GridRPC における複数ノードにまたがる Task Sequencing の実現

谷村勇輔[†] 中田秀基[†,††]
田中良夫[†] 関口智嗣[†]

GridRPC において，依存関係をもつ 2 つ以上の RPC，すなわち直前の RPC の出力が次の RPC の入力データとなる Task Sequencing ジョブを複数ノードにまたがって処理する場合に，クライアントを介さずに RPC の実行ノード間で直接データを転送する機能の設計と実装を行った．大域的な名前空間を提供するグローバルファイルシステムを利用することにより，クライアント・サーバモデルを基本とする GridRPC の特徴を損なわず，かつ既存の GridRPC システムの実装を大きく変えることなく実現した．また，連続する RPC の引数列を解析して中間データを自動判別し，中間データであればそれをグローバルファイルシステム上に作成する機能を Task Sequencing API ライブラリ内部に実装し，LAN 環境，日米間にまたがるグリッド環境における性能評価を行い，本機能が有効であることを明らかにした．

Design and Implementation of Distributed Task Sequencing on GridRPC

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

In the framework of GridRPC, a new function that allows direct data transfer between RPC servers is implemented for efficient execution of a Task Sequencing job on the grid environment. In the Task Sequencing, RPC has dependency between input and output parameters, which means output of previous RPC becomes input of next RPC. In this study, the direct transfer is implemented using the global filesystem neither to destroy the GridRPC programming model nor to change many parts of the existing Ninf-G implementation. Our designed Task Sequencing API library analyzes RPC arguments to detect the intermediate data after task submissions, and tells the information to GridRPC servers so that the intermediate data is created on the global filesystem. Through our performance evaluation on LAN and Japan-US grid environment, the function achieved performance improvement in distributed Task Sequencing.

1. はじめに

GridRPC[1] は，様々な資源が動的に組み合わさって構成されるグリッド環境に適応したアプリケーションを開発するためのプログラミングモデルの 1 つである．GridRPC はクライアント・サーバを基本として，負荷の高い計算を遠隔の計算機で並列に実行するための枠組を提供し，アプリケーション開発者は計算機のアーキテクチャの違いやデータ転送の方法を考慮することなく，ローカルのライブラリ関数を呼ぶように，遠隔に用意されたライブラリを利用することができる．

GridRPC の研究は，システムの設計から応用事例の研究段階に入り，科学的な成果が得られることを目指して性能や安定性が検証されているが[2),3)]，アプリケーション開発者からは，1 つのライブラリ関数だけを並列実行するのではなく，複数の関数の実行を 1 つのセットとして，1 セットを並列実行する要求が高いことが明らかになっている．セット内の一連の関数の実行においては，計算に必要なデータ配列が次の関数の実行に引き継がれ，逐次的に RPC が実行されることから，本論文ではこれを Task Sequencing と呼ぶ．Task Sequencing は，GridRPC を実装する Ninf-G[4)] や，GridRPC に準じた API を提供する NetSolve[5)] や OmniRPC[6)] において個別の追加機能としてサポートされている．これらのアプローチは，一連の RPC の実行先が最初から最後まで同一の計算機で行われる場合を仮定し，実行先の計算機のメモリやストレージにデータ配列を保存することで，クライアントとサーバ間の不要なデータ転送を削減している．

一方，Task Sequencing 中の各関数を異なる計算機上で実行するワークフロー的な実行モデルはグリッドの利用方法として高い要求がある[7)]．例えば，数値シミュレーショ

† 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

ンを大規模な並列計算機で実行し，その計算結果を専用の画像処理サーバで処理する状況が考えられる．しかし，現在 GridRPC システムが提供している Task Sequencing はこのような複数のノードにまたがる Task Sequencing に対応していない．また，クライアント・サーバに基づく簡単なプログラミングインタフェースの提供を特徴とする GridRPC においては，サーバ同士の直接の通信を指定するインタフェースは提供されていない．そのため，複数ノードにまたがる Task Sequencing を単純に実装する場合，サーバ間のデータ転送はクライアントを介す事になり，性能の大幅な低下を招いてしまう．

本研究では，クライアントを介さないサーバ間のデータ転送を実現し，Task Sequencing を要求するジョブ実行の高速化を図る．さらに，Task Sequencing ジョブを記述するための API を GridRPC の上位ライブラリとして構築するための検討を行い，現在の GridRPC の枠組における実現可能性と将来的な課題を明らかにする．

実装に際しては大域的な名前空間を提供するグローバルファイルシステムを利用することにより，クライアント・サーバモデルを基本とする GridRPC の特徴を損なわず，かつ既存の GridRPC システムの実装を大きく変えることなく計算ノード間の直接データ転送を実現した．具体的にはグローバルファイルシステムとしてグリッド・データファーム（Gfarm）[8] を用い，GridRPC の実装としては Ninf-G を用いた．第 1 段階としては，ファイル型の RPC 引数に Gfarm ファイルシステム上のファイルを指定できるように Ninf-G を拡張した．第 2 段階として，Task Sequencing API ライブラリを Ninf-G 上に実装し，ファイル型引数の入出力モードをライブラリ内で解析し，要求された Task Sequencing に沿って一時ファイルを Gfarm ストレージに作成できるようにした．

本論文では 2 章で関連研究，3 章にて Gfarm を用いた Ninf-G のファイル転送機能の拡張，4 章にて Task Sequencing API の設計と実装，5 章にて性能評価，6 章にてファイル型以外のデータを扱うための実装の指針について述べ，最後に結論を述べる．

2. 関 連 研 究

Task Sequencing ジョブを効率的に実行するためのアプローチとしては，次に述べるような研究が挙げられる．

GridRPC システムの先駆けである Ninf[9] や NetSolve では，データ配列を明示的にストレージサーバに格納する API が実装されている．例えば，NetSolve の DSI（Distributed Storage Infrastructure）[10] では，ns_dsi_write_matrix() を用いてデータ配列をストレージサーバに格納し，アクセスするためのハンドルとして NS_DSI_OBJECT を取得，それを RPC の引数に指定すると，ストレージサーバから実行先にデータが直接転送される仕組みである．DSI は IBP（Internet Backplane Protocol）[11] の上に構築されている．

NetSolve の Request Sequence API[12] は，RPC を実行する関数を begin 関数と end 関数で括り，end 関数の引数に最終的にクライアントに戻したい引数のポインタを与える仕様になっている．NetSolve のライブラリ内部で入出力モードの解析が行われ，不要なデータをクライアントに戻さないようにするためである．

Ninf-G や OmniRPC では，リモートオブジェクトとして遠隔のプログラムを起動し，プログラムが終了するまでデータ配列をメモリ上に置いておくことができる．すなわち，以前の RPC が参照していたデータ配列を，後の RPC の実行においても参照することができ，クライアントとサーバ間の不要な転送を削減することができる．

OmniRPC では，大規模な初期データを RPC の実行先に効率良く配布するために，データ転送機構を RPC の機構から分離した OmniStorage を提案している[13]．特に，遠隔の計算機がクラスタであった場合に，クラスタのフロントエンド上で動作するプログラムがクライアントとサーバ間の転送をキャッシュしながら中継し，ツリー型のデータのブロードキャストを実現している．

DIET[14] は，遠隔でのデータの永続性を扱うために Data Management API を提案している．取り扱うデータの Handle を作成する際に，データを実行先に残すのか，クライアントに戻すのか，他のサーバに移動可能かをパラメータで指定することができる．Data Tree Manager（DTM）を DIET システムとして実装し，計算ノード間のデータ転送もサポートする．

本研究では，上記の研究で議論された API を継承しつつ，計算ノード間の直接データ転送を実現している．DIET のデータ転送と異なる点は，データ転送のレイヤにグローバルなファイルシステム（Gfarm ファイルシステム）を利用しているため実装が単純になり，Gfarm が提供するデータの複製機能を利用して障害対策も行える点である．

3. Ninf-G のファイル転送機能の拡張

計算ノード間の直接データ転送の実現には，ネットワーク越しにアクセス可能なファイルパスを計算ノードに通知し，計算ノードがそれぞれクライアント・サーバの役割を果たす必要がある．Ninf-G に実装されている Globus の GASS[15] を用いたファイル転送機能は，ファイルの所在を計算ノード上の遠隔プログラムに通知して，計算ノードにおいて実際の転送ルーチンが開始される仕組みを実装しているが，ファイルの所在はクライアントを想定している．一方，グローバルファイルシステムを利用できる環境であれば，ファイルの所在をノードに依らない大域的なパスとして遠隔プログラムに通知するだけで，既存の Ninf-G を大きく変更することなく，計算ノード間の直接データ転送

```
Module sample
Define func1(IN filename input, OUT filename result)
Required "sample.o"
Calls "C" func1(input, result);
```

図 1 IDL ファイルの記述例
Fig. 1 An example IDL

が可能になる．本研究ではグローバルファイルシステムとして Gfarm を利用し，Gfarm ファイルシステム上のファイルを RPC の引数，RPC 実行先の一時ファイルとして指定できるように Ninf-G のファイル転送機能を拡張した．

3.1 Gfarm の概要

Gfarm は，広域に分散するストレージを統合し，実際にファイルが格納される場所とは無関係に仮想的な階層ディレクトリとしてアクセス可能な広域仮想ファイルシステムの実現を目指したシステムである．Gfarm ファイルシステムの特徴は，大域的な名前空間が提供されていることであり，例えば「gfarm:test.dat」のようなパスでネットワーク上のどのノードからでも test.dat ファイルにアクセスできる．Gfarm はファイルを Gfarm ストレージに格納したり，その複製を作成したりするなどの API を提供しているほか，アクセス用のコマンド群,「/gfarm」にマウントしているように Gfarm ファイルシステムにアクセスすることを可能にするシステムコールフックライブラリを提供している．本研究では 2006 年 1 月の時点でテスト版である Gfarm Ver.1.2.9 を用いた．

3.2 設 計 方 針

本研究では，既存の Ninf-G の変更を最小限に抑えたまま，Ninf-G のファイル転送において Gfarm ファイルシステム上のファイルを取り扱えるように設計する．従来，Ninf-G のファイル型データの転送は**図 1** に示す IDL，**図 2** に示す遠隔プログラムを記述し，クライアントプログラムでは**図 3** の [Original description] に示すように RPC の実行を記述する．クライアントから計算ノードに転送されたファイルは，遠隔プログラムの設定ファイルにおいて指定されたディレクトリ下に一時ファイルとして保管され，遠隔プログラムからアクセスできるようになる．一時ファイルの名前は自動的に付けられ，遠隔プログラムの関数の引数に設定される．

Gfarm を用いた拡張では，クライアントプログラムにおいて，GridRPC の呼び出し関数のファイル型データの値に Gfarm のパスを記述できるようにする．ただし，IDL に記述する引数のデータ型は，Gfarm ファイルシステム上のファイルを指定する場合でもファイル型（filename）のままとし，遠隔プログラムの修正が必要ないものとする．Ninf-G の Ver.2.3，2.4 との互換性が保たれるように Ninf-G のプロトコルも変更しない．

本設計では，計算ノードで作成される一時ファイルにのみファイルの有効期限を適用する．従来通り，一時ファイルがリモートオブジェクトの共有データとして設定されていなければ，RPC の終了とともに消去される．これは一時ファイルが Gfarm ファイルシステム上に作成される場合でも同様とする．一時ファイルでなく，計算結果として Gfarm 上に出力されたファイルは，アプリケーションや Task Sequencing の上位ライブラリによって有効期限が取り扱われる仕様とする．

```
void func1(char *input, char *result){
  FILE fp1, fp2;
  fp1 = fopen(input, "r");
  fp2 = fopen(result, "w");

  /* Do work */

  fclose(fp1);
  fclose(fp2);
}
```

図 2 遠隔プログラムの記述例
Fig. 2 An example remote program

```
[Original description]
    :
grpc_function_handle_init(&handle, "func1", "host1");
grpc_call(&handle, "exp1/input.dat", "result.dat");
    :

[Extended description for access to the Gfarm filesystem]
    :
grpc_function_handle_init(&handle, "func1", "host1");
grpc_call(&handle,
          "GFS:LTMP:GFILE:exp1/input.dat", "result.dat");
    :
```

図 3 クライアントの記述例
Fig. 3 Example client code

ファイルのキャッシュは，Gfarm が提供するファイルの複製機能を用いて扱う．遠隔プログラムが Gfarm ファイルシステム上のファイルを読み込む際，遠隔プログラムが動作する計算ノードが Gfarm のストレージノードである場合に，複製を計算ノードに作成してからアクセスを行う．ファイルを更新すると，複製元のファイルが Gfarm によって自動的に消去される．

Gfarm を用いたファイル転送は，計算ノード間の直接データ転送の実現を目的としているため，Gfarm ファイルシステム上のファイルを扱うための一連の操作は Ninf-G の遠隔ライブラリにて実装する．このため，Ninf-G の遠隔プログラムをコンパイルするノードにおいて，Gfarm のクライアントライブラリ，およびシステムコールフックライブラリがインストールされている必要がある．

3.3 実　　装

従来のファイル転送のパターンに加えて，Gfarm ファイルシステムを用いたファイル転送のパターンを入力ファイル，一時ファイル（遠隔に置かれるファイル），出力ファ

表 1 ファイル転送のパターン
Table 1 File transfer patterns

No.	Input	Temporary	Output	Implementation
1	–	Local	–	No transfer
2	–	Local	Local	Implemented
3	–	Local	GFS	Case 2
4	Local	Local	–	Implemented
5	Local	Local	Local	Implemented
6	Local	Local	GFS	Case 2
7	GFS	Local	–	Case 1
8	GFS	Local	Local	Case 1
9	GFS	Local	GFS	Case 1, 2
10	–	GFS	–	No transfer
11	–	GFS	Local	Case 4
12	–	GFS	GFS	Case 6
13	Local	GFS	–	Case 3
14	Local	GFS	Local	Case 3, 4
15	Local	GFS	GFS	Case 3, 5
16	GFS	GFS	–	Case 5
17	GFS	GFS	Local	Case 4, 5
18	GFS	GFS	GFS	Case 5, 6

イルの格納場所に分けて示したのが**表 1** である．表 1 に示すように 6 つの入出力のケースを新しく実装すれば，残りはケースを組み合わせることで全てのファイル転送パターンを網羅できる．以下に，入出力の 6 ケースの実装について説明する．

- ケース 1（Input/GFS → Temporary/Remote）
  Ninf-G の遠隔ライブラリにおいて，クライアントから指定された Gfarm ファイルシステム上のファイル（以降，GFS ファイルと呼ぶ）を Gfarm の C API を用いて開き，計算ノードの通常のファイルシステム上のファイル（以降，ローカルファイルと呼ぶ）に中身をコピーする．
- ケース 2（Temporary/Remote → Output/GFS）
  Ninf-G の遠隔ライブラリにおいて，クライアントから指定された GFS ファイルを開き，計算ノード上のローカルファイルの中身をコピーする．Gfarm のストレージノード選択メカニズムにより，計算ノードが Gfarm のストレージノードであれば，GFS ファイルは同じノードの Gfarm ファイルシステム領域に作成される．
- ケース 3（Input/Client → Temporary/GFS）
  Ninf-G Ver.2.3 が実装する GASS を用いたファイル転送により，クライアント上のローカルファイルの中身を一時ファイルとして作成した GFS ファイルにコピーする．GFS ファイルの所在は，ケース 2 と同様に Gfarm のストレージ選択メカニズムに従う．
- ケース 4（Temporary/GFS → Output/Client）
  Ninf-G Ver.2.3 が実装する GASS を用いたファイル転送により，一時ファイルとして作成した GFS ファイルの中身をクライアント上のローカルファイルにコピーする．GFS ファイルの所在は，ケース 2 と同様に Gfarm のストレージ選択メカニズムに従っている．
- ケース 5（Input/GFS → Temporary/GFS）
  転送元，転送先が同一となるため，ファイル転送を実装する必要がない．ただし，計算ノードが Gfarm のストレージノードであれば，GFS ファイルの複製を計算ノードの Gfarm ファイルシステム領域に作成する．
- ケース 6（Temporary/GFS → Output/GFS）
  転送元，転送先が同一となるため，ファイル転送を実装する必要がない．ケース 5 と異なり，GFS ファイルの複製は作成しない．

一時ファイルが GFS ファイルである場合，一時ファイルを作成するディレクトリは「gfarm:ng/tmp/」とした．そして，RPC によって呼び出される関数名の引数には，「/gfarm/ng/tmp/⟨filename⟩」のファイル名が渡されるように実装した．このため，計算ノードにおいて，Gfarm のシステムコールフックライブラリがインストールされている必要がある．

3.4 利用方法と制限事項

本機能を利用するためには，クライアントから RPC を行う際のファイル型引数を次のように記述する．

GFS:[$LTMP$|$GTMP$]:[$LFILE$|$GFILE$]:⟨file path⟩
[:[$LFILE$|$GFILE$]:⟨file path⟩]

「GFS:」は拡張機能を利用するためのパラメータが以下に続くことを示す予約語とする．[$LTMP$|$GTMP$] では，一時ファイルをローカルファイル，あるいは GFS ファイルとして作成することを指定する．[$LFILE$|$GFILE$] では，次の「:」に続く ⟨file path⟩ で与えられるファイルがローカルファイルシステム上のパス，あるいは Gfarm ファイルシステム上のパスであることを特定する．続けて指定するファイルパスは，引数の入出力モードが INOUT である場合に，No.6，8，15，17 の転送パターンのように入力と出力とでファイルの所在を変えるのに用いる．これは，INOUT の際に，入力と出力を行うファイルパスが同一であるとしていた既存の Ninf-G と異なる点である．IN か OUT のどちらかであれば，最後のファイルパスは省略可能である．例えば,「gfarm:exp1/input.dat」に存在するファイルを表 1 の No.7 のように計算ノードに転送する場合は,「GFS:LTMP:GFILE:exp1/input.dat」と記述し，図 3 の [Extended description] に示すように RPC を実行する．

表 1 の転送パターンうち，No.2，4，5 は既存の Ninf-G の実装である．No.9，18 は必要とするデータが既に GFS ファイルとして存在するシナリオで利用される．No.11，13 は一時ファイルを GFS ファイルとすることで，障害によって遠隔プログラムが計算途中で終了する対策を行う際に用いられる．No.1，2 を除き，残りの転送パターンは次章で述べる Task Sequencing の実装に用いられる．

```
grpc_begin_sequence(TMP_ON_GFS, DUPLICATION_ON);
grpc_submit("func1", A, B);
grpc_submit("func2", B, C);
grpc_end_sequence();
```

図 4 Task Sequencing API
Fig. 4 Task Sequencing API

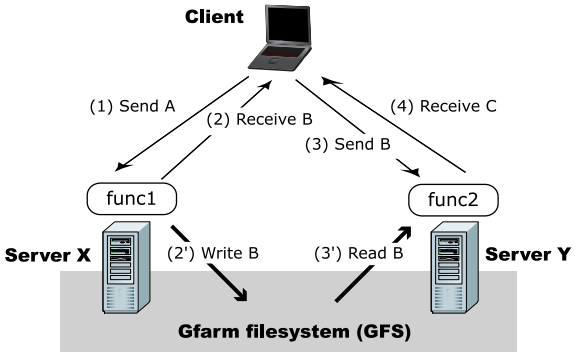

図 5 Gfarm を利用した中間データの転送
Fig. 5 Intermediate data transfer using Gfarm

4. Task Sequencing API の設計と実装

4.1 API

本研究では，Ninf や NetSolve で実装されている Task Sequencing API を参考に，**図 4** に示す API を提供する上位ライブラリを設計する．図 4 において，grpc_begin_sequence() は Sequence の始まりをライブラリに通知する．grpc_submit() は，関数名と引数を与えて RPC の実行を要求する API である．これは，GridRPC で規定されている grpc_call_async() をラップし，RPC の実行先を制御する handle の指定を不要としている．grpc_end_sequence() が呼ばれることで，Sequcence の始まりと終わりの間に要求された RPC の実行先が自動的に決定され，データ転送が最適化される．図 4 の例では，入力データ A を与えて func1 を実行すると B が出力される．B は次の func2 の入力になるため，func2 が func1 と同じ場所で実行できることが望ましい．しかしながら，func2 を別の計算ノードで実行する必要がある場合，func1 から func2 の実行先にデータ B が直接転送されるよう，前章で実装した機能を用いて上位ライブラリ内部で処理される．**図 5** に示すように，従来は (2)，(3) の経路でデータ B を転送していたところを，Gfarm ファイルシステムを経由して (2')，(3') の経路で転送するように変更し，最終的な結果データ C だけをクライアントに返すようにする．RPC の実行先の決定，中間データの転送は本 API ライブラリ内部に隠蔽されるため，アプリケーションプログラムではこれらを気にする必要がない．

一方，本研究で提案するライブラリでは，性能と障害に関するオプションとして中間データの格納場所と複製

1) Allocate a resource to each task (from $Task_1$ to $Task_N$)
2) Set value i to 1
3) Analyze arguments of $Task_i$ and $Task_{i+1}$ in the following loop
3-1) Find a next OUT-mode argument of $Task_i$
If no more argument, go to 3-5)
3-2) Find a next IN-mode argument of $Task_{i+1}$
If no more argument, go to 3-1)
3-3) Compare two arguments found at 3-1) and 3-2) by their data-types and pointers
In case those arguments indicate the same data, the data must be an intermediate which is output of $Task_i$ and input of $Task_{i+1}$.
3-4) Go back to 3-2)
3-5) Increment value i by 1
If value i is $N-1$ or less, go back to 3-1)
Otherwise, exit the loop

図 6 Task Sequence の解析手順
Fig. 6 A process flow of the Task Sequence

表 2 Task Sequencing に用いる転送パターンの分類
Table 2 Classification of data transfer for Task Sequencing

IN–OUT mode	Location of the intermediate file	
	Local	Gfarm
Separate file	No.3, 7	No. 12, 16
Same file	No.6, 8	No. 15, 17

ファイルの作成の有無を指定できるようにした．これらは grpc_begin_sequence() の引数として与え，第 1 引数に TMP_ON_LOCAL が指定されれば，従来通りローカルに中間ファイルを作成して，その転送はクライアントを介して行う．TMP_ON_GFS が指定されれば，GFS ファイルとして中間ファイルを作成して直接転送を行う．第 2 引数に DUPLICATION_ON を指定すると，TMP_ON_GFS が有効である場合に中間ファイルの複製が別の計算ノードに作成される．万が一，Gfarm のストレージノードが利用できなくなった時のバックアップである．

4.2 実　　装

Task Sequencing API ライブラリでは，grpc_submit() が呼ばれた際に，Ninf-G が提供する Argument Stack API を用いて各タスクの引数列を構築する．そして，grpc_end_sequence() の中でそれらの引数列を解析し，効率的に実行できる修正を行った上でタスクを順次実行する．

N 個の RPC からなる Task Sequence を処理する時，grpc_end_sequence() の内部処理は**図 6** の手順で行われる．1）では最初の RPC 要求にある関数名から実行先の候補を選び出して，候補リストの先頭にある計算ノードに RPC を割り付ける．次の RPC 要求にある関数が直前の RPC の実行先で実行可能であれば，実行先を同じとする．実行可能でなければ，候補リストの先頭にある別の計算ノードに RPC を割り付ける．以上の操作を全ての RPC 要求に対して繰り返す．

RPC の実行先が決定した後，3）において前後関係にあ

表 3 計算環境
Table 3 Computational environment

Site (role)	CPU	OS kernel	Globus (flavor)	I/O for Local FS		I/O for Gfarm FS	
				Read	Write	Read	Write
AIST (client)	PentiumIII 1.4GHz × 2	Linux 2.4.20	2.4.3 (gcc32)	144 (46.0)	33.3	–	–
AIST (server)	Xeon 2.8GHz ×2	Linux 2.6.9	3.2.1 (gcc32)	122 (98.2)	112	72.6	92.7
NCSA (server)	Xeon 2.0GHz × 4	Linux 2.4.21	2.4.3 (gcc32pthr)	991 (71.8)	22.5	91.9	21.8

I/O performance unit: Mbytes/sec

表 4 ネットワーク性能
Table 4 Network performance

From	To		
	AIST (client)	AIST (server)	NCSA (server)
AIST (client)	–	59.59 [MB/sec]	0.34 [MB/sec]
AIST (server)	51.46 [MB/sec]	86.13	Not measured
NCSA (server)	0.19	0.34	82.39

る 2 つの RPC の引数を比較し，中間データである引数を発見する．まず，先に実行される RPC の第一引数から順に，出力モードである引数を見つけて，後に実行される RPC の引数のうち，データタイプが同一，かつ入力モードである引数を探す．該当の引数が発見できれば，引数のポインタを比較することで，その引数が 2 つの RPC の中間データであることが確認できる．中間データであれば grpc_begin_sequence() で指定されたパラメータに従って，Gfarm ファイルシステム上に中間ファイルを作成する形で連続するタスクを実行する．

本研究では，RPC の引数がファイル型データである場合を対象に実装を行った．図 4 の例では，func1 の出力ファイルと func2 の入力ファイルが同じ GFS ファイルになるように内部的に処理できればよい．func1 や func2 の一時ファイルの作成場所，引数の入出力モードを考慮して，表 1 に示した転送パターンのうち，Task Sequencing で利用されるパターンを**表 2** に分類する．Separate file は図 4 のように，入力ファイルと出力ファイルが A と B のように分離されている場合であり，Same file は A と B のファイルが同一であり，それが INOUT のモードで与えられる場合である．本研究では，No.3，7 および 6，8 のパターンを用いて Task Sequencing API ライブラリを実装した．一時ファイルを GFS ファイルとする場合は，遠隔プログラムから直接 GFS ファイルを読むことになり，ローカルファイルへのアクセス速度との差を考慮する必要がある．

Task Sequencing のライブラリ内部では，前後関係にある 2 つの RPC の引数の出入力が一致すれば，先に実行する RPC のファイル出力を GFS ファイルに変更し，同じファイルを後に実行する RPC のファイル入力に変更する．Sequence の実行が完了し，クライアントへファイル C が返った時点で，GFS ファイルは消去されるように実装する．

5. 評　　価

本研究では，図 4 に相当する Task Sequencing を産総研と NCSA（National Center for Supercomputing Applications）のクラスタを用いて実行し，Task Sequencing API ライブラリの内部に実装した Gfarm を用いた計算ノード間の直接データ転送の有効性を評価した．実験に用いた計算機は**表 3** の通りである．I/O の項目はローカルファイルシステムと，同じノード内の Gfarm ファイルシステム領域のそれぞれに対して，バッファサイズを 8KB として 1GB のデータの読み書きを行った際の性能である．いずれもローカルファイルシステムへのアクセスにはキャッシュの効果が現れているため，括弧内にディスクアクセスの性能も示す．AIST（server）はディスクに RAID を利用し，NCSA（server）では NFS を経由して RAID にアクセスしている．Gfarm ファイルシステムのメタサーバは SDSC（San Diego Supercomputer Center）サイトのノードの 1 つで動作する．

AIST サイト内の計算機間や NCSA サイト内の計算機間は LAN 接続されている．それに対して，サイト間は広域のネットワークとなる．**表 4** に各計算機間の TCP のスループットを示す．表中の値は，Netperf を用いた 1 分間の計測を時間を変えて 3 回行った結果の平均である．AIST(server) から NCSA(server) への経路については，NCSA 側のファイアウォールによる制限のため計測していない．

5.1 実　　験

実験では遠隔のライブラリ関数として func1，func2 を異なる計算機に用意する．いずれもファイル型データの入力，出力を別々にもち，func1 の実行後に func2 を実行する Task Sequencing ジョブを投入する．全ての実験においてクライアントプログラムを動かすノードを AIST と固定し，func1 と func2 のいずれのプログラムも AIST の計算ノード 2 台に用意して実験を行った結果が**表 5** である．同様に，両方のプログラムを NCSA サイトの計算ノード 2 台に用意して実験を行った結果が**表 6** である．func1 のプログラムを NCSA サイトの計算ノード 1 台に，func2 のプログラムを NCSA サイトの計算ノード 1 台に用意した実験結果が**表 7** である．実験では func1 および func2 の入出力ファイルの

表 5 両方の関数を AIST サイトに用意した場合の転送性能
Table 5 Performance of data transfer when both functions are serviced in the AIST site

Transfer type	10 MB data		50 MB data	
	(Total)−(Func)	(2)+(3)	(Total)−(Func)	(2)+(3)
Gfarm	4.31 [sec]	2.90 [sec]	11.9 [sec]	4.45 [sec]
GASS	2.51	1.10	16.0	8.41
Protocol	1.10	0.470	4.45	1.99
Remote object	1.33	0.00955	4.71	0.0467

表 6 両方の関数を NCSA サイトに用意した場合の転送性能
Table 6 Performance of data transfer when both functions are serviced in the NCSA site

Transfer type	10 MB data		50 MB data	
	(Total)−(Func)	(2)+(3)	(Total)−(Func)	(2)+(3)
Gfarm	70.4 [sec]	7.82 [sec]	302 [sec]	11.8 [sec]
GASS	122	59.2	575	286
Protocol	181	89.7	868	439
Remote object	62.4	0.00842	291	0.0413

表 7 func1 を NCSA，func2 を AIST サイトに用意した場合の転送性能
Table 7 Performance of data transfer when func1 is serviced in the NCSA site and func2 is serviced in the AIST site

Transfer type	10 MB data		50 MB data	
	(Total)−(Func)	(2)+(3)	(Total)−(Func)	(2)+(3)
Gfarm	59.0 [sec]	26.2 [sec]	219 [sec]	66.8 [sec]
GASS	62.2	29.2	295	143
Protocol	90.5	53.5	440	263

サイズを同一に設定し，10MB と 50MB での計測を行った．表中の「(Total)−(Func)」の値は，grpc_begin_sequence() が呼ばれてから grpc_end_sequence() が完了するまでの Task Sequencing 全体に要した時間より，func1 と func2 の実行時間の合計を除いた時間である．「(2)+(3)」の値は図 5 の (2) と (3)，あるいは (2') と (3') で示される中間データの転送に要する時間である．

表 5〜表 7 に示す実験結果において，3 つのファイル転送方式の性能を比較する．表中の Gfarm は本研究にて実装した方式である．Gfarm のストレージノードにアクセスする際の認証には AIST サイト内であれば共通鍵認証，NCSA サイト内および AIST と NCSA 間では GSI 認証を用いている．また，メタデータサーバへのアクセスを高速化するために gfarm_agent を利用した．GASS は Ninf-G が実装する GASS を用いたファイル転送方式であり，実験では http によるアクセスを行うオプションを指定した．Protocol は Ninf-G Ver.2.4 が実装する Ninf-G プロトコル上での転送，すなわち Globus I/O を用いて他のデータ型と同様にファイル型データを転送する．GASS や Protocol ではクライアントを介した転送になる．Remote object は，Task Sequencing API ライブラリを用いずに，Ninf-G のリモートオブジェクト機能を利用して func1 と func2 を 1 つのプログラムに実装し，中間ファイルを計算ノード上のローカルファイルとして保存する実装である．

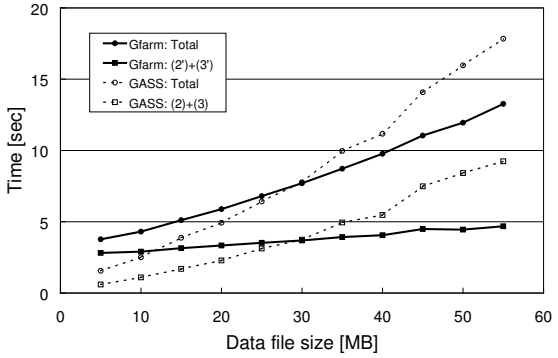

図 7 Gfarm と GASS の中間データの転送時間の比較
Fig. 7 A comparison between intermediate data transfer with Gfarm and one with GASS

図 7 は AIST サイト内において，入出力のファイルサイズに応じた中間データの転送時間への影響を調べた結果である．ファイルサイズを 5MB から 5MB ずつ 55MB まで増やし，Gfarm による転送時間と GASS による転送時間を比較する．

5.2 結果と考察

表 5 では，ファイルサイズが 10MB の時に Gfarm は GASS より遅い．これは Gfarm のストレージにアクセスする際の認証に平均的に 0.2 秒かかること，メタサーバへのアクセスのオーバヘッド，MD5 チェックサムによるファ

イルの一貫性の確認を行っていることが原因である．ファイルサイズが 50MB になれば，クライアントを経由する GASS の転送コストがそれらを上回る．これは図 7 の結果に示されるように，ファイルサイズの増加に対する中間データの転送時間の増加に関して，Gfarm が GASS に比べて小さく，ファイルサイズが 30MB の付近で転送コストが逆転しているためである．

表 6 を見ると，クライアントへデータを転送するコストが格段に大きくなるため，Gfarm を用いた転送方式が GASS を用いた方式に比べて半分の時間で Task Sequencing を処理できる結果となった．表 7 では，func2 の実行先がクライアントと同じサイトにあるため，サイト間にまたがる Gfarm と GASS のデータ転送の性能差が示された．今回の実験環境では効果が小さかったが，環境によっては Gfarm の並列ストリーム機能を用いてさらなる速度向上も期待できると思われる．

Ninf-G の Protocol 上での転送は AIST サイト内では高速であったが，サイト間の転送では性能が悪かった．これは，Gfarm や GASS が遅延の大きなネットワーク上での転送性能を上げるためにソケットの送受信バッファを大きく設定しているのに対し，Ninf-G ではそのような設定を行っていないためである．

Remote object は中間データの転送コストがゼロの結果といえるが，入力ファイルと出力ファイルは GASS を用いて転送しているため，表 5 において，全ての転送を Protocol 上で転送する方式より遅い結果が示された．

これらの結果から，本研究で実装した Task Sequencing API ライブラリが Gfarm を利用することで，サイト間の転送コストが大きい場合に効率良く Task Sequencing が実行できることが示された．サイト内で用いる場合には，転送するファイルサイズに依存して，一定以上の大きさのファイルを扱うのであれば従来の GASS を利用した Ninf-G の転送より短い時間で Task Sequencing を実行できる．実行時間の短縮によるメリットのほか，バックアップ用途に中間ファイルを複製する場合や，クライアントのディスク容量が限られていて中間ファイルをクライアントに戻せない場合にも本機能は有用であると考える．一方，Task Sequencing API を利用することで，このような中間データの所在や転送方式を考慮せずにアプリケーション開発に専念することができる．

6. ファイル型でないデータの直接転送の実装への指針

本研究ではファイル型データの転送を対象に，計算ノード間の直接データ転送を可能にする Task Sequencing API ライブラリを構築したが，これをファイル型以外のデータに適用することは十分に可能であり，その有用性は高いと考える．以下では，本機能をファイル型以外のデータも扱えるように拡張するための実装の指針を述べる．

```
grpc_data_handle_t dhB;
    :
grpc_data_handle_init(&dhB, B);
    :
grpc_call(&handle1, A, &dhB);
grpc_call(&handle2, &dhB, C);
    :
grpc_data_handle_free(&dhB);
```

図 8 データハンドルを用いた RPC の実行
Fig. 8 RPC with data handle

4.2 節に記した実装において，連続する RPC の引数から中間データを判別した後，ファイル型でない引数は func1 において GFS ファイルに書き込まれ，func2 において同じ GFS ファイルから読み出されるように実装すればよい．そのための新しいデータ型としてデータハンドルを定義し，その中で引数のデータタイプ，配列の大きさ，データが保存される GFS ファイルのパスを格納する．Ninf-G や NetSolve では，引数データの大きさは grpc_call() などの RPC の実行関数の中で取得する仕組みであるため，GFS ファイルへの引数の書き込み，読み出し機能は RPC 実行関数の内部に実装する．

データハンドルを用いると，図 4 の func1 と func2 の処理は**図 8** のように記述できる．図 8 において，B は例えば倍精度型の配列である．grpc_data_handle_init() では，初期化されてライブラリ内のリストに登録されたデータハンドル（dhB) のポインタが返される．そして，grpc_call() などの引数に配列 B のポインタの代わりに dhB のポインタを指定することになる．内部的には va_arg() により引数列を構築する際に，リストに登録されたデータハンドルが渡されていないかを確認し，遠隔プログラムの引数情報を使わずにデータハンドルとして処理する．データハンドルであれば GFS ファイル名を決定し，遠隔プログラムにフルパスを通知する．遠隔プログラムにおいて，func1 の B 配列は出力モードであるため，XDR 変換が施された後に GFS ファイルに書き込まれる．func2 の B 配列は入力モードであるため，GFS ファイル内の XDR 変換されたデータを元に戻してメモリに読み込まれる．

図 8 のような処理を Task Sequencing API ライブラリの内部にプログラムしておくことにより，アプリケーション開発者は同じ Task Sequencing API を用いて，計算ノード間の直接データ転送の恩恵を受けることができる．

7. 結　　論

GridRPC システムにおいて，RPC の入出力に依存関係が存在する Task Sequencing ジョブを効率的に実行することを目指して，GridRPC の計算ノード間の直接データ転送を Gfarm ファイルシステムを利用して実現した．Ninf-G の修正を大きく変更することなく，Ninf-G のファ

イル型データに Gfarm ファイルシステム上のファイルを指定できるよう拡張を施し，Gfarm ストレージノードからのデータの読み出し，書き出し，ノード間のデータ複製を Ninf-G の遠隔ライブラリ内に隠蔽した．この拡張機能を利用して Task Sequencing API ライブラリを実装し，日米間にまたがる計算環境で Task Sequencing ジョブを実行する際に，計算ノード間の直接データ転送が有効に動作することを示した．本仕組みを拡張することで，GridRPC のファイル型データ以外のデータについても，Gfarm ファイルシステムを経由して計算ノード間の直接データ転送が可能であり，その実装の指針を示した．

今後は 6 章で述べた指針に沿って実装を進め，Gfarm 以外のグローバルファイルシステムやストレージサーバ上のデータを扱えるようにデータハンドルとその API の標準化を GridRPC の枠組において検討する予定である．一方，Task Sequencing ジョブの資源割り当てメカニズムを改善し，ワークフローを伴う実アプリケーションの実装を通して Task Sequencing API ライブラリを評価する予定である．

謝辞 本研究を行うにあたり，貴重なご意見を頂いた建部修見（現 筑波大）氏に深く感謝いたします．

本研究の実験環境として計算資源を提供頂いた National Center for Supercomputing Applications, San Diego Supercomputer Center に感謝いたします．

参 考 文 献

1) 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 3rd International Workshop on Grid Computing* (Parashar, M.(ed.)), pp. 274–278 (2002).
2) 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).
3) 武宮博, 田中良夫, 中田秀基, 関口智嗣: MPI と GridRPC を用いた大規模 Grid アプリケーションの開発と実行：Hybrid QM/MD シミュレーション, 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 12 (ACS 11) (2005).
4) Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T. and Matsuoka, S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *Grid Computing*, Vol. 1, No. 1, pp. 41–51 (2003).
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 ICL-UT-02-05, University of Tennessee (2002).
6) 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. 44, No. SIG11, pp. 34–45 (2003).
7) Mayer, A., McGough, S., Furmento, N., Lee, W., Gulamali, M., Newhouse, S. and Darlington, J.: Workflow Expression: Comparison of Spatial and Temporal Apporoaches, *In Workflow in Grid Systems Workshop, GGF-10* (2004).
8) 建部修見, 森田洋平, 松岡聡, 関口智嗣, 曽田哲之: ペタバイトスケールデータインテンシブコンピューティングのための Grid Datafarm アーキテクチャ, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, Vol. 43, No. SIG 6 (HPS 5), pp. 184–195 (2002).
9) Nakada, H., Sato, M. and Sekiguchi, S.: Design and Implementation of Ninf: toward a Global Computing Infrastructure, *Future Generation Computing Systems, Metacomputing Issue*, Vol. 15, pp. 649–658 (1999).
10) Arnold, D., Vadhiyar, S. and Dongarra, J.: On The Convergence of Computational and Data Grids, *Parallel Processing Letters*, Vol. 11, No. 2–3, pp. 187–202 (2001).
11) Plank, J.S., Bassi, A., Beck, M., Moore, T., Swany, M. and Wolski, R.: Managing Data Storage in the Network, *IEEE Internet Computing*, Vol. 5, No. 5, pp. 50–58 (2001).
12) 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).
13) 相田祥昭, 中島佳宏, 佐藤三久, 櫻井鉄也, 高橋大介, 朴泰祐: 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, 情報処理学会研究報告, Vol. 2005, No. 97, pp. 7–12 (2005).
14) Caron, E. and Desprez, F.: DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid, Technical Report RR-5601, The French National Institute for Research in Computer Science and Control (2005).
15) Foster, I., Kesselman, C., Tedesco, J. and Tuecke, S.: GASS:A Data Movement and Access Service for Wide Area Computing Systems, *Sixth Workshop on I/O in Parallel and Distributed Systems* (1999).