介した認証およびプロセス生成が行なわれる．また，リモートライブラリの呼び出し時にはクライアントとサーバとの間で引数および返り値の送受信が行なわれ，これらがそれぞれオーバーヘッドとなる．Ninf-G2 の設計においてはこれらをどう削減するかが鍵となる．

4. Ninf-G2 の設計および実装

2節で挙げた設計方針をふまえ，Ninf-G を開発・利用した経験を元に Ninf-G2 の設計および実装を進めている．本節では，Ninf-G2 の仕様および実装について述べる．

4.1 大規模クラスタ利用におけるオーバヘッドの抑制への対応

複数の関数ハンドルを高速に作成する機能を実装し，そのための API を追加する

関数ハンドルの作成においては，GRAM を介したプロセス生成のオーバーヘッドが生じる．GRAM 呼び出しオーバーヘッドの主な原因としては，gatekeeper とクライアントとの間で行なわれる相互認証と，Job manager がキューイングシステムを介してプロセスを生成する際の (キューイングシステム側の) オーバーヘッドがある．キューイングシステムを介さずに fork を用いてプロセスを生成したとしても，GRAM 呼び出しには約 1 秒を要する．例えば 256 個のジョブプロセスを生成する場合，単純に起動すれば起動のオーバーヘッドだけで 4 分以上かかってしまうことになる．

そこで，Ninf-G2 では複数の関数ハンドルをまとめて生成する機能を導入し，そのための API を追加する．具体的には，関数ハンドルの配列を作成・破棄する API を提供する．複数の関数ハンドルをまとめて効率的に作成する実装方法については次節で述べる．

MDS を利用せずに引数情報を取得する機能の提供

リモートライブラリ呼び出しの初期化処理において関数ハンドルを作成する際には MDS を利用して引数情報を取得しているが，MDS の検索時間によるオーバーヘッドが大きい．MDS の検索時間はクライアントと MDS サーバ間のネットワーク性能および MDS に登録されているオブジェクト数やディレクトリツリーの構成に依存するが，実験によると数十秒程度かかってしまうことがある．また，Globus Toolkit の MDS は MDS のバージョンが 2.2 になってから多少改善されたものの依然として不安定であり，MDS サーバが常に安定して利用可能であるという事を期待できない．

そこで，Ninf-G2 では Ninf-G と同様な MDS を利用した引数情報の登録・取得機能に加え，MDS を使用しない方法も実装する．ユーザは利用環境に応じて適切な方法を選択できる．具体的には，Ninf-G2 では新たに以下の 2 つの方法を提供する．

(1) MDS に登録する際に用いられる LDIF ファイルをクライアントマシン上に置き，クライアントはこのファイルを参照する．LDIF ファイルはクライアントの実行時に引数として渡すクライアントコンフィグレーションファイル中で指定する．

(2) クライアントはリモートライブラリを起動し，リモートライブラリに情報を問い合わせる．リモートライブラリの起動に必要な情報 (実行パスなど) は関数リストファイル中で指定する．

4.2 複数のサーバを利用する機能の提供

Ninf-G2 ではサーバごとに属性を指定する方法およびサーバの非均質性に対する機能として，以下の 3 つを提供する．

クライアントコンフィグレーションファイル

クライアントプログラムの動作を指定するための様々な属性を記述したコンフィグレーションファイルを，プログラムの実行時に引数として与える．クライアントコンフィグレーションにおいては，クライアント，MDS サーバ，計算サーバの各項目ごとに，属性名と属性値を記述して属性を定義する．図 2にクライアントコンフィグレーションファイルの記述例を示す．

関数ハンドル作成時のタイムアウト指定

Globus Toolkit の Job manager がバックエンドで利用するキューイングシステムの設定やサーバの負荷の状態によっては，依頼した計算がいつになっても実行

```
# '#' 以降行末までをコメントとする.
<CLIENT>        # クライアントの情報を定義する (複数指定不可能)
    hostname            ホスト名          # クライアントのホスト名
    port                ポート番号        # クライアントのポート番号
    save_sessionInfo    セッション数      # 完了したセッションを幾つ保存するか
    loglevel            [0-5]             # ログレベル
</CLIENT>

<FUNCTION_FILE>       # 関数情報を定義する (複数指定不可能)
    ldif_file           FILENAME    # ローカル LDIF ファイル
    ldif_file           FILENAME    # ローカル LDIF ファイル
    function_list       FILENAME    # ファンクションリストファイル
    function_list       FILENAME    # ファンクションリストファイル
</FUNCTION_FILE>

<MDS_SERVER>        # MDS サーバの情報を定義する (複数指定可能)
    hostname    ホスト名        # MDS サーバのホスト名
    port        ポート番号      # MDS サーバのポート番号
    vo_name     VONAME          # GIIS の vo name
</MDS_SERVER>

<SERVER>        # リモートライブラリを実行するサーバの情報を定義する (複数指定可能)
    hostname            ホスト名              # サーバのホスト名
    mds_hostname        ホスト名              # MDS サーバのホスト名
    jobmanager          ジョブマネージャ      # Job manager
    protocol            [XML/binary]          # プロトコルの指定
    job_timeout         秒                    # Job のタイムアウト時間
    heartbeat           秒                    # ハートビートの間隔
    gass_scheme         [http/https]          # GASS サーバの scheme
    redirect_outerr     [true/false]          # stdout/err のリダイレクト
    debug_exe           [true/false]          # gdb によるデバッグ on/off
    debug_display       DISPLAY               # デバッグ用 xterm の表示先
    debug_terminal      コマンドパス名        # デバッグ用 xterm のパス
    debug_debugger      コマンドパス名        # デバッグ用 gdb のパス
</SERVER>
```

図 2 クライアントコンフィグレーションファイルの例

されず，最悪の場合プログラムのデッドロックを招いてしまう場合も考えられる．この問題に対応すべく，Ninf-G2 は関数ハンドルを作成する際にタイムアウトを指定する機能を提供する．タイムアウト値は，クライアントコンフィグレーションファイルの SERVER に定義される job_timeout 値を使用する．

サーバコンフィグレーションファイル

リモートライブラリの動作に必要な情報を変更可能とするため，サーバコンフィグレーションファイルを利用可能にする．サーバ側に置かれるサーバコンフィグレーションファイルとして， Ninf-G2 が提供するシステム属性ファイルと利用者が用意するユーザ定義ファイルの 2 種類を提供する．システム属性で定義される属性がデフォルトの属性として定義され，ユーザ定義ファイルによってデフォルト値を変更することができる．定義される属性としては，ファイル転送の際に一時的に使用するファイルの置場所などがある．

4.3 大規模アプリケーションへの対応

アプリケーションの実行が長時間にわたる場合，アプリケーションユーザに対してアプリケーションの実行状態をチェックしたり中間結果を表示するような機能を提供することが望ましい． Ninf-G2 では以下にあげる機能を提供する．

ハートビート機能

リモートライブラリの実行中は，リモートライブラリからクライアントに対して定期的にハートビートを発行することにより，リモートライブラリが動作していることをクライアントが確認する機能を提供する．クライアントコンフィグレーションファイルの SERVER に定義される heartbeat 値でハートビートの間隔を指定する．

また，クライアントがリモートライブラリの実行状態を確認するための API *grpc_get_info_np()* を提供する．

コールバック機能

サーバで行なわれている計算の途中結果の表示などはユーザからの要求が高い機能である．リモートライブラリの実行状態 (途中結果) をクライアント側で知るためには，クライアントとサーバとの間で情報を共有する仕組みが必要である． Ninf-G2 では，コールバック機能によりこの仕組みを実装する．コールバック機能は，リモートライブラリからクライアント上の関数を呼び出す機能であるり，計算の途中結果の表示の他に計算の一部のクライアントによる実行や対話的な処理に利用できる．図 3にコールバックを利用する IDL ファイルおよびクライアントプログラムの記述例を示す．

```
/* IDL の記述例 */
Module test;

Define callback_test(IN int a, OUT int *b,
  callback_func(IN int c[1], OUT int d[1]))
{
  int d;
  *b = a * 10;
  callback_func(b, &d);
}

/* クライアントプログラム例 */
void callback_func(int c[], int d[])
{
  d[0] = c[0] * c[0];
}

client()
{
  int b;
  grpc_function_handle_t handle;

  grpc_function_handle_default
    (&handle, "test/callback_test");
  grpc_call(&handle, 100, &b, callback_func);
}
```

図 3 コールバックを利用する IDL およびクライアントプログラム例

セッションキャンセル機能

クライアントプログラムがリモートライブラリを呼び出し，その実行終了を認識するまでの処理をセッションと呼ぶ． GridRPC API にはセッションをキャンセルするための API である *grpc_cancel(int session_id)* があり， Ninf-G もこの API は実装しているが，リモートライブラリのプロセスを Kill することによって実装している．リモートライブラリが単一のスレッドで動作しているため，リモートライブラリを実行している間にクライアントからのキャンセル要求を受け取る事ができず，リモートライブラリのプロセスを Kill するという実装になっている．

Ninf-G2 では前述のようにハートビート機能を実装するが，リモートライブラリからのハートビートの通知に対するクライアントからの応答に，セッションのキャンセル要求を付加できるようにする．リモートライブラリ側の API として， *ngstb_is_canceled()* を追加する．クライアント側で *grpc_cancel()* が実行された場合には，*ngstb_is_canceled()* は *GRPC_TRUE* を返す．キャンセル機能を実現したい場合には，リモートライブラリの関数内で定期的に *ngstb_is_canceled()* を呼び出し，クライアントからキャンセル要求が発行されていないかを

判断し，*ngstb_is_canceled()* が *GRPC_TRUE* を返した場合には計算を終了するようにリモートライブラリを実装しておく．

4.4 データ送受信の効率化

クライアントとサーバ間のデータ送受信の効率化については，次の 3 つの方法を Ninf-G2 に組み込む．

リモートオブジェクトの実装

リモートライブラリは状態を持たないため，リモートライブラリの呼び出しを行なうたびに引数データを送受信する必要がある．同じデータに対して何度か計算を行なうような場合でも毎回データを送信する必要があるが，リモートライブラリにデータを保持させる (状態を持たせる) 事ができれば，冗長な (2 回目以降の) データ送信を省くことができる． Ninf-G2 では，そのような状態を持つリモートライブラリをリモートオブジェクトとして提供し，リモートオブジェクト上に複数のメソッドを定義し，また，クライアント側がそれらのメソッドを呼び出す機能を提供する．

バイナリプロトコルの実装

Ninf-G ではクライアントとリモートライブラリとの間でデータを送受信する際には，XML 化した文字列として表現されたデータを交換している．可読性のある形式にすることによってデバッグを容易にするなどの利点があるが，XML のエンコード／デコードに時間がかかるという問題がある．

Ninf-G2 では XML 化しないバイナリデータの通信も可能とする．クライアントコンフィグレーションファイルの `SERVER` に定義される `protocol` 値で使用するプロトコルを指定する．

送信データの圧縮[☆]

データを送信する際に，必要に応じて送信データの圧縮を行なう機能を追加する．クライアントのコンフィグレーションファイルにおいて，データ圧縮を行なうかどうかを示すフラグ，圧縮を試みるデータサイズの閾値 (指定されたサイズ以上のデータに対して圧縮を試みる)，および圧縮結果に対して実際に圧縮転送を行なうかどうかを判断する閾値を指定し，実行時のデータ圧縮の動作を指定する．

5. 予 備 評 価

Ninf-G2 の実装に際し，より効率の良い実装方法を策定するためにいくつかの項目については予備評価を行なっている．本節では，その中から複数の関数ハンドルをまとめて作成する実装方法に関する予備評価の結果について述べる．

5.1 実 験 方 法

複数の関数ハンドルをまとめて効率良く生成する方法としては，クライアント側で複数のスレッドを起動して関数ハンドルをスレッド並列により生成する方法と，GRAM が提供する 1 度の GRAM 呼び出しで複数のジョブを起動する機能を利用する方法の 2 通りの実装方法が考えられる．前者の方法では関数ハンドルを別々の GRAM 呼び出しにより生成する事になるため，生成された関数ハンドルの個別制御が容易であるが，後者の方法では複数の関数ハンドルを 1 度の GRAM 呼び出しで生成することになるため，生成された関数ハンドルの個別制御が複雑になる．効率の良い実装方法を選択するために，それぞれの実装方法の性能調査を行なった．以下の 3 通りの方法で，32 個のリモートライブラリの起動に要する時間 (関数ハンドル作成を依頼してから，すべての関数ハンドルが作成され，リモートライブラリが起動されるまでの時間) を計測した．

実験 1 : 1 個のスレッドで *grpc_function_handle_init()* を 32 回繰り返す．

実験 2 : 32 個のスレッドを生成し，各スレッドで *grpc_function_handle_init()* を実行する．

実験 3 : 1 度の GRAM 呼び出しで複数のジョブを起動する機能を利用して 32 個のリモートライブラリを起動する．

実験は32 ノードの Linux クラスタで行なった．CPU は Dual Pentium III 1.4GHz，ノード間接続は Gigabit Ethernet である． Globus Toolkit のバージョンは 2.2.4，すべてのノードで gatekeeper が動作しており， fork Job manager がインストールされている．フロントエンドノードでは grd Job manager(Sun Grid Engine をバックエンドで利用するもの) もインストールされており， grd Job manager 経由でバックエンドノードにジョブを振り分けることもできる．各テストでは，フロントエンドノードの grd Job manager を利用して 32 個のリモートライブラリを起動する方法と， 32 台のノードそれぞれに対して直接リモートライブラリ起動を依頼する方法の 2 通りの方法で計測を行なった．grd Job manager はバックエンドにキューイングシステムを利用する Job manager の例として実験に利用した．キューイングシステムのコンフィグレーションやシステムの負荷状況によって動作が変わる可能性はあるが，同様な条件で実験を行なえば， PBS や LSF など他のキューイングシステムを利用した場合でも今回の実験結果と同様な性能が得られると考えられる．

5.2 計測結果および評価

各テストを 5 回実行した結果 (計測値と平均値) を表 1に示す．最も時間のかかったものから順番に記述している．結果より，以下のことが分かる．

- grd Job manager を使用した場合，一度の GRAM 呼び出しで複数の Job を起動する方法が最も速く動作している．
- fork Job manager を使用した場合，マルチスレッドにより関数ハンドルを生成する方法がもっとも早い．また， Non Thread で *grpc_function_handle_init()* を繰り返した場合が

[☆] 本機能については，実装するかどうかを検討中である

表 1 リモートライブラリの起動にかかる時間

	実験 1		実験 2		実験 3	
Jobmanager	grd	fork	grd	fork	grd	fork
最長	59	29	46	10	13	27
	55	29	41	8	13	27
	54	27	41	8	12	27
	52	27	40	8	12	26
最短	51	27	39	8	11	26
平均	54.2	27.8	41.4	8.4	12.2	26.6

単位は秒

最も時間がかかっている.

- 一度の GRAM 呼び出しでは, fork Job manager に比べて grd Job manager を使用したほうが約半分ほどの処理時間で済んでいる.
- fork Job manager を使用して全ての Job が ACTIVE になるまでの時間は, 一度の GRAM 呼び出しと Non Thread で *grpc_function_handle_init()* を繰り返した場合とではあまり変わらない.

クラスタの一般的なソフトウェア構成においては, フロントエンドノードで動作する Job manager を介してバックエンドノードでジョブを起動する方法が一般的であると考えられるため, Ninf-G2 の実装においては「一度の GRAM 呼び出しで複数のジョブを起動する」方法を採用する方向で検討している.

6. 現状と今後の計画

我々は GridRPC システム Ninf-G2 の開発を進めている. Ninf-G2 は GridRPC に基づくプログラミングミドルウェアであり, グリッドアプリケーションを容易に開発し, 開発したアプリケーションを大規模なグリッド環境上で効率良く動作させる事ができるようなシステムとすべく, 設計および実装を行なっている. 2003 年 11 月に米国で開かれる国際会議 Supercomputing にあわせて Ninf-G2 Version 0.9 を, 2003 年度末には Version 1.0 をリリースする予定である. その後実際に数百〜千プロセッサ程度のテストベッド上でアプリケーションを実行し, Ninf-G2 の機能および性能面での検証を進めていく予定である.

謝　　辞

日頃の議論を通じて有益なコメント等頂いております Ninf チーム諸氏に感謝致します. なお, 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) Catlett, C.: Standards for Grid Computing: Global Grid Forum, *Journal of Grid Computing*, Vol. 1, No. 1, pp. 3–7 (2003).
2) http://graal.ens-lyon.fr/GridRPC/.
3) Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C. and Casanova, H.: Overview of GridRPC: A Remote Procedure Call API for Grid Computing, *Proceedings of Grid Computing - Grid 2002*, pp. 274–278 (2002).
4) http://ninf.apgrid.org/.
5) http://icl.cs.utk.edu/netsolve/.
6) 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).
7) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Proceedings of Supercomputing '96* (1996).
8) Caron, E., Desprez, F., Lombard, F., Nicod, J.-M., Quinson, M. and Suter, F.: A Scalable Approach to Network Enabled Servers, *Proceedings of the 8th International EuroPar Conference*, Vol.2400 of Lecture Notes in Computer Science, Springer Verlag, pp. 907–910 (2002).
9) 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 GridRPC システム, 先端的計算基盤システムシンポジウム SACSIS 2003, Vol. 2003, No. 8, pp. 105–112 (2003).
10) Casanova, H., Bartol, T., Stiles, J. and Berman, F.: Distributing MCell Simulations on the Grid, *International Journal of Supercomputing Applicaionts*, Vol. 15, No. 3, pp. 243–257 (2001).
11) 朴泰祐, 佐藤三久, 小沼賢治, 牧野淳一郎, 須佐元, 高橋大介, 梅村雅之: HMCS-G: グリッド環境における計算宇宙物理のためのハイブリッド計算システム, 先端的計算基盤システムシンポジウム SACSIS 2003, Vol. 2003, No. 8, pp. 235–242 (2003).
12) Tanaka, Y., Takemiya, H., Shudo, K. and Sekiguchi, S.: Climate Simulation using Ninf-G on the ApGrid Testbed, *Grid Demo Workshop* (2003).
13) 武宮博, 首藤一幸, 田中良夫, 関口智嗣: Grid 環境上における気象予報シミュレーションシステムの構築, 先端的計算基盤システムシンポジウム SACSIS 2003, Vol. 2003, No. 8, pp. 251–258 (2003).
14) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, Vol. 11, No. 2, pp. 115–128 (1997).