並列 CORBA におけるデータ分散管理手法

蒲 池 恒 彦[†] Thierry Priol[††]

複数の並列プログラムを分散コンピューティング環境下でカップリングし，高性能並列分散コンピューティングを実現するフレームワークとして，並列 CORBA システムを開発した．並列 CORBA システムは，CORBA を拡張し，データ並列プログラムを容易に，かつ効率良く CORBA オブジェクトにカプセル化可能としたシステムである．本システムにより，たとえば，複数の並列シミュレーションコードをカップリングした複合シミュレーションが容易に実現可能となる．本稿では，並列 CORBA システムにおけるデータ分散管理手法，特に並列オブジェクト間の通信時に必要となる，パラメータ配列のデータ再分散機能の実装方法について詳細に述べる．並列クライアント–並列サーバ間で必要となるパラメータ配列の再分散機能は，スタブ（クライアント側），スケルトン（サーバ側）の両方に実装し，コンパイル時にどちらで再分散を行うか指定可能とした．これにより，並列分散コンピューティング環境下で，データを再分散するために必要となる処理を最も高速に行える並列マシン上でデータ再分散を実行させることができる．また，並列コンピュータ NEC Cenju-4 と PC クラスタからなる異機種並列分散システム上での評価結果によりその有効を示す．

Distributed Data Management for Parallel CORBA

Tsunehiko Kamachi[†] and Thierry Priol[††]

We designed and implemented a parallel CORBA: a parallel and distributed computing infrastructure which makes it possible to couple different parallel programs easily and efficiently. The parallel CORBA is designed based on an extension to the Common Object Request Broker Architecture (CORBA) in order to encapsulate a parallel program into a CORBA object. In this paper, we present techniques for handling distributed data within the parallel CORBA and redistributing a distributed parameter between two parallel objects. We implemented a capability of performing data redistribution within the parallel CORBA in both a stub and a skeleton. This allows programmers to select a client or a server to carry out redistribution in order to obtain the maximum performance under their distributed computing environment. Experimental results on a practical heterogeneous parallel and distributed system, consisting of a NEC Cenju-4 parallel computer and a PC cluster, demonstrate the effectiveness of our approach.

1. はじめに

近年，ネットワークの高速化にともなって，地理的に分散されたネットワーク上の計算資源を組み合わせて 1 つのシステムを構築する，グローバルコンピューティング（グリッドコンピューティング）の研究がさかんに行われており[1)]，グローバルコンピューティングシステムにおける共通の通信インタフェースとして MPICH-G[2)]が提供されている．MPICH-G は Globus[3)]が提供するサービスを利用して構築された MPI インタフェースであり，MPICH-G を利用することによって，MPI で記述された既存の並列プログラムをそのままグローバルコンピューティングシステムで実行することができる．しかしながら，並列マシン内のネットワークと計算機間を結合するネットワークの性能差はまだ大きく，並列マシン向けに開発された単一の並列プログラムをグローバルコンピューティングシステム上で高速化するのは困難である．

このため，我々は，グローバルコンピューティング向けのアプリケーション開発手法として，機能的に分割されたアプリケーションを 1 つの単位とし，これらを有機的に結合して複合アプリケーションを構築するアプローチが有力であると考えている．たとえば，複数のシミュレーションコードをカップリングした複合シミュレーションの実現である．ここで問題となるのが，いかにして複数の並列プログラムをカップリング

† NEC インターネットシステム研究所
Internet Systems Research Laboratories, NEC Corporation
†† IRISA/INRIA

させるかである．異なるプログラム間の通信を MPI の通信インタフェースを用いて実現することは可能ではあるが，プログラミングが煩雑になるばかりでなく，オリジナルのプログラムを修正しなければならないという問題がある．

このような背景から，我々は，分散コンピューティングのプラットフォームとして標準化されている CORBA（Common Object Request Broker Architecture）をベースシステムとして選択し，これを拡張して，複数の並列プログラムを分散環境下でカップリングし，高性能並列分散コンピューティングを実現する，並列分散コンピューティングシステムを構築した．

CORBA を選択した理由は次のとおりである．CORBA は，機能単位に分割したアプリケーション（オブジェクト）を分散システム上で実行・通信・管理させるための仕組みを提供するものであり，オブジェクト間のインタフェースを IDL ファイルに記述するだけで，各々のアプリケーションプログラムを変更することなく，オブジェクト間のカップリングを容易に行うことができる．したがって，複数の並列プログラムのカップリングを，分散コンピューティング環境下で実現するためのシステムとして CORBA は有力な候補となりうる．

しかしながら，高性能コンピューティングの観点からは，CORBA にはいくつかの問題点がある．1 つは，CORBA は高機能なサービスを提供する反面，オブジェクト間の通信性能は必ずしも高くないということである．この問題に対しては，現在，ORB 通信の効率化/高速化の研究が進められており，高性能な CORBA の実装が行われつつある．

もう 1 つの重要な問題は，並列プログラムを CORBA オブジェクトとして実装する手段が提供されていないということである．我々は，この問題を解決するために CORBA を拡張し，複数の並列プログラムを容易に，かつ効率良く CORBA オブジェクトにカプセル化できる並列 CORBA システムの開発を行った[4),5)]．並列 CORBA システムは，並列プログラム内では通信インタフェースとして MPI を利用して並列処理を行い，並列プログラム間では ORB を利用して分散処理を行う，2 つの通信パラダイムを融合させた並列分散システムである．なお，我々は，ベースとなる CORBA の実装として MICO[6)]を利用した．

以下，本稿では，まず 2 章で CORBA の概要を述べ，3 章で我々が CORBA を拡張して開発した並列 CORBA システムの概要と，並列プログラムを CORBA オブジェクトにカプセル化する方法について説明する．4 章で並列 CORBA システムにおけるデータ再分散機能の実現方法の詳細を説明し，5 章で Cenju-4 と PC クラスタからなる異機種並列分散システムを用いて行った評価結果を報告する．6 章で関連研究との比較を行い，最後に 7 章でまとめと今後の課題について述べる．

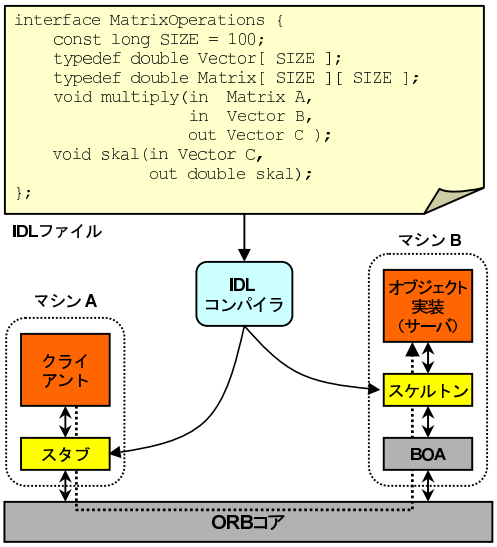


図 1 CORBA アーキテクチャ
Fig. 1 CORBA architecture.

2. CORBA の概要

CORBA のアーキテクチャを**図 1** に示す．CORBA は，米 OMG（Object Management Group）が作成した，異なるハードウェア/OS 上のアプリケーションがネットワークで連携できるようにするための分散オブジェクト管理の標準仕様であり，主に以下の仕様を定めている[7)]．

(1) 異機種上のオブジェクトの連携を可能にする ORB（Object Request Broker）
(2) インタフェース記述言語（IDL：Interface Definition Language）
(3) 分散オブジェクト環境の利用を支援するサービス

CORBA では，アプリケーションはすべてオブジェクトと見なされ，CORBA の仕様に従った ORB によって分散オブジェクト間の連携が可能となる．ORB では，ネットワーク上に存在する，あるサービスを実行するプログラムをオブジェクトと見なし，クライアントプログラムは，このオブジェクトへのメソッドの呼び出しを行うという形でサービスの実行を要求する．CORBA のアプリケーションは以下の手順で作成

する．

(1) サーバ，クライアントの処理を記述したソースをそれぞれ作成する．

(2) サーバのインタフェースを IDL ファイルに記述する．

(3) IDL ファイルを IDL コンパイラでコンパイルする．IDL コンパイラは，スタブ，スケルトンと呼ぶソースを生成する．

(4) クライアントソースとスタブ，サーバソースとスケルトンをそれぞれコンパイル・リンクして，クライアントオブジェクト，サーバオブジェクトを作成する．

クライアントがサーバに実装されている関数（図 1 の例では multiply と skal）を呼び出すと，スタブ，スケルトン，ORB，BOA（Basic Object Adaptor）☆を介してリクエストが送られ，サーバが起動される（図 1 中の破線矢印）．この遠隔起動にともなう煩雑な処理は，スタブ，スケルトン，ORB，BOA によって行われるため，プログラマがクライアント–サーバ間の通信コードを明示的に記述する必要はない．

3. 並列 CORBA の概要

我々は，並列マシンを分散コンピューティングシステムのコンポーネントとし，この上で動作する並列プログラムを CORBA オブジェクトにカプセル化することで，CORBA をベースとした高性能並列分散コンピューティング環境を構築することを目的としている．

ここで，データ並列 SPMD プログラムを CORBA オブジェクトとして実装する方法を考える．並列プログラムでは，同時に複数のプロセスが並列実行されるため，クライアントがサーバに実装された関数を呼び出す場合，クライアントから複数のプロセスに対して同時にリクエストを送出する必要がある．また，データ並列プログラムでは各プロセス間でデータを分散しているため，分散されたデータを送受信するためにも，クライアントはサーバ側の全プロセスと直接通信する必要がある．

しかし，CORBA では，サーバオブジェクトを呼び出す際には，リクエストは 1 度しか送信できないため，SPMD プログラムをサーバアプリケーションとして実装するためには，**図 2** に示すように，SPMD プログラムの 1 プロセスをマスタ，他プロセスをスレーブと見立て，マスタプロセスをサーバとして CORBA オブジェクトにカプセル化し，クライアントと SPMD プログラム間のデータの送受信はすべてこのマスタプロセスを経由するという実装方法をとらなければならない．しかしながら，この方法では，クライアント–サーバ間のデータ転送のたびに分散されたデータをサーバのマスタプロセスに収集する必要があるため，クライアント–サーバ間のデータ転送が逐次化されることによる通信性能の劣化，およびメモリ利用効率の低下を招くという問題がある．さらに，オリジナルの SPMD プログラムを，マスタ/スレーブ方式で実行するように修正する必要が生じる．

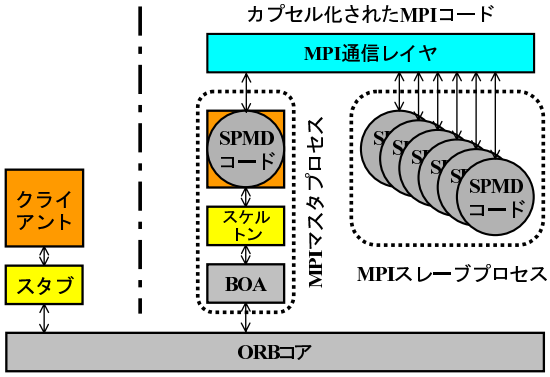


図 2 マスタ/スレーブ方式
Fig. 2 Master/Slave method.

この問題を解決するため，我々は，SPMD プログラムの各々のプロセスを 1 つのオブジェクトとしてカプセル化し，さらに，これらのオブジェクト群を 1 つのオブジェクトとして扱える枠組みを提供する並列 CORBA オブジェクト（以下，これを並列オブジェクトと呼ぶ）を実現する並列 CORBA システムを開発した（**図 3** 参照）．すなわち，並列オブジェクトとは，同一 CORBA オブジェクトの集合であり，それぞれの CORBA オブジェクトが SPMD プログラムの各プロセスに対応する．さらに，クライアントが，サーバ側の並列オブジェクトに属する各々のオブジェクトと直接通信することができるように，スタブおよびスケルトンの機能を拡張した．このため，前述の方法と比較して，通信性能およびメモリ利用率をともに向上させることができる．

また，クライアント–サーバ間の煩雑な通信処理は，後述する拡張 IDL コンパイラが生成する，機能拡張されたスタブおよびスケルトンによって行われるので，クライアントは，サーバが通常の CORBA オブジェクトか並列オブジェクトなのかを意識する必要はない．

さらに，サーバ，クライアントは，いずれも並列オブジェクトとして実装することが可能である．すなわ

☆ サーバアプリケーションにアクセスするときに使われるアダプタ．

```
interface[4] MatrixOperations{
    const long SIZE = 100;
    typedef double Vector[ SIZE ];
    typedef double Matrix[ SIZE ][ SIZE ];
    void multiply(in  dist[BLOCK][*] Matrix A,
                  in  Vector B,
                  out dist[BLOCK] Vector C );
    void skal(in dist[BLOCK] Vector C,
              out csum double skal);
};
```

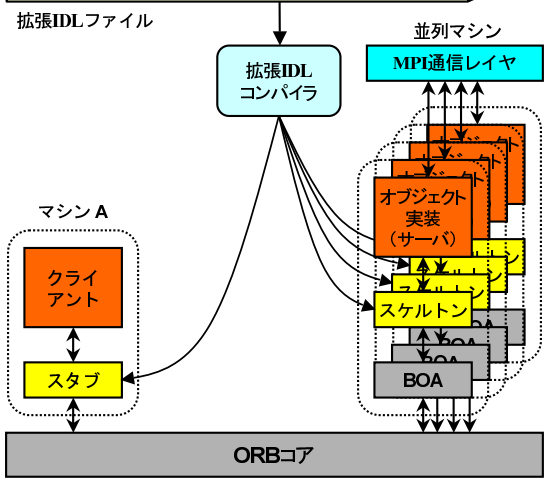


図 3 並列 CORBA システム
Fig. 3 Parallel CORBA system.

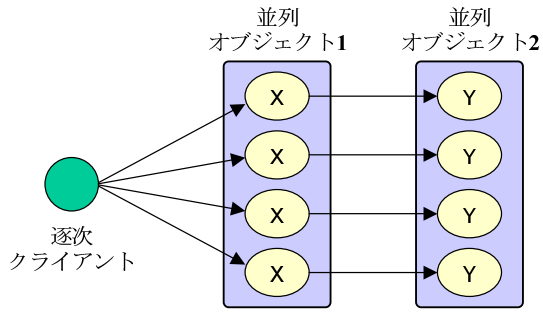


図 4 並列 CORBA のプログラミングモデル
Fig. 4 Parallel CORBA programming model.

ち，並列クライアント–並列サーバという構成も可能である．**図 4** に逐次クライアント，並列クライアントおよび並列サーバからなる並列 CORBA のプログラミングモデルを示す．この場合，並列オブジェクト 1 はクライアントとサーバの 2 つの機能をあわせ持つことになる．つまり，逐次クライアントから見ればサーバであり，並列オブジェクト 2 から見ればクライアントである．

このモデルを用いることにより，複数の異なる並列マシンがネットワーク接続された環境において，タスク並列処理とデータ並列処理を組み合わせた高性能並列分散コンピューティングの実現が可能となる．すなわち，並列化されたアプリケーションプログラムを関数など複数の機能単位（タスク）に分割し，それぞれを異なる並列マシンに割り付けることや，あらかじめ並列化された複数のプログラムを組み合わせてサーバプログラムを構築することが可能となる．

図 4 は，並列オブジェクト 1 に実装された関数X から並列オブジェクト 2 に実装された関数Y を呼び出している様子を表しており，関数X 内の他の処理と関数Y の呼び出しが独立に実行できる場合はタスク並列実行による高速化が可能である．また，関数X と関数Y に依存関係がある場合でも，規則処理を行う関数をベクトル型並列マシンで，不規則処理を行う関数をスカラ型並列マシンで実行するなど，各並列マシンの特長を活かすことによる高速化が期待できる．

並列 CORBA の機能を実現するために，我々は，2 章で説明した CORBA の標準仕様のうち，クライアント–サーバ間通信のインタフェース部分（IDL，スタブ，スケルトン），およびオブジェクトサービスの 1 つであり，名前管理の機能を提供する NamingService の拡張を行った．ただし，ORB の仕様は変更していないため，並列 CORBA システムでは，並列オブジェクトと通常の CORBA オブジェクトの共存が可能である．以下，拡張部分の詳細を述べる．

3.1 IDL の仕様拡張

サーバが並列オブジェクトである場合のサーバの並列度（オブジェクト数），パラメータ配列のサーバ上での分散方法，および返り値のリダクション演算の記述ができるように IDL の仕様を拡張した．

- **並列性の記述**

  `interface` キーワードを拡張して，並列オブジェクトのオブジェクト数，およびその形状を記述できるようにした．たとえば，図 3 の拡張 IDL 中の `interface[4]` は，サーバを 4 つのオブジェクトで実行することを意味する．オブジェクト数には ‘*’ を用いることも可能であり，`interface[*]` と記述すると，実行時にオブジェクト数が確定されることを意味する（コンパイル時には不明）．並列オブジェクトは MPI プログラムであり，`mpirun` コマンドで起動される．オブジェクト数が指定されている場合は，拡張 IDL コンパイラは，指定されたオブジェクト数のみに対応した制御コードをスタブあるいはスケルトンソースに埋め込むため，`mpirun` の引数で与えるプロセス数は拡張 IDL ファイルで指定したオブジェクト数と同じ値でなければならない．一方，‘*’ が指定された場合は，拡張 IDL コンパイラは，実行時にオブジェクト

数を取得する制御コードを生成するため，mpirun の引数で与えるプロセス数は任意の値でよい．

- **入出力パラメータに対するデータ分散方法の記述**
  クライアント–サーバ間のインタフェースとなるパラメータ配列の並列オブジェクト内での分散方法を，新たに導入したキーワード dist を用いて記述する．現在サポートしている分散形式はBLOCK と CYCLIC であり，HPF（High Performance Fortran）[8] と同様の方法で指定する．分散が指定されていない場合は，非分散配列と見なされる．図 3 の例では，関数multiply のパラメータである配列A は 1 次元目でBLOCK 分散，配列B は非分散，配列C はBLOCK 分散である．
- **リダクション演算の記述**
  サーバからのスカラの返り値に対するリダクション演算の指定を行う．現在指定可能な演算は，総和，最大，最小，積和，ビット演算である．図 3 中の拡張 IDL ファイルでは，関数skal の出力パラメータskal に対して総和のリダクション演算が指示されている．この場合，スタブで並列サーバ中の各オブジェクトが返すローカルな総和値からグローバルな総和値を計算し，結果をクライアントに返す処理を行う．

3.2 スタブおよびスケルトンの機能拡張

前述の拡張した IDL の仕様に基づいて，我々は拡張 IDL コンパイラを開発した．我々が開発した拡張 IDL コンパイラは，拡張 IDL ファイルを受理し，分散データの処理，および並列オブジェクトに属する全オブジェクトの遠隔起動処理を行えるように機能拡張したスタブとスケルトンを生成する．

ここでは，**図 5** に示す逐次クライアントと並列サーバの構成例を用いて，並列 CORBA システムにおける，クライアント，スタブ，スケルトン，サーバのそれぞれの基本動作を説明する．

- クライアント
  後述する拡張された NamingService の機能を用いて並列オブジェクトのリファレンスを取得し（例ではpaco），並列オブジェクト上に実装された関数foo を呼び出す．
- スタブ
  まず，拡張 IDL ファイルに記述された分散方法に従って，パラメータ配列A を分割して部分配列A’ を生成し，次に，サーバに属するすべてのオブジェクトに対してそれぞれリクエストを生成・送信する．サーバから返り値がある場合には，各オブジェクトから受信した部分配列A’ からオリジナルのパラメータ配列A を生成し，クライアントに返す．
- スケルトン
  スタブからリクエストを受信した後，関数foo を起動し，分割されたパラメータ配列A’ を渡す．foo の実行終了後，A’ をスタブに送り返す．
- サーバ
  パラメータ配列A の部分配列A’ を入力とし，関数foo を並列実行する．なお，4 つのサーバオブジェクト間の通信は MPI により行われる．

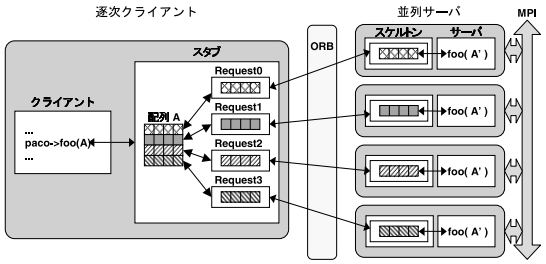


図 5 逐次クライアント/並列サーバ構成
Fig. 5 Sequential client and parallel server.

逆に，クライアントが並列オブジェクト，サーバが逐次オブジェクトという構成も可能である．この場合は，クライアント側の 1 つの代表オブジェクトのスタブが，分散されたパラメータ配列を他のオブジェクトから MPI 通信を用いて収集した後，サーバの起動処理を行う．

クライアント，サーバがともに並列オブジェクトである場合には，前述の処理に加えて，クライアント–サーバ間でパラメータ配列の再分散が必要になる場合が生じる．この再分散処理については次章で詳細に述べる．

3.3 NamingService の拡張

NamingService は，オブジェクトを名前で管理するためのオブジェクトサービスの 1 つであり，クライアントは NamingService を利用して，オブジェクト名からオブジェクトリファレンスを取得し，取得したオブジェクトリファレンスを用いて目的のサーバオブジェクトに実装された関数を呼び出す．

複数のオブジェクトからなる並列オブジェクトを 1 つのサーバオブジェクトとして扱えるようにするためには，オブジェクト群に対して 1 つの名前を対応付ける必要がある．しかしながら，通常の NamingService では，1 つのオブジェクトに対しては 1 つの名前しか対応付けることができない．そこで，我々は，オブジェクト群に対しても 1 つの名前を対応付けられるように NamingService を拡張した．

```
module CosNaming {
 ...
 interface NamingContext {
  ...
  typedef sequence<Object> ObjectCollection;
  void join_collection(in Name n,in Object obj);
  void leave_collection(in Name n,in Object obj);
  ObjectCollection resolve_collection(in Name n);
 };
};
```

図 6 並列オブジェクト用の NamingService オペレーション
Fig. 6 NamingService operations for a parallel object.

表 1 並列オブジェクト用 NamingService オペレーションの概要
Table 1 Overview of NamingService operations for a parallel object.

オペレーション	処理概要
join_collection（bind に相当）	並列オブジェクト名とオブジェクトリファレンスの登録
leave_collection（unbind に相当）	並列オブジェクト名とオブジェクトリファレンスの削除
resolve_collection（resolve に相当）	並列オブジェクト名からオブジェクトリファレンスを取得

具体的には，**図 6** に示すように，NamingService のオペレーションに，新たに`join_collection`，`leave_collection`，`resolve_collection` の 3 種のオペレーションを追加した．これらは，それぞれ通常のオブジェクト用に用意されている，`bind`，`unbind`，`resolve` に相当する機能を提供するもので，これらを用いて並列オブジェクト名の登録，削除，リファレンスの検索を行えるようにした．

表 1 に並列オブジェクト用に実装した NamingService のオペレーションの処理概要を示す．また，**図 7** にサーバが並列オブジェクトを`Matrix` という名前で NamingService に登録・削除する例を，**図 8** にクライアントが名前から並列オブジェクトのリファレンスを取得する例をそれぞれ示す．

4. 並列 CORBA におけるデータ再分散機能の実現

本章では，遠隔起動処理にともなって，並列クライアント–並列サーバ間で送受信される分散パラメータ配列の再分散機能について述べる．

4.1 データ再分散の必要性

並列 CORBA システムでは，各並列オブジェクト（並列クライアント/サーバ）に対して，データの分散方法をそれぞれ指定することができる．したがって，

```
// インスタンスの生成
MatrixOps_impl* obj = new MatrixOps_impl();
// オブジェクトリファレンスを登録
NamingService->join_collection(Matrix,obj);
...
// オブジェクトリファレンスを削除
NamingService->leave_collection(Matrix,obj);
```

図 7 並列オブジェクトの登録・削除操作（サーバ側）
Fig. 7 Creating and removing a binding of a parallel object (server side).

```
// オブジェクトリファレンスの獲得
objs =
   NamingService->resolve_collection(Matrix);
srv = MatrixOps::_narrow(objs);  // クラス変換
...
srv->MatVect(A,B,C);  //オペレーションの呼出
```

図 8 並列オブジェクトのリファレンス取得操作（クライアント側）
Fig. 8 Resolve operation for a parallel object (client side).

並列クライアントにおけるパラメータ配列の分散方法と並列サーバにおける分散方法が異なる場合，あるいは分散方法は同一であっても，並列クライアントと並列サーバでオブジェクト数が異なる場合は，パラメータ配列の送受信時に再分散を行う必要がある．たとえば，並列クライアントでは 2 次元配列の 1 次元目が `BLOCK` 分散されており，並列サーバでは 2 次元目が `BLOCK` 分散されている場合，並列クライアントがこの配列を入力パラメータ（クライアントからサーバへ転送するパラメータ）とする並列サーバの関数を起動する際には，当該関数が実行される前にパラメータ配列を‘`[BLOCK][*]`’から‘`[*][BLOCK]`’へと再分散する必要がある．

並列オブジェクトどうしのカップリングを容易にするためには，CORBA システムがこの再分散の機能を提供しなければならない．すなわち，遠隔起動メカニズムの一部として，分散されたパラメータ配列の再分散機能を実現する必要がある．

4.2 データ再分散機能の実現

分散されたパラメータ配列を再分散する方法はいくつか考えられるが，最も容易に再分散を実現する方法は，分散パラメータの送信元および送信先の 1 つの代表オブジェクトでそれぞれ分散データを収集/分配する方法である（以下，gather/scatter 方式と呼ぶ）．**図 9** に gather/scatter 方式で入力パラメータ配列を

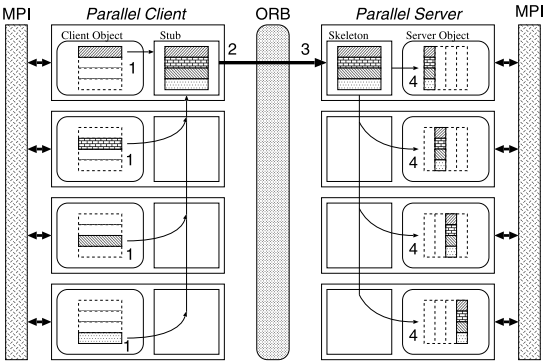


図 9 Gather/Scatter 方式による再分散
Fig. 9 Data redistribution in a gather/scatter approach.

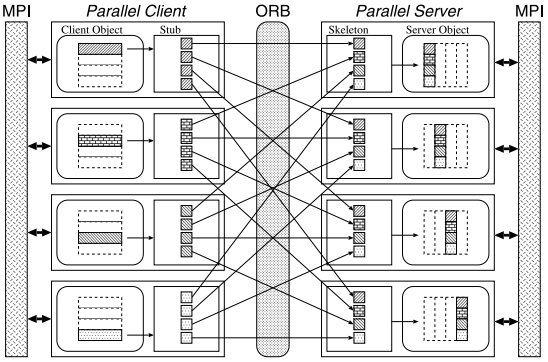


図 10 ORB 上での再分散
Fig. 10 Data redistribution over CORBA ORB.

再分散する手順を示す.

(1) MPI 通信により並列クライアント中の 1 つのオブジェクトに全分散データを収集する.
(2) データを収集したオブジェクトが ORB を介して並列サーバを起動し，並列サーバ中の 1 つのオブジェクトにパラメータを送信する.
(3) 起動されたサーバオブジェクトのスケルトンがパラメータを受信する.
(4) スケルトンが残りのオブジェクトを起動し，受信したパラメータを MPI 通信により分配する.

本方式は，実装は容易ではあるが，マスタ/スレーブ方式と同様に，分散パラメータを収集/分配するオブジェクトでクライアント–サーバ間の転送が逐次化されるため，並列 CORBA システムの利点を活かすことができない．さらに，分散パラメータをいったん収集するため，メモリ消費量が著しいという問題がある.

この問題を解決するためには，並列遠隔起動処理と分散パラメータの再分散処理を統合する必要がある．その方法の 1 つとして，並列クライアントと並列サーバの各オブジェクトどうしが，当該オブジェクトが所有すべき分散データを ORB を介して直接送受信する方法が考えられる（**図 10** 参照）．本方式では，データの総送受信量は最小化されるものの，gather/scatter 方式と比較して ORB による通信回数が増加する．一般に，ORB 通信はオーバヘッドが大きいため，性能劣化が予想される．さらに，サーバは 1 度の起動操作で複数のリクエストを受付けることができないため，ORB プロトコルを変更しなくてはならないことから，本方式は現実的ではない.

そこで，我々は，上記の問題を解決するため，並列クライアントから並列サーバにパラメータを転送する場合は，パラメータがクライアントからサーバに送られる前にクライアント側が，あるいは送られた後でサーバ側が当該パラメータの再分散を行う方法を採用した.

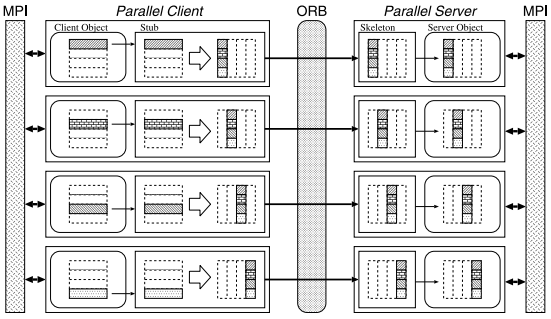


図 11 クライアント側での再分散
Fig. 11 Data redistribution in a client stub.

具体的には，クライアント側ではスタブに，サーバ側ではスケルトンに再分散機能を実装した．**図 11** は入力パラメータ配列をクライアント側で再分散する様子を示したものである．なお，再分散は，ORB を介してではなく，MPI 通信により高速に実行される.

再分散機能をスタブ，スケルトン双方に実装した理由は，クライアント，サーバが走行しているそれぞれの並列マシンの性能差などを考慮し，再分散を高速に実行できるマシンで再分散を実行させることにより，最小時間で再分散処理を行うことを可能とするためである．現在，クライアント，サーバのどちらで再分散を行うかは拡張 IDL コンパイラのオプションにより，ユーザがコンパイル時に選択する．すなわち，スタブで再分散を行う場合はオプション '`--pco-stub`' を，スケルトンで再分散を行う場合はオプション '`--pco-skel`' を指定する．拡張 IDL コンパイラは，これらのオプションに従って，生成する拡張スタブあるいは拡張スケルトンのソースコード中に再分散処理用のコードを埋め込む．なお，これらのオプションが明示的に指定されなかった場合は，'`--pco-stub`' が指定されたもの

と見なされる．

4.3 実　　装

前述した並列オブジェクト間でのデータ再分散を実現するために，並列CORBAシステムを拡張した．具体的には，データ再分散を実現するためのスタブおよびスケルトンの機能拡張，拡張スタブおよび拡張スケルトンコードを生成するためのIDLコンパイラの拡張，およびデータ再分散を実現する実行時ライブラリ関数の実装を行った．ここでは，これらの実現方法の詳細について述べる．

4.3.1 スタブおよびスケルトンにおける再分散処理

4.3.1.1 スタブでデータ再分散を実行する場合

拡張スタブが，クライアント側とサーバ側のデータ分散情報，および並列サーバのオブジェクト数を比較して再分散の必要があるか否かを判定し，必要な場合には，サーバ側の分散方法に適応するように再分散ライブラリを利用して再分散を行う．そして，各クライアントオブジェクトに対応するサーバオブジェクトに対して，並列起動処理を行う．

4.3.1.2 スケルトンでデータ再分散を実行する場合

再分散処理に関しては，拡張スケルトンは拡張スタブと同様の処理を行う．ただし，クライアント側ではコンパイル時に拡張IDLファイルからサーバの分散情報を取得することができるが，サーバ側ではクライアントの分散情報をコンパイル時に取得できないため，拡張スタブが，起動時にパラメータ配列の分散情報（分散方法，オブジェクト数など）を拡張スケルトンに送信する．

4.3.2 データ再分散ライブラリの実装

並列CORBAシステムに求められるデータ再分散機能は，HPFにおけるデータ再分散と同等であるため，既存のHPF処理系の実行時ライブラリを利用することが可能である．そこで，NECが開発したHPF処理系[9]の実行時ライブラリをベースにして並列CORBAシステム用のデータ再分散ライブラリを構築した．

本HPF処理系は，HPF-1.1で定められたデータの分散方法をすべてサポートしている．並列CORBAシステムで記述可能なパラメータの分散方法は，HPF-1.1で記述可能な分散方法のサブセットであるため，本再分散ライブラリで並列CORBAシステムで必要となる再分散パターンをすべてカバーすることができる．

ただし，本ライブラリを用いた場合，並列クライアントのオブジェクト数と並列サーバのオブジェクト数が異なる場合には，オブジェクト数が多いサイドでしか再分散を実行することができないという制限が生じる．これは，本HPFライブラリには，実行中にプロセッサ数を増加させる機能がないためである．

4.3.3 クライアントオブジェクトとサーバオブジェクトの対応付け

並列クライアントと並列サーバのオブジェクト数が等しい場合には，それぞれオブジェクト番号が等しいオブジェクトどうしが対応付けられるが，異なる場合には，以下のように対応付けを行う．

4.3.3.1 クライアントオブジェクト数 > サーバオブジェクト数の場合

それぞれオブジェクト番号が等しいオブジェクトどうしを対応付ける．対応するサーバオブジェクトが存在しないクライアントオブジェクトは，サーバへのリクエストは送出せず，サーバオブジェクトの呼び出し処理は行わない．なお，再分散処理はクライアント側で行われる．

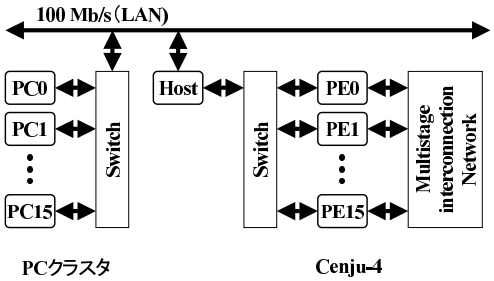
4.3.3.2 クライアントオブジェクト数 < サーバオブジェクト数の場合

サーバオブジェクトをクライアントオブジェクトにラウンドロビン方式で番号順に割り付ける．本方式により，まず，それぞれ同一番号のオブジェクトどうしが対応付けられる．これは，前述のとおり，サーバオブジェクト数がクライアントオブジェクト数より多い場合には，サーバ側でデータ再分散を行わなければならないためである．つまり，各クライアントオブジェクトに割り付けられた分散データを，同一番号のサーバオブジェクトに送信しなければならないため，クライアントオブジェクトに対して，同一番号のサーバオブジェクトが必ず対応付けられるようにする必要がある．また，ラウンドロビン方式を採用したもう1つの理由は，各クライアントオブジェクトに対応付けられるサーバオブジェクト数をできるだけ平均化することで，各クライアントオブジェクトにおけるサーバオブジェクトの起動時間を平均化し，結果としてサーバの呼び出しに要する総時間を最小化するためである．

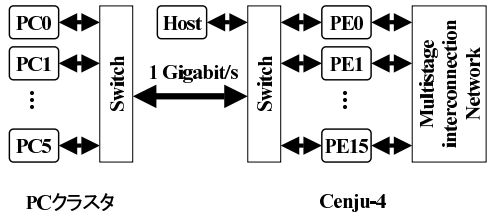
5. 評 価 結 果

5.1 実 験 環 境

並列CORBAシステムにおけるデータ再分散機能の評価を行うため，フロントエンドであるホストEWSと16台のPE（Processing Element）で構成される並列コンピュータNEC Cenju-4[10]，および16台構成のPCクラスタの2種類の並列マシンを接続した異機種並列分散計算機システムを構築した．Cenju-4の各PEは200 MHzのVR10000，256 MBのメモリ，1 MBの2次キャッシュを備え，最大200 MB/sの多段ネットワークと100 Mb/sのイーサネットワークで



(a) PC クラスタ/Cenju-4 を 100 Mb/s で接続（実験 1）



(b) PC クラスタ/Cenju-4 を 1 Giga b/s で接続（実験 2）

図 12 実験環境
Fig. 12 Experimental environment.

相互に接続される．Cenju-4 では，プログラムはホスト EWS でコンパイルされ，ここから PE へジョブが投入される．一方，PC クラスタの各ノードは 2 つの 450 MHz PentiumIII を備え，100 Mb/s のイーサネットワークで相互に接続される．OS は Linux2.2.7 である．MPI は Cenju-4 では多段ネットワーク上に，PC クラスタでは 100 Mb/s のイーサネットワーク上に実装されている．**図 12** に実験環境を示す．PC クラスタと Cenju-4 の間を 100 Mb/s の LAN で接続したシステムと（図 12 (a) 参照），1 Gb/s で接続したシステム（図 12 (b) 参照）の異なる 2 種類の実験環境を構築した．

まず，我々が提案する再分散方式の評価を行うため，並列クライアントが，並列サーバに実装されたオペレーションを呼び出すという簡単なコードで実験を行った．オペレーションは入力属性のパラメータを 1 つ持ち，以下のように並列ループ内で配列要素に値を代入する処理を行うものである．プログラム中，`block_size` は各オブジェクトに割り当てられる配列サイズ，`my_id` はオブジェクト番号である．

```
for( int i1 = 0; i1 < block_size; i1++ ) {
    for( int i0 = 0; i0 < 2000; i0++ ) {
        out_array1[i0][i1] = my_id * k + 5;
    }
}
```

また，パラメータは 2000×2000 の 2 次元配列で，クライアント側は‘`[BLOCK][*]`’，サーバ側は‘`[*][BLOCK]`’の分散方法を指定した．したがって，関数が起動される前にパラメータ配列の再分散が実行される．

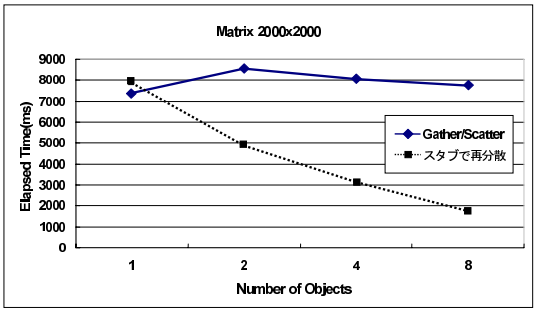


図 13 Gather/Scatter 方式との比較
Fig. 13 Comparison to gather/scatter approach.

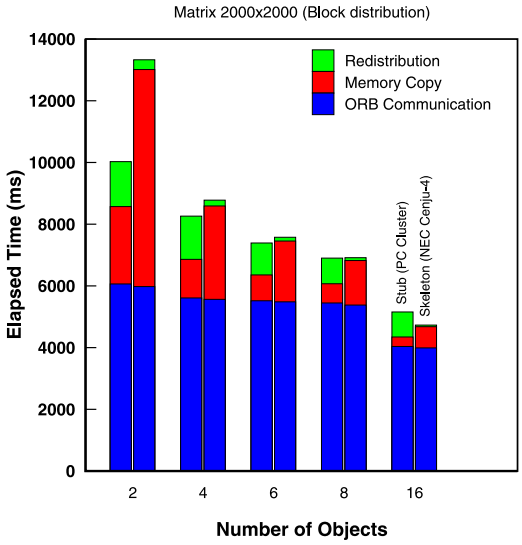
5.2 gather/scatter 方式との比較

本実験は，我々が提案した再分散方式と gather/scatter 方式との比較を行うことを目的とするため，同数のオブジェクトからなる並列クライアント，並列サーバをともに PC クラスタで動作させ，gather 処理を行うクライアントオブジェクト 0 において，サーバオブジェクトに実装された関数の呼び出しに要した時間を計測した．具体的には，クライアントプログラム内の関数呼び出しコードの前後に時間計測ルーチンを挿入し，関数を呼び出してから制御が戻ってくるまでの時間を計測した．なお，1 台の PC には 1 つのオブジェクトのみを走行させた．

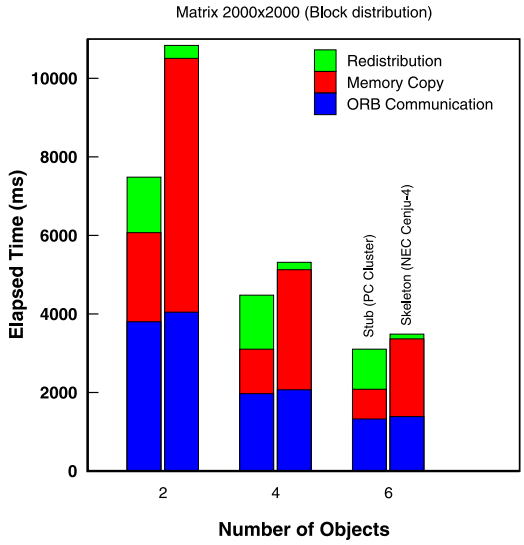
結果を**図 13** に示す．スタブで再分散を行った場合，並列オブジェクト内の各オブジェクトが，独立に呼び出し処理を行うことができるため，オブジェクト数が増加するに従って，呼び出し処理にかかる時間が減少しており，並列 CORBA システムの効果が確認できる．一方，gather/scatter 方式では分散パラメータを収集/分配するオブジェクトでクライアント–サーバ間の転送が逐次化されるため，並列 CORBA システムの優位性がまったく活かされていない．

また，スケルトン（サーバ側）で再分散を行う実験も行ったが，スタブで再分散を行った場合とほぼ同じ結果が得られた．スケルトンで再分散を行う場合は，パラメータ配列に加えて，クライアント側での分散情報もサーバへ送信するが，配列サイズと比較して分散情報のデータが十分小さいためと考えられる．

gather/scatter 方式では，1 台で実行した場合と比較して 2 台で実行した場合の方が実行時間が増加しているが，これは，1 台で実行する場合には gather/scatter 処理を行わないためである．また，2 台以降，台数が増加するに従って性能が向上しているが，これはサー



(a) 実験 1 の結果（100 Mb/s で接続）



(b) 実験 2 の結果（1 Gb/s で接続）

図 14 実験結果
Fig. 14 Experimental results.

バプログラムの並列化の効果のためであると思われる．1 台で実行した場合に，gather/scatter 方式よりも再分散方式の方が性能が悪くなっているが，これは，gather/scatter 方式と比較して，再分散方式における再分散処理用の制御コードが複雑であり，1 台で実行する場合には，結果的には再分散処理は行わないものの，gather/scatter 方式と比較してオーバヘッドが大きいためと考えられる．

5.3 異機種システム上での評価

5.3.1 サーバ側での再分散

PC クラスタと Cenju-4 にそれぞれ並列クライアント，並列サーバを走行させ，分散パラメータ配列のデータ再分散をクライアント側（PC クラスタ）で行った場合とサーバ側（Cenju-4）で行った場合の性能比較を，図 12 (a)，(b) のそれぞれの実験環境で行った．

5.3.1.1 実 験 1

まず，通常の LAN 環境をベースとして PC クラスタと Cenju-4 を接続した図 12 (a) の環境で実験を行った．本実験環境では，PC クラスタと Cenju-4 の PE 間の ORB 通信は 100 Mb/s のイーサネットを介して行われ，さらに Cenju-4 のホスト EWS を経由する．また，Cenju-4 の PE ⇔ EWS 間は 100 Mb/s で接続されている．PE がホスト EWS を経由して LAN に接続されているのは，通常の Cenju-4 の運用環境では，ホスト EWS と PE 群はプライベートネットワークに属しており，PE と外部のマシンとの間で通信を行う場合は，ホスト EWS を経由せざるをえないためである．

結果を**図 14** (a) に示す．結果にはサーバでのオペレーションの実行時間は含まれていない．図中，Redistribution，ORB Communication，Memory Copy の意味は，それぞれ以下のとおりである．また，各オブジェクト数（プロセッサ数）ごとに 2 種類の結果を示しているが，左側がスタブ（PC クラスタ）で再分散を行った場合，右側がスケルトン（Cenju-4）で再分散を行った場合の結果である．

- ORB Communication：クライアントからサーバへデータを転送するのに要した時間．再分散およびメモリコピーの時間は含めない．
- Redistribution：スタブあるいはスケルトン内で‘`[BLOCK][*]`’から‘`[*][BLOCK]`’への再分散に要した時間．
- Memory Copy：再分散を行う際のメモリコピーに要した時間．

再分散については，Cenju-4 で実行した方がかなり良い性能が得られている．これは，Cenju-4 では専用の多段ネットワークにより再分散が実行されるが，PC クラスタでは，100 Mb/s のイーサネットワーク上で実行されるためである．しかしながら，再分散の実行時間と比較して 10〜15 倍もの時間を要しているメモリコピーの性能が PC クラスタと比較して 2 倍以上遅いため，結果として PC クラスタで再分散を行った方が性能が良くなっている．ただし，16 台の場合では，1 台あたりに割り当てられる配列サイズが小さく

なることから，メモリコピーに要する時間が短くなるため，Cenju-4 で再分散を行った方が性能が良くなっている．したがって，パラメータ配列のサイズが小さい場合には，より少ない台数においても，Cenju-4 で再分散を実行する方が有利になると予想される．

また，並列 CORBA システムでは，並列クライアント/サーバ内の各オブジェクトが各々独立に通信することが可能なため，オブジェクト数が増加すれば総 ORB 通信時間が減少するはずであるが，ORB 通信性能の台数効果がほとんど見られない．これは，本実験環境では，PC クラスタの各 PC と Cenju-4 の各 PE との通信はすべてホスト EWS を経由するため，EWS がボトルネックになっていること，また，PC クラスタ⇔ホスト EWS 間のイーサネットワークが占有状態でないことが原因と考えられる．

なお，データ再分散を行う際にメモリコピーが必要になるのは，以下の 2 つの理由による．

(1) 並列 CORBA システムで用いているデータ構造と再分散ライブラリで用いているデータ構造が異なる

並列 CORBA システムでは，分散された配列と，配列の分散情報を同時に格納可能な`darray`[4]と呼ぶデータ構造を用いている．これは，CORBA の`sequence`データ構造をベースにしたもので，2 次元配列以上の配列を実現する場合には，メモリ上でデータが連続にならない．一方，データ再分散ライブラリは HPF 用のライブラリをベースに作成したため，通常の配列構造を用いている．したがって，データ再分散ライブラリを呼び出す前に，`darray` →配列のデータ構造変換が必要となり，結果としてメモリコピーが必要になる．

(2) C++と Fortran でメモリマッピングスキームが異なる

データ再分散ライブラリは HPF 用のライブラリをベースに作成したため，内部では Fortran 形式の配列しか扱えない．一方，CORBA でサポートされている言語マッピングは C++，C，Java である．したがって，これらから Fortran 配列への変換が必要となり，結果として 1 要素ずつのメモリコピーが必要になる．

5.3.1.2 実 験 2

実験 1 と同じ実験を，図 12 (b) の環境で行った．すなわち，PC クラスタの各 PC と Cenju-4 の各 PE との間の ORB 通信を 1 Gb/s のリンク上で行った．さらに，本実験では，相互接続した PC 群と PE 群でプライベートネットワークを構成し，1 Gb/s リンクを占有できるようにした．ただし，Cenju-4 の各 PE とスイッチの間は，実験 1 と同じく 100 Mb/s で接続される．なお，図 12 (b) で PC クラスタの PC 数が 6 台となっているのは，Gigabit 対応のネットワークカードを 6 枚しか用意できなかったためである．このため，本環境ではクライアント，サーバともにオブジェクト数を 2，4，6 として実験を行った．結果を図 14 (b) に示す．PC クラスタの各 PC と Cenju-4 の各 PE 間の通信が EWS を介さなくなったことと，両者の間の接続リンクが 1 Gb/s になったことで，ORB 通信の性能が向上したと同時に，ORB 通信時間に台数効果が確認できる．このように，並列マシン間の接続リンクが高速になるほど，並列クライアント/ サーバ内の各オブジェクトが各々独立に通信することができる並列 CORBA システムの優位性が顕著になる．一方，ORB を介した通信時間と比較して，再分散のための処理時間（メモリコピーとデータ再分散）の影響が顕著になっている．特に，パラメータ配列のデータサイズが大きくなるほど，メモリコピーのオーバヘッドが大きくなるため，C++形式の配列を扱えるようにして本来不必要なメモリコピーをなくすなど，再分散ライブラリを改善する必要がある．

6. 関 連 研 究

群馬大学では，イメージプロセッシングをターゲットとして，CORBA をベースに並列処理環境の構築を行っており，PVM で記述した並列プログラムのマスタのみを CORBA オブジェクトとして実装する方式と，マスタおよび複数のスレーブの各々を CORBA オブジェクトとして実装する 2 種類の方式を提案している[11]．前者は，本稿で述べたマスタ/スレーブ方式に相当する．いずれの方式も，CORBA を拡張した我々の方式とは異なり，クライアントが並列プログラムのプロセスと直接通信することができないため，マスタがボトルネックになるという欠点がある．

我々と同様のアプローチでは，インディアナ大学の PARDIS[12]がある．PARDIS は，CORBA オブジェクトを拡張した`SPMD object`と呼ぶオブジェクトを新たに導入し，SPMD プログラムを CORBA オブジェクトとして実装可能としているが，並列オブジェクト間のデータ再分散は考慮されていない．

CORBA 同様，RPC 型のグローバルコンピューティングシステムとしては，Ninf[13]や Netsolve[14]が代表的である．これらは，数値計算に特化した実装を行っているものの，並列 CORBA のように，並列プログラムを効率良く実装する方法については考慮されていない．しかしながら，我々が開発した並列 CORBA システムで採用した方式は，Ninf などの他の RPC ベー

スのシステムにおいても適用可能と考えられる．

7. まとめと今後の課題

以上，我々が開発した並列 CORBA システム，特に並列 CORBA におけるデータ分散管理の実装について述べ，Cenju-4 と PC クラスタからなる異機種並列分散システムを用いた評価結果によりその有効性を示した．

並列 CORBA により，並列プログラムを効率良く CORBA オブジェクトにカプセル化することができ，高性能並列分散コンピューティング環境を容易に構築することが可能となる．並列クライアント–並列サーバ間で必要となるパラメータ配列の再分散機能は，スタブ（クライアント側），スケルトン（サーバ側）の両方に実装し，コンパイル時にどちらで再分散を行うか指定可能とした．これにより，並列分散コンピューティング環境下で，データを再分散させるために必要となる処理が最も高速に行える並列マシン上で，データ再分散を実行させることができる．

このように，プログラマによる制御を可能とすることは重要であるが，一般に，どのマシンが最も高速に再分散を実行できるかを判断するのは難しい．これは，プログラマが，データ再分散処理や各々の並列マシンの特性を把握しておく必要があり，さらに，サーバが動作している並列マシンの負荷の変化や，サーバが他の並列マシンに移動する場合など，これらの特性が実行時に変化する可能性があるためである．この問題を解決するため，実行時に各マシンのシステム情報や，負荷情報などを収集・管理するサービスを実現し，これを利用して，データ再分散を実行するのに最適なマシンを自動的に選択するシステムを設計する予定である．また，データ再分散時にメモリコピーを必要としない高速なデータ再分散ライブラリを開発する予定である．

謝辞 本研究をサポートしていただいた NEC ラボラトリーズ 後藤支配人，同インターネットシステム研究所中田統括マネージャに感謝いたします．また，数々の議論を通じて有益な助言をいただいた IRISA/INRIA の CAPS チームメンバに感謝いたします．

参 考 文 献

1) Foster, I. and Kesselman, C. (Eds.): *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Inc (1998).
2) Foster, I. and Karonis, N.T.: A Grid-Enabled MPI: Message passing in heterogeneous distributed computing systems, *Supercomputing'98* (1998).
3) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *The International Journal of Supercomputer Applications and High Performance Computing*, Vol.11, No.2, pp.115–128 (1997).
4) René, C. and Priol, T.: MPI Code Encapsulating using Parallel CORBA Object, *Proc. 8th IEEE International Symposium on High Performance Distributed Computing*, pp.3–10 (1999).
5) Kamachi, T., et al.: Data distribution for Parallel CORBA Objects, *Euro-Par 2000*, pp.1239–1249 (2000).
6) Puder, A.: The MICO CORBA Compliant System, *Dr Dobb's Journal*, Vol.23, No.11, pp.44–51 (1998).
7) OMG: The Common Object Request Broker: Architecture and Specification, Version 2.0, Technical Report PTC/96-03-04, Object Management Group (1996).
8) High Performance Fortran Forum: High Performance Fortran Language Specification, Version 1.0 (1993).
9) 蒲池ほか：HPF 処理系の実現と評価，情報処理学会論文誌，Vol.37, No.7, pp.1255–1264 (1996).
10) Nakata, T., Kanoh, Y., Tatsukawa, K., Yanagida, S., Nishi, N. and Takayama, H.: Architecture and Software Environment of Parallel Computer Cenju-4, *NEC Research & Development*, Vol.39, No.4, pp.385–390 (1998).
11) Aritsugi, M., Fukatsu, H. and Kanamori, Y.: Implementation of parallel image convolution processing based on CORBA, *Trans. IPS. Japan*, Vol.41, No.2, pp.488–497 (2000).
12) Keahey, K. and Gannon, D.: PARDIS: A Parallel Approach to CORBA, *Supercomputing'97* (1997).
13) Sekiguchi, S., Sato, M., H.N. and Nagashima, U.: – Ninf – : Network base information library for globally high performance computing, *Parallel Object-Oriented Methods and Applications* (*POOMA*) (1996).
14) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Supercomputing'96* (1996).

(平成 13 年 9 月 3 日受付)
(平成 14 年 2 月 13 日採録)

蒲池 恒彦（正会員）

1964 年生．1988 年九州大学工学部情報工学科卒業．1990 年九州大学大学院総合理工学研究課情報システム学専攻修了．同年 NEC 入社．現在，インターネットシステム研究所主任．並列計算機アーキテクチャ，自動並列化コンパイラ，並列化支援システム等に興味を持つ．

Thierry Priol

Dr. T. Priol received a Ph.D. degree in 1989 and an Habilitation in 1995 in Computer Science from the University of Rennes. Since 1989, he has held a researcher position at INRIA. He has a Research Director position at INRIA since 1995. His research interests are parallel and distributed systems, Grid computing, Distributed Shared Memory (DSM) and High performance visualisation.