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

2005－HPC－103 (12)
2005／8／3

GridRPCを用いたタスクファーミングAPIの設計と実装

谷村勇輔[†] 中田秀基[†,††]
田中良夫[†] 関口智嗣[†]

本研究では，グリッド上でタスク並列を行うアプリケーションを開発する手間を削減することを目指して，GridRPCの上位にタスクファーミングAPIを提供するミドルウェアを設計した．実アプリケーションの事例研究から得られた要求仕様をもとにAPIを提案し，タスクの自動割り当てや耐障害に関する機能をミドルウェア内部に実装した．そのような上位ミドルウェアを実装するために，Argument Array APIに引数データのコピー機能が求められること，ノンブロッキング呼び出しにおけるデータ通信のタイミングとRPCの実行情報を取得する方法がGridRPCで標準化される必要があることを明らかにした．

Implementation of A Task Farming API over GridRPC Framework

YUSUKE TANIMURA,[†] HIDEMOTO NAKADA,[†,††] YOSHIO TANAKA[†]
and SATOSHI SEKIGUCHI[†]

In this paper, a middleware which provides Task Farming API is studied over the GridRPC standard, in order to contribute effective development of task parallel applications for the grid. The proposed API and higher functionality in task scheduling and fault tolerance are implemented in the middleware, based on our past experiences with the Ninf-G. Through our study, it is revealed that the argument-array API needs to provide means to copy arguments for duplicated task assignment. Timing of data transfer in the non-blocking RPC and method to retrieve execution information of each RPC are expected to be standardized in the GridRPC.

1. はじめに

パラメータ探索やマスタースレーブで計算を行うアプリケーションは，分割して行う計算の粒度が大きければ，グリッドを利用して大規模な計算を短期間で終えることができる．GridRPC[1]は，そのようなグリッドのアプリケーションを容易に開発する枠組である．その主要なAPIセットであるエンドユーザAPIでは，リモートに用意されたライブラリ関数をローカルの関数のように呼び出すためのAPIが規定されている．これは計算機のアーキテクチャの違いや計算機間の通信方法をアプリケーションプログラマに意識させず，記述性が著しく向上したAPIセットであるといえる．

著者らは，GridRPCの参照実装としてNinf-G[2]の研究開発を進め，これまでに複数の科学計算アプリケーションを用いて実際的な評価を行ってきた[3),4)]．特に，TDDFT方程式の計算を長時間にわたりグリッドで実行する研究[4)]では，Ninf-Gの安定性とともに，耐障害性を考慮したアプリケーション開発のための留意点を示し，数百ノードからなるグリッド上でGridRPCアプリケーションを長時間実行できることを示した．その研究ではさらに，障害に強いアプリケーションを開発するために，より高位のミドルウェアに求められる機能も明らかにした．しかし，そもそもGridRPC APIの設計では，エンドユーザAPIはRPCをプリミティブに実行する機能に限っており，RPCの実行先の選択や耐障害に関する機能は，エンドユーザAPIの上位に実装されることが想定されている．それに対して，大規模で長時間実行を伴う実アプリケーションの開発においては，ユーザが望む計算を行うために上位機能を部分的にでも実装せざるを得ず，アプリケーション毎に実装がなされている状況である．

これらを踏まえて，本研究ではエンドユーザAPIを用いて共通化できるコンポーネントを作成し，その上位に，アプリケーション開発のコストを削減するTask Farming（以降，タスクファーミングと記す）APIを設計することにした．タスクファーミングとは，入力データやパラメータを変えながら同一プログラムを多数独立に実行することである．タスクファーミングAPIを用いることで，タスクファーミングが簡単に記述でき，さらに最大限に近い性能

† 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

や安定性が得られることが望ましい．現在，タスクファーミングAPIは，NetSolve[5]やNinfのプロジェクト[2]において試験的な実装がなされている．NetSolveでは，タスクファーミングAPIを利用することで逆に性能が低下する問題があったり，全タスクの終了を1度に待つ方法しかないというインタフェースの問題がある[6]．Ninfでは，メモリ容量の制約を受けにくい形のAPIが提案されている[7]が，Cプログラムに慣れていないアプリケーションユーザにとってコールバックを用いた記述が分かりにくいこと，タスクを実行する前にサーバを初期化する機能がほしいといった意見も出されている．

本研究では，それらのユーザの意見を取り入れてAPIの再設計を行い，かつセルフスケジューリングによるタスク割り当てや耐障害機能を追加する．そして，APIの検討および実装を通して，タスクファーミングのような上位ミドルウェアを実装するために，GridRPCで標準化が必要な項目を事例として明らかにすることを目的とする．本稿ではNinf-Gを用いた実装について述べるが，Ninf-Gに限らず，その他のGridRPCの実装にも有用な項目について議論を進めていく．

2. GridRPCとタスクファーミング

本章ではGridRPCとその現状，その上位に構築するタスクファーミングAPIに期待される機能を述べる．

2.1 GridRPCとその現状

GridRPCは，グリッドアプリケーションのプログラミングモデルの1つを規定する．GridRPCを利用したアプリケーションはメインプログラムとそのスタブからなる．メインプログラムの中でRPC（Remote Procedure Call）を発行するAPIを記述することで，計算の一部（スタブ）が遠隔の計算機で処理されるようになる．スタブである遠隔の計算プログラムは，RPCの発行以前，またはRPC実行時に遠隔の計算機にインストールする必要がある．GridRPCの特徴は，RPCを非同期に発行することで複数の計算機を利用してタスクを並列実行でき，パラメータ探索型やマスタースレーブ型のアルゴリズムをもつアプリケーションとの親和性が高いことである．

GridRPCのAPIはGGF（Global Grid Forum）[8]のワーキンググループにおいて議論が行われている．2005年6月の時点でエンドユーザAPI仕様が推奨ドキュメントとして最終段階にある．エンドユーザAPIはプリミティブなAPIとして設計され，スケジューリングやフォールトトレランスはアプリケーションレベル，あるいはGridRPCの上位レイヤで実装される設計となっている．一方，そのような上位レイヤとしては，本研究が着目するタスクファーミングや依存関係のあるタスクをまとめて実行するTask Sequencing[9]が考えられ，それらの実装を支援するミドルウェア向けAPIに関する議論も始まっている．特に，RPCの引数を操作するArgument Array APIと持続性のあるデータを扱うためのData Handle APIについて意見が交わされている．

2.2 タスクファーミングAPIへの要請

本節では，タスクファーミングAPIへの要請を以下の項目に分けて説明する．

上位ツールを意識した設計

タスクファーミングの利用事例としては，Matlabのようなツールから対話的に実行したり，スクリプトから実行したり，APIを使ってプログラムしたりすることが考えられる．GridRPCはCのAPIを規定しているので，まず，Cベースのタスクファーミング APIを考える．ただし，MCellのようなタスクファーミングを対話的に実行するツール[6]やパラメータを半自動的に生成するようなツールがC APIの上に実装されることを想定すべきである．

C APIによるタスクファーミングの記述性

GridRPCのエンドユーザAPIは，遠隔の計算機上で実行するプログラム内の関数をハンドルで抽象化し，ハンドルを明示的に指定してRPCを発行する仕様である．しかし，アプリケーションによっては，実行場所を意識せずにタスクを投入でき，ミドルウェア側で適切な実行場所が決定されて処理される方が望ましい場合もある．その場合，投入されたタスクはミドルウェア内部で監視し，一部あるいは全てのタスクが終了したらアプリケーションに通知する仕組みが必要となる．例えば，大型計算機に導入されているバッチシステムのように，タスクをqsubコマンドで投入するようなプログラミングは，アプリケーション・ユーザにとっては親しみがあり，グリッドのアプリケーションを抵抗なく開発することを可能にする．

ファーミングの実行前に，一様あるいは個々に与えられたパラメータを使って遠隔のプログラムの初期設定を行うファーミングの形も考えられる．これを実現するためには，前処理となる関数とパラメータを登録する手段，登録された処理を自動的に行う仕組みが必要である．

逆に，処理されたタスクに後処理を施すファーミングの形も考えられる．例えば，処理した順に結果をグラフィック表示させるアプリケーションが考えられる．これを実現するためには，全タスクの完了を待つだけでなく，1つずつタスクの完了を待つことのできるAPIが必要である．

タスクの自動割り当て機構

ミドルウェアには，計算機やネットワークの状態を監視しながら，効率よくタスクを割り当てる機能が望まれる．例えば，Ninf-Gでは，ユーザが指定した計算機上に遠隔プログラム（サーバ）が起動されるが，サーバ情報がプールで管理され，計算機の性能順に並び変えられ，ラウンドロビンでタスクが投入されるのが望ましい．また，実際に要したRPCの実行時間や障害の頻度をもとに，タスク割り当てを待つサーバの順序や割り当てられるサーバ数の上

限を決めるといった受動的なスケジューリングアルゴリズムの導入も考えられる．一方，メインプログラム（クライアント）が動く計算機のメモリやディスク容量などの資源を考慮して，タスク投入の API を一時的にブロックする機能も必要である．

耐障害機構

障害対策としては，障害により RPC が失敗してもメインプログラムの実行を継続し，失敗したタスクは別の計算機に再投入されることが望まれる．さらに，障害からの復旧の後，終了していた遠隔プログラムを再起動して，常に一定数の計算資源が確保できるような仕組みが期待される．そのほか，タスクスループットを向上させるために，障害の頻度を考慮してタスクの投入に冗長性を持たせたり，ユーザやシステム管理者向けに，障害の原因解明のヒントとなるログを残したりする機能があると有用である．

3. タスクファーミング API の設計と実装

前章を踏まえてタスクファーミング API を設計し，最新の Ninf-G を用いた実装の概要および API を示す．

3.1 設計の指針

- タスクファーミングの前処理として，起動した遠隔プログラムを個々に初期化する機能をサポートする．初期化メソッドは遠隔プログラムに実装され，クライアントのプログラムの中でそのメソッドとメソッドの実行（初期化）に必要な入力データを指定する．
- 遠隔プログラムの起動時に ID を付加して，その ID によってタスクの実行先を指定する機能をサポートする．投入先が指定された場合は，タスクを自動的に割り当てる機構は働かない．
- 待ちタスクと遠隔計算機の空き状況を監視しながら，自動的なタスク割り当て機能を提供する．ただし，メモリ不足に陥らないよう，割り当てレンジの最大値をユーザが指定できるようにする．
- タスクスループットが最大になるよう，各サーバの過去のタスク実行に要した時間をもとに割り当てレンジを調整する．
- 障害が発生した場合，影響のあった計算機には新たなタスクを割り当てず，失敗したタスクを自動的に別のサーバに再投入する機能を提供する．
- 障害が回復して計算機資源が再び利用できるようになった場合，遠隔プログラムを自動的に起動する機能を提供する．また，上述の初期化メソッドとその引数は自動的に保存され，遠隔プログラムの再起動時にも自動的に初期化が行われる機能を提供する．
- 未処理のタスク数が少なくなり計算資源が余っている場合に，タスクを冗長投入できる機能を提供する．

3.2 Ninf-G が提供する GridRPC API

タスクファーミング API の実装について述べる前に，実

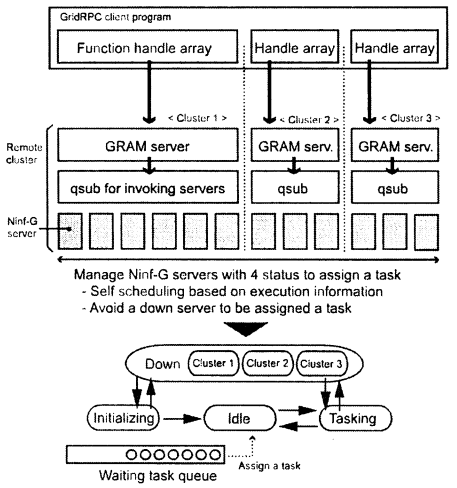

図 1 タスクファーミングの実装の概要

装に用いる Ninf-G について説明する．Ninf-G は産総研や東工大らによって開発が進められている GridRPC の参照実装の 1 つであり，2002 年 11 月に公開されたバージョン 1.0 以来，種々の科学計算アプリケーションを実装して，その性能やスケーラビリティ，安定性が改善されてきた[3),4)]．その経験の中，アプリケーションプログラマの要求に沿って，GridRPC で規定されていない拡張的な機能や API も Ninf-G では提供されている．

まず，Ninf-G ではその性能や安定性を最大限に引き出せるよう，非同期のデータ通信，圧縮通信，ハートビートといった項目を環境に即した形で設定ファイルに記述できる．通常はデフォルト値が設定されているが，安全側に設定されているため，最大性能を求めるユーザは設定ファイルの記述に注意が必要である．これらはタスクファーミングの内部実装で吸収されることが望ましいと考える．

次に，Ninf-G 独自の API が「_np」のサフィックスが加えられた形で実装されており，主に次の 3 つを挙げることができる．1）クラスタ毎に遠隔プログラムを一斉に起動するための機能，2）遠隔プログラムに内部状態をもたせて，RPC をまたいで変数の値を共有する機能（リモートオブジェクト機能），3）あらかじめ RPC の引数をスタックに積んで，スタックを指定して RPC を発行する機能である．このうち，1）は Globus を用いる Ninf-G に限定された機能であるが，2）3）については Ninf-G 以外の GridRPC システムにも関係し，各システムにおいて何らかの形で実装が検討されている機能である．

3.3 実　　装

前節を踏まえた実装の概要を図 1 に示す．Ninf-G は下位に Globus を利用しているので，遠隔プログラム，すなわち Ninf-G のサーバは Globus の Gatekeeper からロー

```
/* 引数配列の初期化 */
grpc_error_t grpc_arg_array_init(
                grpc_function_handle_t * handle,
                grpc_arg_array_t * array);

/* va_list から引数配列を初期化 */
grpc_error_t grpc_arg_array_init_with_va_list(
                grpc_function_handle_t * handle,
                grpc_arg_array_t * array,
                va_list list);

/* 引数の挿入 */
grpc_error_t grpc_arg_array_put(
                grpc_arg_array_t * array,
                int index,
                void * item);

/* 引数の取得 */
grpc_error_t grpc_arg_array_get(
                grpc_arg_array_t * array,
                int index,
                void ** item_ptr);

/* 引数配列を用いたノンブロッキング呼び出し */
grpc_error_t grpc_call_arg_array_async(
                grpc_function_handle_t * handle,
                grpc_sessionid_t * sessionId,
                grpc_arg_array_t * array);
```

図 2 主な Argument Array API

カルのバッチシステムを通して起動される．現状，Globus 2.x/3.x の性能と安定性を考えるとクラスタ毎に一括でサーバを起動せざるをえない．前節の 1）を用いてサーバを一括起動した場合，サーバのハンドルは配列として得られるが，内部処理としては，それをばらしてサーバの状態毎のプールに入れて管理する．起動前のサーバハンドルはクラスタ毎に設けられた Down プールに格納される．起動後，初期化メソッドを実行中のサーバハンドルはグローバルな Initializing プールに入れられる．初期化が終了したサーバハンドルはグローバルな Idle プールに格納され，さらにサーバに順位づけがなされる．順位づけは初期化メソッドやタスク実行の完了順に行い，さらにユーザのオプション次第で，全タスクの完了を待つ API が呼ばれた際に，直前の RPC の実行情報に基づいた順に並び替えられるようにする．タスクを実行中のサーバはグローバルな Tasking プールに格納される．一方，タスクはシングルのキューで管理され，アプリケーションから投入された順にタスク情報を格納し，Idle プールの最上位のサーバに割り当てられる．初期化によりリモートに作成された配列データは，前節の 2）の機能を用いてタスクの実行時に利用される．

障害により停止したサーバのハンドルはクラスタ毎の Down プールに戻される．Down プールに存在するサーバが一定の割合を越えたクラスタに対しては，定期的にサーバの再起動が試みられる．当然，再起動は同じハンドル配列で管理されるサーバの全てに対して適用される．

```
/* タスクファーミング API 利用のための初期化，終了 */
int grpcg_init(char *conf, sched_attr_t *sched,
               ft_attr_t *ft);
int grpcg_fin();

/* 遠隔プログラム（Ninf-G サーバ）の起動，停止 */
int grpcg_remote_init(int num_pe, char * func, ...);
int grpcg_remote_init_n(int server_id, int num_pe,
                        char * func, ...);
int grpcg_remote_fin(int num_pe);
int grpcg_remote_fin_n(int server_id, int num_pe);

/* タスクの投入 */
int grpcg_submit(char * func, ...);
int grpcg_submit_n(int server_id, char * func, ...);
int grpcg_submit_r(void * ref, char * func, ...);
int grpcg_submit_nr(int server_id, void * ref,
                    char * func, ...);

/* タスクの完了待ち，キャンセル */
int grpcg_wait_all();
int grpcg_wait_any(int * task_id, void ** ref);
int gprcg_cancel(int task_id);
```

図 3 タスクファーミング API

さらに，タスクファーミングのような上位 API の実装に際しては，エンドユーザ API とは別に，任意の引数をもつ API を提供する必要がある．そこで，GridRPC のワーキンググループで検討中の**図 2** の Argument Array API の利用を前提として実装を検討した．Argument Array API は前節の 3）を再設計した API である．この中には，va_list から RPC の引数配列を作成できる API が提供されているため，任意の引数をもつタスクファーミング API を提供し，ミドルウェア内部で必要な引数を取り出して GridRPC を実行することが可能になる．

3.4 API

図 3 に提案する API を示す．クライアントでは最初に grpcg_init() を呼び，利用したいクラスタのフロントエンドのホスト名，遠隔で起動するプログラムの実行ファイル名と起動したい数が記されたリストを conf で与える．sched にはタスクの割り当て戦略，ft には耐障害に関する戦略のパラメータを必要に応じて与える．次に，grpcg_remote_init() を用いて，conf で与えたリストに記されたクラスタにおいて必要な数（num_pe）だけ遠隔プログラムを起動する．この時，func に初期化メソッド名，以降に初期化メソッドを呼び出す際の引数をポインタで与える．これにより，遠隔プログラムの起動／復旧時に自動的に初期化メソッドが呼ばれるようになる．一方，grpcg_remote_init_n() を用いて遠隔プログラムを 1 つずつ起動することもできる．この場合，server_id に Int 型の値を指定することで，遠隔プログラムに任意の ID 番号を付与することができる．その ID を指定することで，特定のタスクを特定の遠隔プログラムに担当させることが可能になる．それぞれの遠隔プログラ

```
  :
rc = grpcg_init("servers.list", &sched, NULL)
if(rc != GRPCX_OK){
  perror("grpcg_init() failed.");
  exit(1);
}
grpcg_remote_init(NUM_PES, NULL);

for(i=0; i<numTask; i++){
  grpcg_submit("SP.S", "SP", &npbClass, &ngbClass,
               &ascii, "ED", &i, &width, &depth,
               &pid, &verbose, &filter,
               &numInput, &numOutput, &ngbBin,
               &report[i]);
}
rc = grpcg_wait_all();

grpcg_remote_fin(NUM_PES);
grpcg_fin();
  :
```

図 4 タスクファーミング API の利用例

ムに対して，異なる初期化メソッドと引数データを登録することも可能である．

タスクの投入方法として 4 種類の API を提供する．grpcg_submit() は計算メソッドとその引数データだけを指定し，タスクの実行場所はミドルウェアに任せる投入方法である．タスクの実行場所を ID を使って指定する場合には，grpcg_submit_n() を用いる．タスクに何らかのリファレンスポインタを与えて，タスクが完了した後にそのポインタを使って特定の処理を行うようなプログラムを書く場合は，grpcg_submit_r() を用いて，ref にリファレンスを指定する．このリファレンスは，任意のタスクの完了を待つ grpcg_wait_any() によって取得可能である．タスクの完了待ちは，全てのタスクの完了を待つ場合は grpcg_wait_all() を利用し，ある特定のタスクが完了するのを待つ場合は grpcg_wait_any() を利用する．RPC の実行時間が短いサーバからタスクが割り当てられるように，サーバの順序を変更する機能が有効になるのは，grpcg_wait_all() を用いた場合のみである．

タスクファーミングを終了してプログラムを終える場合は，grpcg_remote_fin() を呼んで遠隔プログラムを終了し，grpcg_fin() によってファーミングを行うミドルウェアの終了処理を行う．

3.5 API の利用例

提案するタスクファーミング API の利用例を図 4 に示す．これは NAS Grid Benchmark の EG で定義されるベンチマーク[10] の実装であり，S クラスの SP を 1 つのタスクとして numTask 数だけ投入し，全タスクの完了を待つプログラムである．エンドユーザ API を用いたプログラムに比べると，タスクファーミング API の利用によって，タスクの実行先の資源管理やエラー処理などの繁雑な分岐処理が隠蔽され，ファーミングのパラメータ設定のプログラミングに専念しやすい単純な記述になる．本例では，タスク投入の際にタスク番号を「&i」で与えて SP の初期条件とするループを記述している．EG では初期化メソッドを登録していないが，遠隔プログラムの起動時に初期化メソッドを登録・実行することも可能である．

提案 API は，初期化以外に処理の依存関係がなく，逐次処理されるタスクのファーミングへの適用が考えられる．ただし，分枝限定法においてタスク投入とは別に枝刈り情報をサーバに送ったり，サーバが動作するクラスタを意識してタスクを投入したり，データの転送やタスクの実行先を明示的に指定する場合には適用が難しい．そのような場合はエンドユーザ API を利用するのが良いと思われる．

4. 実装における課題

本章では，タスクファーミング API の設計および実装を通して，GridRPC の標準化を進めていくにあたり議論されるべき課題と解決案を述べる．

4.1 実装に用いた Ninf-G の拡張機能

本実装に用いた Ninf-G の独自 API は，Argument Array API に関係する部分を除くと図 5 の通りである．まず，タスク実行の前処理機能の実装には，クライアントから渡されたパラメータを使って処理されたデータをサーバ内部に保存し，連続する RPC 間でデータを共有する必要があった．これには Ninf-G のリモートオブジェクト機能を利用した．

次に，クラスタ毎に一斉にサーバを起動する機能を利用した．先述のように Globus の性能を考えると，数百のサーバを起動するためには一括起動の仕組みが必要であった．

非同期 RPC の呼び出しにおいては，データ転送をノンブロッキングで行う Ninf-G の「transfer_argument = nowait」オプションを用いた．これは非同期 RPC が呼ばれた際にデータの転送を待たずに関数が返るオプションであるが，そもそもいつ関数から返るかは GridRPC のエンドユーザ API では実装依存とされている．例えば，NetSolve ではデータ転送が完了してから関数が返るように実装されている．しかし，NetSolve のタスクファーミングはそのような非同期 RPC の上に作られているため，並列実行の効率が低いという問題がある．今回の検討を通して，タスクファーミングのようなミドルウェア内部で，RPC の引数データが管理され，適切にブロッキング／ノンブロッキングのデータ転送が発行されることが望ましいと考える．

タスク割り当てのためのサーバの順位づけには，過去の 1 回分の RPC の実行情報を利用した．その実行情報の取得のために，Ninf-G が提供する grpc_get_info_np() を用いた．これは，RPC の発行時に割り当てられるセッション ID により，Ninf-G が測定した入力および出力のデータ転送時間，リモートでの処理時間を取得する関数である．

これらの Ninf-G の拡張機能はタスクファーミングの

```
/* リモートオブジェクト機能をもつサーバの一括起動・終了 */
grpc_object_handle_array_init_np();
grpc_object_handle_array_destruct_np();

/* RPC の実行時間・データ転送時間の情報取得 */
grpc_get_info_np();

/* エラー出力 */
grpc_perror_np();
```

図 5 実装に用いた拡張 API

実装に欠かせないものであった．リモートオブジェクトは GridRPC で議論されているデータハンドルと関連し，サーバの一括起動は Globus に依存した話であるため，ここではこれ以上議論しない．一方，データの非同期転送，および RPC の実行情報の提供に関しては，複数の上位ミドルウェアで共通に使える機能であるため，GridRPC の標準化で規定されるべきと考える．

4.2 Argument Array API への要請

本研究では，Argument Array API の利用を前提としてタスクファーミング API の実装を検討した．しかしながら，現在の Argument Array API では RPC の引数のポインタを格納するだけであり，引数を内部にコピーするわけではない．それは以下の 2 つの点で不都合がある．

1 点目は，タスクファーミング API を利用するユーザにデータ領域を書き換えないことを常に求める点である．エンドユーザ API では，RPC のセッションをユーザ自身が管理する仕様であるため，データ領域の変更を意識的に行うことができるが，タスクファーミング API ではセッションを隠蔽してしまうため，この部分が API の外に現れるのは適切でないと考えられる．

2 点目は，タスクを複製して，複数の計算機に冗長投入を行う機能が実装しにくい点である．もし引数のコピーが可能であれば，内部的にタスクを複製し，RPC の実行結果をリモートから個々に受信し，grpcg_wait*() から返る際に，どちらか一方のデータをアプリケーションプログラムに返すような実装が可能になる．あるいは，RPC の実行結果の受信をロックできる別の API を GridRPC API 仕様に加える必要がある．

以上より，Argument Array API として，引数のポインタを保存しておくだけでなく，引数データ自身を内部にコピーできる API が追加されることが望ましいと考える．当然ながら，大量のタスクを投入する場合，クライアントを実行する計算機は大容量のメモリを持つことが要求され，メモリが不足することも少なくないであろう．しかし，同時に投入できるタスクを制限する機能を設けることは可能であり，かつ，大きなデータを必要とするアプリケーションはメモリに展開せず，ファイルとして入出力データを渡すケースも考えられるため，冗長投入が実装できることのメリットは十分にあると考える．

5. まとめと今後の課題

本研究では，GridRPC の上位ミドルウェアとしてタスクファーミング API の設計を行い，エンドユーザ API と Ninf-G 独自の機能や Argument Array API を用いて実装を検討した．タスクファーミングに要求される機能を実現するために，実行情報の取得機能やデータ転送のオプションが GridRPC 標準で規定される必要があること，GridRPC のミドルウェア API として，RPC の引数を扱う Argument Array API に求められる機能を明らかにした．今後は GridRPC の標準化の進捗に合わせて，フル機能のタスクファーミングを実装して実アプリケーションによる評価とユーザからのフィードバックを得ていきたい．

謝辞 本研究を行うにあたり，貴重なご意見を頂いた武宮（産総研）氏，池上氏（産総研）に深く感謝いたします．

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) Seymour, K. and et al.: Overview of GridRPC: A Remote Procedure Call API for Grid Computing, *Proceedings of 3rd International Workshop on Grid Computing* (Parashar, M.(ed.)), pp.274–278 (2002).
2) Ninf project: http://ninf.apgrid.org/.
3) 武宮博, 田中良夫, 中田秀基, 関口智嗣: Ninf-G2：大規模 Grid 環境での利用に即した高機能，高性能 GridRPC システムの実装と評価，情報処理学会論文誌 コンピューティングシステム (2004).
4) 谷村勇輔, 池上努, 中田秀基, 田中良夫, 関口智嗣: 耐障害性を考慮した Ninf-G アプリケーションの実装と評価，情報処理学会論文誌：コンピューティングシステム，Vol. 46, No. SIG 7 (ACS 10), pp. 18–27 (2005).
5) Arnold, D. and et.al.: Users' Guide to NetSolve V1.4.1, Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee (2002).
6) Casanova, H., Kim, M., Plank, J. S. and Dongarra, J. J.: Adaptive Scheduling for Task Farming with Grid Middleware, *High Performance Computing Applications*, Vol. 13, No. 3, pp. 231–240 (1999).
7) 中田秀基, 田中良夫, 松岡聡, 関口智嗣: GridRPC を用いたタスクファーミング API の試作，情報処理学会研究報告，Vol. 2003, No. 102, pp. 61–66 (2003).
8) GGF: http://www.gridforum.org/.
9) Arnold, D., Bechmann, D. and Dongarra, J.: Request Sequencing: Optimizing Communication for the Grid, *Lecture Notes in Computer Science: Proceedings of 6th International Euro-Par Conference*, Vol. 1900, pp. 1213–1222 (2000).
10) Rob F. Van der Wijngaart and Frumkin, M.: NAS Grid Benchmarks Version 1.0, Technical Report NAS-02-005, NASA Ames Research Center (2002).