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

武宮 博[†] 首藤一幸[†]
田中良夫[†] 関口智嗣[†]

気象予報シミュレーションシステムという応用ソフトウェアを Grid 上に実装し，これを通じて標準に用いられている Grid の機能を評価する．当システムは，数週間から数カ月にわたる中・長期的な全球規模の気象変動を精度良く予測することを目的として開発された順圧 S-model と呼ばれる逐次 FORTRAN プログラムを，Grid RPC の参照実装である Ninf-G を利用することによって Grid アプリケーションとしたものである．Ninf-G を用いることにより，Grid 環境の複雑な構造を意識することなく容易に Grid アプリケーション化することができた．韓国，タイに設置された 5 台のクラスタから構成される ApGrid テストベッド上に本システムを実装し，性能評価を試みた．その結果，（1）初期起動コスト，（2）終了検知コスト，（3）クライアントと複数のサーバ間の通信競合，が効率的な処理を妨げていることが分かった．それらを低減するためには，アプリケーションのチューニングだけでなく Globus および Ninf-G における機能追加/修正が必要であることが分かった．

Developing a Simulation System for Atmospheric Prediction on a Grid Environment

Hiroshi Takemiya,[†] Kazuyuki Shudo,[†] Yoshio Tanaka[†]
and Satoshi Sekiguchi[†]

An application system for atmospheric prediction has been implemented on a grid and functions of de facto standard grid middlewares have been evaluated through the implementation work. The system has been constructed by gridifying the sequential FORTRAN program called barotropic S-model which aims to predict middle- to long-term climate change accurately. Ninf-G was used to gridfy the system. By using Ninf-G, the program could be easily gridified without worrying about the complex structure of a grid. We have implemented the system and evaluated its performance on the ApGrid test bed which consists of 5 clusters in Japan, Korea, and Thailand. As a result, (1) process invocation cost, (2) process termination cost, and (3) contention of communication between client and servers hinder from efficient execution. In order to reduce these costs, it is necessary to extend or modify functions of Globus and Ninf-G as well as to tune-up the application itself.

1. はじめに

ネットワーク技術の発達により，広域に分散した種々の資源を連携させ，大規模/複雑な問題を解くという新しい形態の処理方式 Grid Computing が可能となってきた[1)]．

Grid Computing の代表的な処理形態として，複数の並列計算機を同時に利用して 1 つの問題を解く Metacomputing[2)]，広域ネットワーク上に散在する遊休計算資源を利用して大規模な問題を解く Voluntary Computing[3)]，大規模データを世界規模で分散管理する Data Intensive Computing[4)] 等があげられる．

これら処理形態の実現を支援するプログラミングモデルの 1 つとして，Grid 上での RPC（Remote Procedure Call）— Grid RPC —を利用した Grid Computing が提案されている[7)]．Grid RPC は，Grid の複雑さを隠蔽し，ローカルな関数呼び出しと同様なインタフェースを提供するため，Grid に関する特別な知識を持たないユーザが Grid 上にアプリケーションを容易に構築できるという特徴を持つ．

バイオ情報技術，分子科学計算，地球環境科学等多くの分野で大規模な計算資源を必要としており，Grid の利用に対する期待は大きい．しかしそれらの分野の研究者は Grid のスペシャリストではないため，Grid アプリケーションの構築が困難であった．Grid RPC はその障害を除去し，Grid アプリケーションの構築を加速，推進する可能性を持っている．

産業技術総合研究所グリッド研究センター（以下，当センター）では，広域に分散配置された計算資源への

† 産業技術総合研究所グリッド研究センター
National Institute of Advanced Industrial Science and Technology

RPC（Remote Procedure Call）機構として，Ninf[5]と呼ばれるライブラリを開発してきた．また，Ninfの構築で得られた知見に基づき，GGF[6]（Global Grid Forum）においてGrid RPCプロトコルを提案する[7]とともに，Grid RPCの参照実装（Reference Implementation）として，事実上の標準Grid基盤となっているGlobus[8]上にNinf-G[9]を開発してきた．

しかし，現状ではNinf-Gを用いて実際の応用プログラムをいかにGrid上で実装するか，現状どの程度の性能が得られるのか，また，高性能処理を実現するためにはどのような工夫が必要なのかといった情報は少ない．そこで，本稿では，気象予報シミュレーション用に開発された順圧S-modelと呼ばれる逐次アプリケーションを題材とし，Ninf-Gを用いてGrid化した際の具体的な手順，実装上の工夫，実際のテストベッド上における実行結果，およびそれらより得られた知見について述べる．

順圧S-modelプログラムは，Grid上での実行に適していると考えられている典型的な処理形態の1つであるパラメータサーベイを行うプログラムである．したがって，上記の情報は，Gridの利用を期待する潜在的な応用分野の研究開発者に対して非常に有益である．

本稿の構成は以下のとおりである．まず，次章において本システムの概要を述べるとともに，基本構成要素である順圧S-modelプログラムおよびNinf-Gライブラリについて説明する．3章では気象予報シミュレーションシステムの実装方法について述べる．4章において，本システムの実行性能について評価を行った後，最後にまとめと今後の課題について述べる．

2. 気象予報シミュレーションシステム

今回構築した気象予報システムは，数週間から数カ月にわたる中・長期的な全球規模の気象変動を精度良く予測することを目的として開発された順圧S-model[10]と呼ばれる，逐次FORTRANプログラムをNinf-Gによって Gridアプリケーションとした（Ninf化）ものである．

以下に，構成要素である順圧S-modelプログラムおよびNinf-Gに関して説明する．

2.1 順圧S-modelプログラム

順圧S-modelプログラムは，スペクトル法を用いて気象変動を予測するFORTRANプログラムである．

一般に，中・長期にわたる気象予報は，そのカオス的な振舞いのために困難であることが知られている．順圧S-modelプログラムは，鉛直方向に平均化した状態量に関する基礎方程式を解くことで，観測誤差に起因する予報のずれの成長を抑制しているという特徴を持っており，1週間から10日にわたる中期予報に成功している[10]．

また，2次元の簡易なモデルでありながら，大気のバロトロピック要素（barotropic component）に関するシミュレーションを行うことで，バロトロピックな構造（barotropic structure）を持ち中・長期にわたる気象予報に大きな影響を及ぼすジェットストリームの変化や，高気圧のブロッキングといった現象を精度良く再現できる．

順圧S-modelプログラムは，観測データの誤差やシミュレーションの誤差に起因する予報精度の低下を抑制し精度をさらに長期的に保持することを目的として，シミュレーション結果の統計的処理を行っている．具体的には，観測結果に基づく初期値に対して乱数を利用した擾乱を加え，複数の異なる初期値を生成する．これらの初期値に基づいて複数のサンプルシミュレーションを実行するとともに，乱数に基づく擾乱を時間ステップごとに方程式の外力項に加えて計算を行う．得られた計算結果の統計平均をとることで，予報精度の長期的保持を試みている．

統計的な手法を用いて誤差の成長を抑制するためには，数十個から数百個のサンプルシミュレーションを実行する必要がある．これらのシミュレーションは完全に独立に行える典型的パラメータサーベイである．したがって，多数の計算資源を利用できるGrid上で本シミュレーションを実行可能にすることで，効率的かつ精度良い中・長期的気象予報が期待できる．

順圧S-modelプログラムにおけるシミュレーション処理の流れは，以下のようになっている（**図1**参照）．

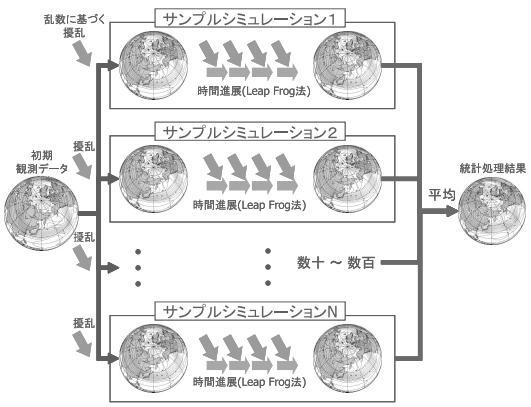

図1 気象予報シミュレーションの流れ
Fig. 1 Execution flow of a weather forecasting simulation.

(1) 以下に示すシミュレーション初期値用観測データおよびシミュレーションパラメータを読み込む.
- 指定された時刻の観測データから算出された初期スペクトル係数データ,
- 外力項として表されるフォーシングパラメータ,
- シミュレーション開始時刻, サンプルシミュレーション数, シミュレーション期間, 結果出力頻度等出力制御パラメータ, 拡散係数等のシミュレーションパラメータ

(2) 初期スペクトル係数データに乱数に基づく擾乱を加え, サンプルシミュレーションを実行する. 各サンプルシミュレーションでは, その時間ステップにおけるスペクトル係数データに対し, 乱数に基づく擾乱を加えた後, Leap Frog 法を用いて時間積分を行う.

(3) 指定された予報期間に達するまで (2) の処理を繰り返す.

(4) 指定されたサンプルシミュレーション数に達するまで (2), (3) の処理を繰り返す.

(5) 得られたサンプルシミュレーション結果 (スペクトル係数データ) に対して, 各成分の平均をとり, 統計処理結果とする.

2.2 Ninf-G ライブラリ

Ninf-G ライブラリは, Globus 上に構築された Grid RPC システムである. Grid RPC は, クライアント–サーバモデルに基づき, Grid 環境上に存在するサーバ上のプログラム (サーバプログラム) に定義された手続きを, 利用者の手元にあるプログラム (クライアントプログラム) から, 容易かつセキュアに呼び出す機構である.

Ninf-G では, Grid 環境上に存在する多数の計算資源の並列利用を支援するために, 通常の同期型の関数呼び出しに加え, 非同期型の呼び出し機構も提供している. これらの手続き呼び出しは, 以下の手順で行われる.

(1) サーバプログラムの手続き呼び出し情報 (引数情報およびパス情報) を取得する.
(2) サーバプログラムを起動する.
(3) サーバプログラムとの接続を確立し, サーバプログラムへの入力データ, サーバプログラムからの出力データを, 引数データとして送受信する.

Ninf-G は Globus のコンポーネントを利用して上記の動作を実現している (**図 2**). すなわち,

(1) サーバプログラムの手続き呼び出し情報は, MDS (Metacomputing Directory Service) と呼ばれるディレクトリサービス上に格納され, クライアントプログラムは MDS に問い合わせることにより, 手続き呼び出し情報を動的に取得する.
(2) サーバプログラムの起動は, GRAM (Grid Resource Allocation Manager) と呼ばれる資源管理サービスによって実現される.
(3) クライアント–サーバ間のデータ送受信は, Globus I/O と呼ばれるライブラリを用いて実現される. Globus I/O は GSI (Grid Security Infrastructure) と呼ばれるセキュリティ基盤上に構築されており, ユーザ認証や暗号化によるセキュアな通信を実現している.

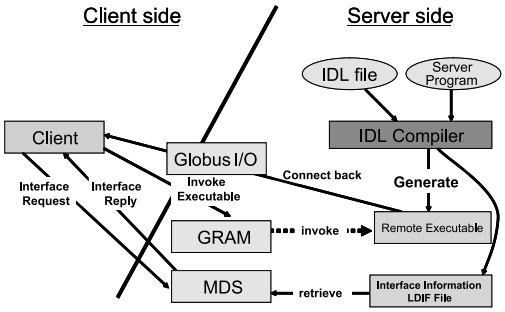

図 2 Ninf-G ライブラリの概要
Fig. 2 Overview of the Ninf-G library.

Globus のコンポーネントを利用して Ninf-G を開発することは, Ninf-G 開発者, アプリケーション開発者双方にメリットがある.

Ninf-G 開発者にとってのメリットは, セキュリティ, 資源管理等の機能開発に煩わされることなく, RPC 固有機能のみに開発を集中できる点である. また, 世界規模で進められている Globus の機能増強, 拡張の成果を, 容易に Ninf-G に組み込むことができるというメリットもある.

アプリケーション開発者にとっても, Globus が複数の機種, OS, バッチシステムに対応しており, また Grid 基盤の事実上の標準として多くのサイトで実装されていることから, Ninf-G の高レベルな API を用いて構築したアプリケーションがそのまま標準的な Grid アプリケーションとなり, 多くのサイトで動作するというメリットを享受できる.

3. 気象予報プログラムの Ninf 化

3.1 逐次プログラムの Ninf 化手順

一般に, 逐次プログラムの典型的な Ninf 化作業手順は, 逐次実行環境におけるプログラム修正作業と Grid 環境におけるプログラム修正作業の 2 種類に大別でき, 以下のようにまとめることができる.

- 逐次実行環境におけるプログラム修正作業

(1) 遠隔呼び出し関数の作成

逐次プログラム中からサーバ上で行うべき処理を抽出し，関数としてまとめる．その際，以下の点を考慮する必要がある．まず，関数の遠隔呼び出しではネットワークを介して引数渡しが行われるため，単一プロセス内の関数呼び出しに比べ，コストが大きい．したがって，なるべく処理の粒度が大きい部分を抽出し，遠隔呼び出し関数とすることが望ましい．

同時に，遠隔呼び出し関数の引数のサイズを小さくすることにより，通信コストを低減する必要もある．また，Ninf-G では，関数に指定された引数を 1 つずつ個別に送付する実装となっているため，レイテンシの大きい Grid 環境におけるスカラ変数の転送は避けたほうがよい．したがって，スカラ引数はなるべく配列化して一括転送を行うことが望ましい．

(2) データ依存関係の除去

遠隔呼び出し関数間のデータ依存関係および，呼び出し元関数–遠隔呼び出し関数間の非明示的データ依存関係を除去する．

抽出部分間にデータ依存関係が存在すると，個々の関数呼び出しを逐次的に実行しなければならない．依存関係をなくせば，非同期呼び出しを行うことで通信の隠蔽，タスクの並列実行が可能となり，処理効率が大幅に高まる．

呼び出し元関数–遠隔呼び出し関数間の非明示的データ依存関係の除去とは，両者の間で共有されている大域変数の除去を意味する．C プログラムにおける Global 変数，FORTRAN プログラムにおける COMMON 変数が対象となる．これらの変数を呼び出し関数の引数として定義する必要がある．

- Grid 環境におけるプログラム修正作業

(3) Ninf-G 関数の挿入

(1)，(2) の作業によって作成した遠隔呼び出し関数をサーバプログラムとして切り出した後，遠隔呼び出し関数を実行するための Ninf-G 関数をクライアントプログラムに挿入する．Ninf-G は遠隔呼び出し関数実行用 API として `grpc_call()`（同期型実行用 API），`grpc_call_async()`，`grpc_wait()`（非同期型実行用 API）を提供しており，クライアントプログラム中の遠隔呼び出し関数実行箇所にそれらの関数を挿入する．これらの関数に加え，初期化関数（`grpc_initialize()`，`grpc_function_handle_init()`）を挿入し，終了関数（grpc_function_handle_destruct()，grpc_finalize()）を挿入することで，関数の遠隔呼び出しが実現される．

(4) スケジューリングルーチンの作成

非均質なネットワークや計算資源から構成され，さらに動的に負荷が変動する Grid 環境でプログラムを効率的に実行するためには，動的なタスク割付けが重要である．したがって，タスクの動的割付けを実現するスケジューリングルーチンを作成する必要がある．

(5) 遠隔呼び出し関数用スタブの生成

遠隔呼び出し関数を実行する各サーバ上でスタブの生成を行う．NinfIDL（Ninf Interface Definition Language）と呼ばれる言語を用いて遠隔呼び出し関数のインタフェース情報を定義し，NinfIDL コンパイラを起動することにより，スタブは自動生成される．

(6) インタフェース情報の MDS への登録

2 章で述べたように，Ninf-G は MDS を用いて動的に遠隔呼び出し関数のインタフェース情報を取得する．このインタフェース情報は，スタブと同様 IDL コンパイラによって LDIF（LDAP Data Interchange Format）形式のファイルとして自動生成される．生成された LDIF ファイルを Globus のインストールされたディレクトリ（deploy ディレクトリ）のサブディレクトリ（`${DEPLOY_DIR}/var/gridrpc/`）に格納する．

3.2 順圧 S-model プログラムの Ninf 化

3.1 で述べた手順に沿って，順圧 S-model プログラムの Ninf 化を行った．

(1) 遠隔呼び出し関数の作成

原プログラムにおいて最も粒度が大きく並列実行可能なサンプルシミュレーション部を切り出し，遠隔呼び出し関数を作成することにした．切り出しに際し，シミュレーション実行のたびに変更される可能性の高い標準入力パラメータおよび指定された時刻のスペクトル係数データのみをクライアント側から全サーバに転送させ，その他のデータはサーバ側でローカルディスクから読み込ませることにより転送データ量の削減を図った．

また，転送されるデータのうちスカラ引数となる標準入力パラメータは，データの型に合わせて配列を宣言し，その配列にコピーをしてまとめることで，一括転送することで，転送データ数の削減も図った．

(2) データ依存関係の除去

- 乱数ルーチンの並列化

順圧 S-model プログラムにおいては，サンプルシミュレーション部における乱数の利用に関してデータ依存関係が存在する．2 章で述べたように，原プログラムでは，タイムステップごとに乱数を利用して擾乱を加えている．乱数の生成は合同乗算法に基づいて行われるため，乱数のシードは前回用いたシードから生成さ

れる．原プログラムでは，サンプルシミュレーションは逐次に実行され，擾乱の作成に使用される乱数シードはそれ以前に行われたサンプルシミュレーションの使用した乱数シードから生成される．したがって，原プログラムをそのまま Ninf 化すると，1 つのサンプルシミュレーションが終了しないと次のサンプルシミュレーションを実行することができない．

この依存関係を解消するために，原プログラムの乱数生成ルーチンを変更し，Leap Frog 法[11] に基づく並列乱数生成アルゴリズムを実装した．Leap Frog 法は，ある乱数系列における乱数シードを N 個の独立なプログラムが利用する場合，各プロセスが M 個（`M >= N`）おきに乱数シードを利用する，あるいは乱数系列を M 個の部分列に分割して利用することにより，異なるプログラム間で同一の乱数シードが利用されないことを保証するアルゴリズムである．

Leap Frog 法を用いることにより，利用する乱数の順番が原プログラムの場合と異なることになるが，本プログラムではもともとランダムな擾乱効果を加えるために乱数を利用しているため，異なるサンプルシミュレーションで同一の乱数シードが利用されて擾乱に歪みが発生しない限り，問題はない．

- COMMON 変数の除去

原プログラムでは，標準入力パラメータ，初期データのほか，シミュレーション結果であるスペクトル係数データが COMMON 変数として宣言されており，クライアント–サーバ間で共有されていた．したがって，それらを遠隔呼び出し関数の引数に組み込んだ．

(3) Ninf-G 関数の挿入

作成したサーバ側関数を遠隔に呼び出すための Ninf-G 関数をクライアントプログラムに挿入した．乱数生成ルーチン内のデータ依存性を消去したため，非同期呼び出し関数（`grpc_call_async()`）を用いて並列実行可能とした．

(4) スケジューリングルーチンの作成

今回の実装では，アイドルとなった計算機にサンプルシミュレーションを 1 個ずつ割り付ける pure self scheduling を実装した．

(5) 遠隔呼び出し関数用スタブの作成

遠隔呼び出し関数起動用スタブを生成するために，IDL ファイルにサンプルシミュレーション実行用関数インタフェースの定義を行った．定義されたインタフェースを**図 3** に示す．

IDL ファイルにおいて，遠隔呼び出し関数の引数情報は Define 文で定義される．図中で入力データとして定義（IN）されている IBUF，DBUF 変数は，各々int 型，double 型の標準入力パラメータを格納したものである．もう 1 つの入力データ（WW）は，指定された時刻におけるスペクトル係数データである．出力データ（OUT）として定義されているのは，TARRAY と WTOT 変数であり，各々，タイムステップ幅を格納した配列，各タイムステップにおけるスペクトル係数データを格納した配列となっている．

```
Module S-model;
Define servmain(IN int N1, IN int IBUF[N1],
        IN int N2, IN double DBUF[N2],
        IN int N3, IN double WW[N3],
        IN int N4, OUT double TARRAY[N4],
        IN int N5, OUT double WTOT[N5])
Required "S-model_serv.o"
Calls "Fortran" servmain(N1, IBUF, N2, DBUF,
        N3, WW, N4, TARRAY, N5, WTOT);
FortranFormat "%s_";
```

図 3 サンプルシミュレーション実行用 IDL ファイル
Fig. 3 IDL file for executing sample simulations.

(6) インタフェースファイルの MDS への登録

定義された IDL ファイルを基に LDIF ファイルを生成し，MDS へ登録した．

これら一連の処理以外に，今回の S-model プログラムの Ninf 化に特化した作業として，Fortran プログラムから Ninf-G ライブラリを呼び出すためのラッパを作成した．

前述のように，S-model プログラムは FORTRAN で記述されている．大規模科学技術計算プログラムの多くが FORTRAN で記述されている現状を考慮し，Ninf-G は FORTRAN で記述されているサーバプログラムには対応している．しかし，比較的処理が少ないと想定されるクライアントプログラムに対しては，まだサポートがなされていない．そのため，ラッパの作成が必要となった．

3.3 開発容易性の評価

S-model プログラムにおける Ninf 化作業項目，および作業量の指標としてのプログラム修正量（原プログラムの行数は 807 行）を以下に示す．

(1) 遠隔呼び出し関数の作成
入力処理の一部およびサンプルシミュレーション処理の関数化：50 行

(2) データ依存関係の除去
乱数ルーチンにおけるデータ依存関係の除去：80 行 COMMON 変数の除去および遠隔呼び出し関数の引数化：40 行

(3) Ninf-G 関数の挿入
Ninf-G 初期化/終了関数，非同期呼び出し実行

関数の挿入：10 行

(4) スケジューリングルーチンの作成
pure self scheduling アルゴリズムの実装：50 行

(5) IDL ファイルの作成
遠隔呼び出し関数インタフェースの記述：5 行

(6) インタフェースファイルの MDS への登録
LDIF ファイルの MDS への登録：0 行

上述のように，プログラムの Ninf 化作業は逐次実行環境におけるプログラム修正作業（上記作業項目 (1)，(2)）と並列実行環境におけるプログラム修正作業（上記作業項目 (3)，(4)，(5)，(6)）の 2 種類に大別できる．今回の作業においては，逐次実行環境において 170 行，Grid 環境において 65 行のプログラム修正が行われ，前者の作業に 1 週間，後者の作業に 1 日の作業時間を要した．一般にデバッグが困難であるといわれる並列実行環境での作業が逐次実行環境における作業時間よりも短くなった要因は，Ninf-G が RPC に基づくプログラミングを採用していることにより，

(1) 最も修正量の多い作業を逐次環境で行うことができた，

(2) 逐次環境でプログラムが正しく動作することを確認すれば，抽出したサーバプログラムの実行に必要な追加部分は，IDL コンパイラによって自動生成され，その動作は保証される，

(3) その結果，Grid 環境で行われるプログラム修正作業は，クライアントプログラムのみに集約される，

点である．

アプリケーションを Grid 化する際には，Globus を直接利用したり，あるいは MPICH-G2 [12] に代表されるメッセージパッシングライブラリを利用する事が考えられるが，これらのツールを用いた場合，クライアント，サーバプログラムの双方の修正作業を Grid 環境で行わなければならない．

たとえば，メッセージパッシングに基づく作業では，依存性を持つ部分を発見した場合，その部分のアルゴリズムを変更すると同時にクライアント，サーバ双方のプログラムに send，receive 関数を用いてデータの授受処理を陽に記述しなければならない．さらに，それが正しく動作するかどうかは，Grid 環境で実際にプログラムを実行してみなければ分からない．そのため，プログラムが複雑になるし，うまく動作しない場合は Grid 環境でデバッグを行わなければならない．

当然，任意のプログラムに対して RPC に基づく Grid 化が有効であるわけではなく，処理の形態によってはメッセージパッシングに基づく Grid 化のほうが容易に行えるような場合も存在する．たとえば，サーバ間でデータの授受が発生するような処理を考えると，メッセージパッシングの場合，データの授受を自然に記述できるのに対し，RPC の場合，サーバ間のデータ授受を直接行うことができないため，いったんクライアント側にデータを渡して再度対応するサーバにデータを送信するといったアルゴリズムを実装する必要がある．これは不必要にプログラムを複雑化するし，効率も低下する．

しかしながら，今回の S-model プログラムに代表される独立なタスクを複数のサーバ上で実行させるような処理形態のプログラムに対しては，RPC に基づく Grid 化は非常に適しているということができる．修正を行ったプログラムから対象となる関数を切り出し，その関数の呼び出し引数情報を定義するだけで自動的にスタブが生成され，クライアントとサーバ間の連携が保証される Ninf-G の機構は開発者の負担を大きく軽減する．

Ninf 化作業におけるその他の特徴としては，作業項目中にプログラムを実行する計算機に関するセキュリティやバッチシステムの違いを意識させるような項目がないことである．一般にネットワークを介して連係動作させるシステムを構築する際は，実行対象となる計算機のセキュリティポリシやバッチシステムの種類等に関する知識が必要であるが，Ninf-G を利用してシステムを構築する限りそのような知識は必要なかった．このようなアプリケーションの動作とは直接関係ない知識が不要であることは，アプリケーション構築者にとって重要である．

4. 実行性能評価

構築した気象予報シミュレーションシステムを実際に Grid 環境で動作させ，性能評価実験を行った．本章ではその結果について述べる．

4.1 実験環境

実験に際して，ApGrid [13] と呼ばれる Grid テストベッドを構成する産業技術総合研究所（以下 AIST，日本），Korea Institute of Science，Technology and Information（以下 KISTI，韓国），Kasetsart University（以下 KU，タイ）の 3 サイトに設置された計 6 台 158 CPU の Linux クラスタ上に気象予報シミュレーションシステムを実装した．

気象予報クライアントプログラムは，AIST に設置された Linux クラスタ（Koume）上で動作する．サーバプログラムは，AIST（1 台，Ume），KISTI（3 台），

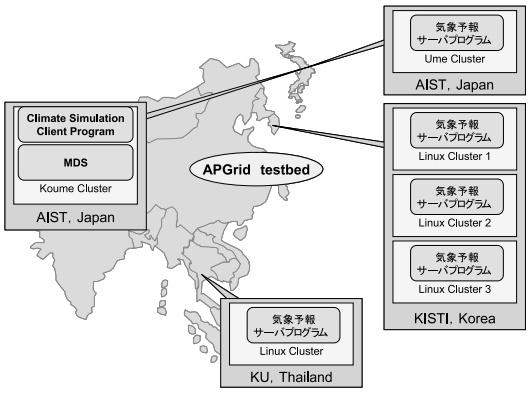

図 4 実行性能評価実験環境
Fig. 4 Testbed for performance evaluation.

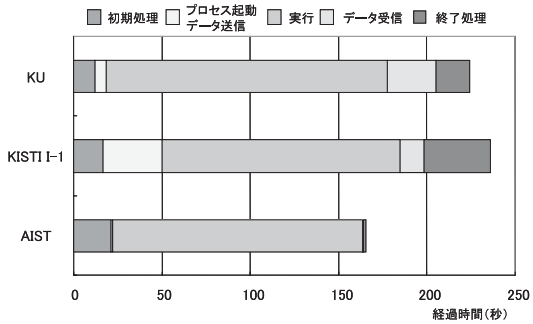

図 5 単一サンプルシミュレーション実行時間
Fig. 5 Performance result of a single sample simulation.

KU（1 台）の Linux クラスタを利用して動作する．（**図 4** 参照）．これらのクラスタは CPU が各々異なり（Ume: Intel PentiumIII，KISTI: Intel PentiumIV，KU: AMD Athlon），プログラム実行モードも異なる（Ume クラスタおよび KU のクラスタはインタラクティブ実行，KISTI のクラスタは PBS を用いたバッチ実行）．また，Koume クラスタ–Ume クラスタ間は，Giga bit Ethernet で接続され，その他のサーバは WAN（Wide Area Network）を介して Koume クラスタと接続されているという非均質なシステム構成となっている．

Globus の MDS は AIST の koume クラスタ上で動作しており，各サイトの Linux クラスタ上で動作するサーバプログラムの手続き呼び出し情報が格納されている．

4.2 評価対象

今回の計測では，北半球を対象として，2002 年 1 月 1 日を基点とした 100 日間の気象予報を行った．この場合，サンプルシミュレーションに転送されるデータサイズは 3.4 KB，サンプルシミュレーションから転送されるデータサイズは 1,350 KB となる．

測定項目は以下のとおりである．

(1) 初期処理時間
MDS からの遠隔呼び出し手続き情報入手時間および GRAM への遠隔呼び出し手続き要請時間

(2) プロセス起動およびデータ送信時間
GRAM によるプロセス起動，クライアント–サーバ間の接続確立およびサンプルシミュレーションへのデータ転送時間

(3) シミュレーション実行時間
サンプルシミュレーションの実行時間

(4) データ受信時間
サンプルシミュレーションからの結果データ転送時間

(5) 終了処理時間
呼び出された遠隔手続きの終了後，GRAM がその終了を検知し，クライアントプログラムに終了を通知するまでの時間

4.3 単一サンプルシミュレーション実行結果

まず最初に各 Linux クラスタ上で 1 個のサンプルシミュレーションを実行した場合の性能測定を行った．結果を，**図 5** に示す．

2.2 で述べたように，順圧 S-model プログラムは鉛直方向に平均化した状態量に関する二次元モデルであるため，100 日間の予報処理に対してシミュレーション時間は 150 秒程度と比較的軽微である．

シミュレーション実行時間と比較して，LAN 環境（AIST 内での実行）において 15%，WAN 環境（KISTI，KU での実行）において 30〜40%程度，Ninf 化によるオーバヘッドが発生している．

主な原因は，

(1) MDS へのアクセスに 10 秒から 20 秒程度必要としている（初期処理コスト）．

(2) KISTI では PBS を用いたバッチ運用を行っているため，GRAM に対するタスク起動の懇請から実際に起動されるまでに 20 秒程度の時間を要する．この間，クライアントプログラムはサーバプログラムとの接続待ちとなる（タスク起動コスト）．

(3) WAN 環境での実行では，データの送受信に各々 10 秒程度要する（データ転送コスト）．

(4) KISTI および KU では，GRAM がサンプルシミュレーションの終了を検知するのに 10 秒から 30 秒程度必要とする（終了処理コスト）．

である．

このうち (1)，(2)，(4) のコストは，以下の手法により低減可能である．

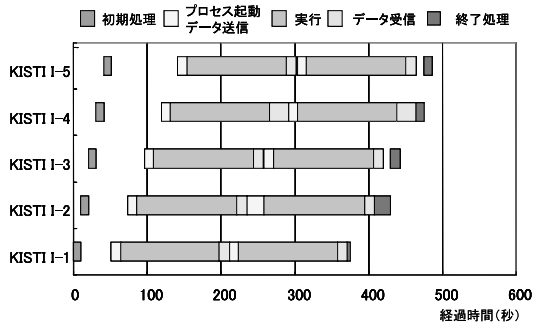

図 6 10 サンプルシミュレーション実行時間（5 ノード利用）
Fig. 6 Performance result of 10 sample simulations on 5 nodes.

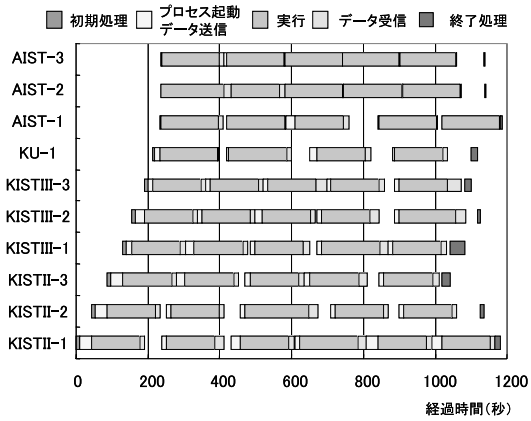

図 7 50 サンプルシミュレーション実行時間（10 ノード利用）
Fig. 7 Performance result of 50 sample simulations on 10 nodes.

(1) Ninf-G では，クライアント計算機上に遠隔手続き呼び出し情報をファイルとして準備することにより，MDS を利用することなくサーバプログラムにアクセスできる機能がある．MDS を利用すれば遠隔手続き呼び出し情報を一括管理できるのに対し，クライアント計算機ごとに呼び出し情報を準備しなければならないというデメリットはあるが，この機能を利用することにより初期処理コストを低減できる．

(2) GRAM のタスク起動時間自体を短縮することはできないが，GRAM のタスク起動はクライアントプログラムと独立に動作する．したがって，`grpc_handle_init` 関数呼び出しによるタスク起動懇請と `grpc_call_async` 関数によるタスク実行の間に他の処理を入れ込むことにより，タスク起動コストを隠蔽することができる．

(3) GRAM は起動したプロセスに対し定期的にポーリングをかけることにより，タスクの終了を検知する．この間隔がデフォルトで 30 秒になっているため，終了処理コストが大きくなっている．このコストは，Globus のソースを修正しポーリング間隔を短縮することにより低減可能である．実際，AIST サイトではポーリング間隔を 5 秒に設定しているため，図から分かるように，他のサイトと比較して終了処理コストが小さくなっている．

4.4 複数サンプルシミュレーション実行結果（1）

上記 (1)，(2) の効果を評価するために，KISTI のクラスタ 5 ノードを利用して計 10 サンプルシミュレーションを実行させた（**図 6**）．本評価では，5 ノードに対する初期処理を連続して行うことにより，GRAM のタスク起動コストの隠蔽を試みている．

上記の手法を用いることにより，初期処理時間は 40%，タスク起動およびデータ転送時間は 70%減少している．これらのコストは起動時にのみ発生するものではあるが，利用する計算機台数に比例して増大するため，多数の計算機を利用する場合は考慮する必要がある．

4.5 複数サンプルシミュレーション実行結果（2）

Grid RPC を利用した効率的なシミュレーション実行を妨げる要因としては，起動時コスト以外に，通信コスト，ネットワークや計算機の非均質性に起因するロードインバランス，クライアントの処理とサーバの処理の不均衡に起因するクライアント処理のボトルネック等があげられる．これらのコストを評価するために，実問題に近いサイズのシミュレーションとして，3 サイト 4 台のクラスタ計 10 ノードを用い，計 50 個の 3 カ月予報サンプルシミュレーションを実行した．結果を**図 7** に示す．

今回の実験では，シミュレーション経過時間（1,197 秒）に占めるシミュレーション実行時間の割合は 57.7%にとどまった．図から分かるように，実行効率低下の最大の原因は，シミュレーション実行初期において複数のサーバからクライアントへの通信が集中し，クライアントの通信処理がボトルネックとなってしまったためである．図において，個々のサンプルシミュレーション実行間の空白として表されているサーバアイドル時間は，総経過時間の 14%を占めている．

また，初期起動コストも性能低下原因の 1 つとなっている．前節の実験と同様，本実験でも GRAM へのタスク起動請願を一括して行うことで GRAM によるタスク起動コストの隠蔽を図っているが，それでも 10 ノードすべてにタスクを割り付けるのに，経過時間の 10%を超えるコストが必要となっている．

通信の集中による実行性能の低下は気象シミュレー

ションに限った現象ではなく，クラスタ上に複数の均質なタスクを割り付ける際にはつねに発生する可能性がある．

複数のサーバからの通信競合に起因する処理効率の低下を回避するためには，

(1) サーバに割り付ける初期タスクの数をばらつかせることにより，通信の同時集中を軽減する，
(2) 計算機やネットワーク資源の状況に応じて割り付けるタスクの数を変化させる，

ことが重要である．

タスク数の変化による競合の回避は並列計算分野において従来から研究されており，Guided Self Scheduling (GSS)[14] 等のアルゴリズムが提案されている．しかしながら，これらのアルゴリズムは均質な環境におけるタスクのスケジューリングを目的としたもので，非均質かつ動的に負荷が変化する環境では有効ではない．Grid 環境では，負荷に応じて動的にタスクサイズを変化させることが必要である．そのためには，資源の負荷状況をモニタし，最適なタスク数の推測を支援する機能が必要である．

気象シミュレーションの場合，複数のサンプルシミュレーションをまとめてサーバに割り付けても，1 タスクあたりの通信量を一定に保つことが可能である．すなわち，送信データでは，乱数のシードのみがサンプルシミュレーションごとに異なるだけである．また，得られた結果は最終的に平均化されるため，複数のサンプルシミュレーション結果の平均をサーバ側でとるようにすれば，受信データのサイズも一定となる．資源の負荷状況をモニタする機構があれば，この特性を利用して負荷状況にあわせた割付タスク数の調整が可能である．

一方，初期起動コストの増加の原因は，クラスタ上の個々のノードに対しサーバプログラムの実行のために GRAM を毎回呼び出していることにある．そのため，利用ノード数に比例して起動コストが増加してしまう．このコストを低減するために，同一クラスタ内のノードに対し，一度に複数のタスクを生成する機能を Ninf-G に実装することで GRAM 呼び出し回数を減少させることを検討している．

5. まとめ

本稿では，典型的なパラメータサーベイプログラムである順圧 S-model プログラムを Ninf-G により Grid 化した際の具体的な手順，実装上の工夫，ApGrid テストベッド上での実行結果，およびそれらより得られた知見についてまとめた．

既存プログラムの Grid 化は，Ninf-G を利用することで，Grid の複雑な機構を意識することなく容易に行うことができた．

また，ApGrid テストベッドを構成する 3 サイト計 5 台の Linux クラスタ上に本システムを実装し，性能測定を行った．その結果，遠隔呼び出し関数の生成/終了コスト，クライアント–サーバ間の通信競合により，効率的な処理が妨げられていることが分かった．

これを解決するためには，アプリケーション，Ninf-G，Globus のすべてにわたって機能追加，修正が必要であることが分かった．具体的には，Globus に関して，

(1) MDS アクセスコストの低減，
(2) PBS 等バッチシステムへの GRAM によるタスク起動時間の低減，
(3) GRAM の終了検知コストの低減，

Ninf-G に関して，

(1) クラスタへの一括タスク生成機能の実現，
(2) モニタ機能の実現，

シミュレーションプログラムに関して，

(1) 動的なタスクサイズ変更機能の実現，

が重要であることを指摘した．

最後に，本稿では気象予報シミュレーションシステムの実装について述べたが，気象研究者や一般の利用者が本システムを容易に利用可能とするためには，ネットワークを介した本システムへの容易かつセキュアなアクセス機構を提供するポータルシステムの構築が重要である．現在，当センターでは，Grid lib と呼ばれるポータルフレームワークを利用して，気象予報システムのポータル化を進めている[15]．

謝辞 本研究に際し，順圧 S-model プログラムを提供いただくとともに，本システムの仕様策定に貴重な御助言をいただいた筑波大学田中博助教授に感謝いたします．

また，ApGrid 参加組織の皆様，特に実験に際して計算資源を提供していただいた KISTI，KU に感謝いたします．

参考文献

1) Foster, I. and Kesselman, C. (Eds.): *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann (1999).
2) たとえば Kimura, T. and Takemiya, H.: Distributed Parallel Computing for Fluid-Structure Coupled Simulations on a Heterogeneous Parallel Computer Cluster, *Journal of Supercomputer Applications and High Perfor-*

mance Computing, Vol.13, No.4 (1999).
3) たとえば，http://setiathome.ssl.berkeley.edu/
4) たとえば，http://www.eu-datagrid.org
5) Nakada, H., Sato, M. and Sekiguchi, S.: Design and implementations of Ninf: towards a global computing infrastructure, *FGCS*, Vol.15, pp.649–658 (1999).
6) http://gridforum.org
7) Nakada, H., Matsuoka, S., Seymour, K., Dongarra, J., Lee, C. and Casanova, H.: GridRPC: A Remote Procedure Call API for Grid Computing, *GWD-I*, APM Research Group, http://www.eece.unm.edu/~apm /docs/APM_GridRPC_0702.pdf (2002).
8) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *Proc. Workshop on Environments and Tools, SIAM* (1996). http://www.globus.org
9) Tanaka, Y., et al.: Globus による Grid RPC システムの実装と評価，HPC 研究会報告，Vol.87, No.29, pp.165–170 (2001).
10) Tanaka, H.L., et al.: A study of Deterministic Predictability for the Barotropic Component of the Atmosphere, Science Reports of the Institute of Geoscience, University of Tsukuba, Section A, Vol.22, pp.1–21 (2001).
11) Foster, I.: *Designing and Building Parallel Programs*, Addison Wesley (1994).
12) Karonis, N., Toonen, B. and Foster, I.: MPICH-G2: A Grid-enabled Implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing* (2003). http://www3.niu.edu/mpi/
13) http://www.apgrid.org
14) Polychronopoulos, C. and Kuck, D.: Guided self-scheduling: A practival scheduling scheme for Parallel Supercomputers, *IEEE Trans. Comput.*, Vol.36, No.12, pp.1425–1439 (1987).
15) 首藤一幸，武宮　博，平野基孝，田中良夫，関口智嗣：気象予報グリッドポータルの開発，情報処理学会研究会報告，Vol.2003, No.29, pp.167–172 (2003).

(平成 15 年 2 月 3 日受付)
(平成 15 年 5 月 30 日採録)

武宮　博（正会員）

日立東日本ソリューションズ（株）公共ソリューション本部サイエンス&テクノロジーセンター主任研究員．昭和 61 年東北大学大学院理学研究科天文学博士前期課程修了．平成元年同博士後期課程中退．同年日立東日本ソリューションズ（株）入社．平成 14 年より産業技術総合研究所グリッド研究センターへ派遣．

首藤　一幸（正会員）

平成 8 年早稲田大学理工学部情報学科卒業．平成 10 年同大学メディアネットワークセンター助手．平成 13 年同大学大学院理工学研究科情報科学専攻博士後期課程修了．同年産業技術総合研究所入所．現在に至る．博士（情報科学）．分散処理方式，プログラミング言語，言語処理系，情報セキュリティ等に興味を持つ．IEEE–CS，ACM 各会員．

田中　良夫（正会員）

昭和 40 年生．平成 7 年慶應義塾大学大学院理工学研究科後期博士課程単位取得退学．平成 8 年技術研究組合新情報処理開発機構入所．平成 12 年通産省電子技術総合研究所入所．平成 13 年 4 月より独立行政法人産業技術総合研究所．現在同所グリッド研究センター基盤ソフトチーム長．博士（工学）．グリッドにおけるプログラミングミドルウェア，計算ポータル，およびテストベッド構築に関する研究に従事．IC'99 論文賞．ACM 会員．

関口 智嗣（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 59 年筑波大学大学院理工学研究科修了．同年電子技術総合研究所入所．情報処理アーキテクチャ部主任研究官．以来，データ駆動型スーパーコンピュータ SIGMA-1 の開発等の研究に従事．平成 13 年独立行政法人産業技術総合研究所に改組．平成 14 年 1 月より同所グリッド研究センターセンター長．並列数値アルゴリズム，計算機性能評価技術，グリッドコンピューティングに興味を持つ．市村賞受賞．日本応用数理学会，ソフトウェア科学会，SIAM，IEEE 各会員．