仮想化概念に基づく分散処理フロー制御システムの実現

菊池一彦[†] 菅沼拓夫[††]
白鳥則郎[††] 宮崎正俊[†††]

複数の計算機上のプログラムを連携させて利用する場合，アクセス透過性とネットワーク透過性を備えたシステムを利用することで，ユーザは個々のプログラムの仕様差やネットワーク上の位置を意識せずに利用することできる．しかし利用する計算機やプログラムの数や種類が増加するに従い，これらを連携させる分散処理プログラムの量や複雑度が増加し，またユーザが持つべき計算資源に関する知識の量も増加する．本システムは，透過的に扱う複数の計算資源を，仮想的に1つの計算資源として定義し利用するための機能をユーザに提供する．これにより個々の計算処理に適した仮想的な計算資源を実現し分散処理の記述において利用することにより，分散処理の記述量と記述難易度を低くすることが可能となる．本システムを計算機間通信機能を持つシステム（Java RMIなど）上に実装し，仮想的な計算資源を用いることによって，分散処理の記述性が向上することを確認した．

Implementation of the Distributed Processing Flow Control System Based on Virtualization

Kazuhiko Kikuchi,[†] Takuo Suganuma,[††] Norio Shiratori[††]
and Masatoshi Miyazaki[†††]

Users can use the computational resources without care about the resources that have each specification and location in network by using a system that has the access transparency and the network transparency. But increasing the number and the kind of computers and programs increases the complexity and steps of the distributed processing program. And it requires more knowledge of the computational resources which users should have. The distributed processing flow control system provides the function to define and use the virtual computational resources, composed of several real resources. It becomes possible to describe a distributed processing flow by using the virtual resources, suitable for each processing. We developed this system on the distribution support system (e.g., Java RMI) and confirmed the improvement of the ability to describe the distributed processing flow.

1. はじめに

インターネットに代表されるネットワーク環境が一般的に利用されるようになり，単一のプログラムで処理される作業だけでなく，複数の作業から構成される一連の作業全体を，いくつかのプログラムを組み合わせて利用することによって，ユーザの思考や操作を途切れさせることなく処理することが可能となってきた．

多様な計算機から構成される分散環境上で，計算資源であるプログラムやデータ，計算機などを複数組み合わせて処理の流れを表現したものを，本論文では分散処理フローと呼ぶ．分散処理フローを表現する場合，計算資源ごとに異なる仕様や，これらのネットワーク上の位置を把握しながら利用する必要があるが，これらは，アクセス透過性やネットワーク透過性[1),2)]を備えたシステムであるCORBA[3)]やJava RMI[4)]，Globus Toolkit[5)]などを利用することで，異なる計算機上の計算資源であることを意識することなく操作することができるようになってきた．また，一般に分散環境上には多種多様な特性を持った計算資源が混在し，またそれらは時間の経過とともに状態（たとえば計算機の稼動状態や負荷など）が変化する可能性がある．よって分散処理を効率良く行うためには，複数の計算資源の中からプログラムの実行に適した計算資源を選択し

† 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University
†† 東北大学電気通信研究所
Research Institute of Electrical Communication, Tohoku University
††† 岩手県立大学ソフトウェア情報学部
Faculty of Software and Information Science, Iwate Prefectural University

たり，分散環境の動的な状態変化に対応した計算資源の再割当てなどが行われる必要があり，これらを実現するものとして，Condor[6]や Legion[7]，AppLeS[8]などが研究されている．

これら分散処理環境を利用することにより，分散処理の高パフォーマンス化が実現されてきているが，その利用においては，ユーザは計算資源の物理的な構成や特性（CPU 性能やメモリ量など）を十分把握したうえで実行するプログラムに適した資源の割当てなどを行う必要がある．そのため計算資源に関する知識をあまり持たないエンドユーザは，その利用が困難である．特に分散環境の広域化が進むに従い，多数の計算資源の構成や個々の特性を把握することが困難になるため，この問題が顕在化してくる．

このような利用時の困難さを解決するためには，計算資源の物理的な構成や特性に合わせて分散処理フローの処理内容や構造を考えるのではなく，分散処理フローに適した構造や特性を持った計算資源を創出し，これを用いて分散処理フローを表現できる環境が必要である．ここで創出される計算資源は，実際に存在する計算資源ではなく，計算処理の内容や特性に合わせていくつかの実計算資源から構成されるユーザの概念的な計算資源であり，本論文ではこれを仮想計算資源と呼ぶ．

仮想計算資源を用いて分散処理フローを記述する場合，計算資源の物理的な構造や特性に左右されることなく処理フローが表現できるため，記述量や記述難易度の低減が可能である．たとえば，ある 1 つの作業を複数の計算資源を使って処理する場合，ユーザが作業を分割し各計算資源に割り付けるのではなく，複数の計算資源を概念的に 1 つの計算資源としてとらえた仮想計算資源に対し，作業を分割することなくそのまま割り付けて利用することにより，ユーザの操作量や分散処理フローの記述量を減らすことが可能である．

また，数値計算処理やグラフィック処理のように計算処理が何らかの特性を持つ場合，従来は，計算処理の特性に適した計算資源を選択し，その構造に合わせて分散処理フローを表現する必要があった．そのため，計算資源の物理的な構造や特性に関する知識をユーザが十分持っている必要があった．これに対して，計算処理の特性に合った仮想計算資源を利用した場合には，フローの記述において計算資源の物理的な構造や特性に合わせるための記述が不要になり，記述難易度を低くすることができる．

分散処理フローの記述を困難にする他の原因として“計算資源の不確定性”がある．計算資源の不確定性とは，実行時に利用する計算資源が，フローの記述時点では未確定である状態を指す．不確定性を持った計算資源の例としては，分散処理フローの実行時点で最も負荷の軽い計算機や，処理対象データの量によって分散数が変化するプログラムなどがあげられる．前者は実際に利用される計算機が未確定で，後者は実行されるプログラム数が未確定な例である．

従来，計算資源の不確定性は，ユーザがプログラムとして表現したり，資源割当て機能を持ったシステムを利用し，割当てルールを記述することによって解決していた．しかしこの方法は，分散処理フローの資源要求に基づき割当てルールを記述するため，分散環境上の計算資源の構成や状態が変化した場合に，その影響に応じて分散処理フローの方を変化させることが困難である．たとえば，処理対象データの量によって分散数が変化するプログラムは，プログラムが要求する分散数と処理の実行時に利用可能な計算機数を，相互に調整しながら実行時の分散数が決定され分散処理フローが構成されることが望ましいが，このような動作が困難である．また，不確定性を持った計算資源の組合せが増加するに従い，分散処理フローの記述難易度が増加してしまう問題がある．

これに対して，仮想計算資源に不確定性を内包しユーザから隠蔽することにより，動的に変化する実行時の状態を意識せずにフローが表現できるため，記述難易度を低減することができる．また，分散環境の状態変化に応じて，計算資源の組合せ情報から動的に分散処理フローを構成することが可能となる．

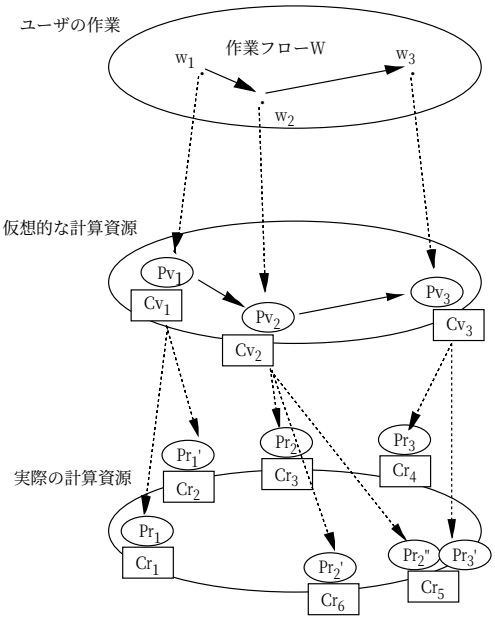
本論文では，計算資源の仮想化という概念を導入することによって，分散処理に適した特性を持つ仮想的な計算資源を生成し，これを用いて分散処理フローの表現と実行を可能にする分散処理環境の実現について述べる．

まず，2 章において仮想化の概念について述べる．次に 3 章で，仮想的な計算資源を表現するためのスクリプト構造化モデルについて述べる．4 章では，スクリプト構造化モデルに基づく分散処理スクリプトの文法について説明する．5 章では，スクリプト処理系の実装について述べ，6 章で実装システムの評価について述べる．7 章では関連研究との比較を行い，最後に 8 章でまとめと今後の研究課題について述べる．

2. 計算資源の仮想化

本章では仮想化概念の実現方法について述べる．

従来，ユーザは，作業単位 w から構成される一連の作業 W を処理するために，実際の計算資源である



W_i：利用者の作業
Pv_j：利用者の作業を処理可能な仮想プログラム
Cv_k：Pv_jを実行可能な仮想計算機
Pr_l：Pv_jを具象化した場合の実プログラム
Cr_m：Cv_kを具象化した場合の実計算機

図 1 ユーザの作業と仮想的な資源，実資源の関係
Fig. 1 Relations between users tasks, virtual resources and real resources.

プログラム Pr や計算機 Cr を使って，分散処理フローを記述している．

本研究では，仮想的な計算資源を扱う枠組みを実現するために，一連の作業 W と実際の計算資源 Pr，Cr の間に，仮想的な計算資源である Pv，Cv を定義するための層を設ける（**図 1**）．

仮想的な計算資源の層では，ユーザの作業の内容や特性，ユーザの計算資源のとらえ方などに合わせて，複数の実計算資源を組み合わせて，仮想的な計算資源を定義する．またこの定義では，計算処理を行う際に実計算資源を実行プログラムに割り当てるための条件を記述する．仮想計算資源には，実計算資源と同様に計算資源名を付け，実計算資源と同じように処理フローの定義で扱えるようにする．

このように，複数の計算資源を概念的に 1 つの計算資源としてとらえられるようにすることで，ユーザは分散処理フローの記述において，実際の計算資源の構成を意識して記述するのではなく，ユーザの作業単位をそのまま仮想的な計算資源に割り付けることで分散処理フローを表現することができる．これにより，計算資源の不確定性を処理フローから独立化させることができるため，ユーザは処理フローの記述において，実行時点の計算資源の状態を予測しながらフローを定義する必要がなくなる．また，定義された分散処理フローは，分散環境の状態変化へ動的に対応可能になる．

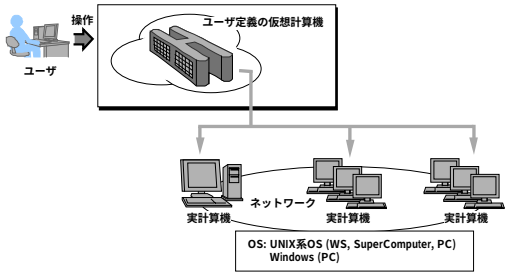


図 2 仮想的な計算資源を定義した例
Fig. 2 An example of virtual computational resource.

図 2 に，仮想的な計算資源を定義した例を示す．例では，複数の計算機から構成される 1 台の仮想的な計算機を定義している．ユーザは，仮想計算機が複数の実計算機から構成されていることを意識することなく，1 台の実計算機と同様に扱うことが可能である．また，仮想的な計算機の利用形態として，並列分散，ユーザ定義ルールによる分散，実行プログラムに適した計算機の選択利用などをいくつか設定しておき，仮想計算資源の利用において，ユーザの作業の特性に合わせて利用形態を選択することによって，計算機利用知識の少ないユーザでも，複数の実計算資源を効率良く利用することができる．また，計算資源の不確定性をユーザは意識する必要がなく，分散処理フローと実計算資源が直接関連付けられないため，実計算資源の構成が変化した場合でも，分散処理フローは影響を受けない．

3. 分散処理スクリプトの構造化モデル

本章では，仮想的な計算資源を用いて分散処理フローを記述するための分散処理スクリプトについて，仮想的な計算資源を表す要素と，これら要素から構成される分散処理スクリプトの構造について述べる．

3.1 スクリプトを構成する要素

複数の計算機上のプログラムやデータを用いて処理の流れを記述するためには，実行するプログラムと処理対象となるデータやパラメータに関する情報に加え，これらを処理する際に必要となる計算資源（実行計算機など）に関する情報を扱える必要がある．よって，分散処理スクリプトを構成する要素として，プログラムやデータを扱う要素に加え，リソースを扱うための要素を導入し，これら 3 要素でスクリプトを表現する．ここで，プログラムに関する情報とは，実行するプログラム名やプログラムの保管場所などである．ま

た，データに関する情報とは，処理対象データやパラメータデータ，およびこれらの保管場所などである．リソースに関する情報とは，プログラムを実行するために必要となる計算資源に関する情報であり，実行計算機名などである．このように，分散処理スクリプトを構成する 3 要素は，それぞれいくつかの情報を内包できる必要がある．この仕組みを実現するために，個々の要素をオブジェクトとしてとらえモデル化する．

3.2 スクリプト構造化モデル

仮想的な計算資源を用いたスクリプト構造を実現するために，オブジェクト指向概念に基づき，スクリプト構造化モデルを検討する．

仮想的な計算資源は，分散処理フローの定義の時点では仮想的なものとして扱われるが，処理の実行時点においては，実際に存在する 1 つ以上の計算資源が割り当てられ処理に用いられる．すなわち，仮想的な計算資源は，複数の実計算資源から構成される抽象的な計算資源であると考えることができる．よって，オブジェクト指向概念に基づいた場合，抽象クラスとインスタンスの関係を用いて，実際の計算資源をインスタンスオブジェクトとしてとらえ，仮想的な計算資源をクラスオブジェクトととらえることで構造をモデル化することができる．また，クラスオブジェクト化することによって，実際の計算資源へのアクセス方法を統一することになり，多様な計算資源を透過的に扱えるようにすることができる．よって，複数のオブジェクトを包含した抽象クラスを定義することによって，ユーザが仮想的な計算資源を定義することが可能となる．

計算資源の不確定性とは，分散処理スクリプトの記述時点では実行に用いられる計算資源が不明で，処理の実行時に決定される性質を指している．これは，仮想的な計算資源をクラスオブジェクトとして定義している場合，実際の計算資源であるインスタンスオブジェクトを生成する際に，分散環境の状態に応じて，システムが自動的に，または，ユーザの指定によって，インスタンスの生成方法を選択することにより実現することができる．この場合，仮想的な計算資源から実際に利用する計算資源を計算するためのルールをオブジェクトのメソッドとして表現する．これにより，計算資源の不確定性をオブジェクトの内部に隠蔽することが可能となる．

以上のようにプログラム，データ，リソースをオブジェクトクラス化することによって，仮想的な計算資源を実現することができる．次にこれら 3 要素を関連付け，構造化することによって，分散処理スクリプトを構成する方法について述べる．分散処理スクリプトを構成する 3 つの要素をオブジェクトクラス化したものを，それぞれプログラムオブジェクト，データオブジェクト，リソースオブジェクトと呼ぶ．

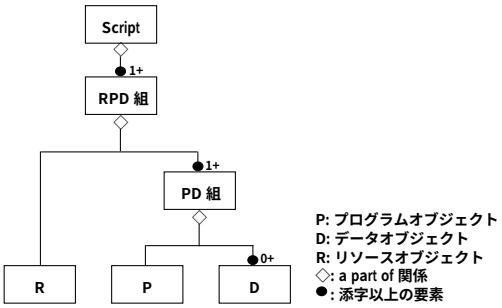


図 3 スクリプト構造化モデル（OMT 表現）
Fig. 3 A structured model for script.

一般に 1 つのプログラムは，0 個以上のデータ（入出力データやパラメータなど）とともに処理される．また，1 つのリソース（実行計算機など）では，複数のプログラムとデータの組を処理することができる．これらの関係から，3 要素を関連付けて分散処理スクリプトを記述する場合の構造化モデルを**図 3** に示す．

図 3 は OMT [9] 記法を用いて 3 要素の関係を示している．P はプログラムオブジェクトを，D はデータオブジェクトを，R はリソースオブジェクトを示したものである．PD 組とは，1 つのプログラムオブジェクトと 0 個以上のデータオブジェクトから構成される．RPD 組とは，PD 組の処理に必要なリソースの情報を加えるために，1 つのリソースオブジェクトと 1 つ以上の PD 組から構成される．最終的にスクリプトは，1 つ以上の RPD 組から定義される．

4. 分散処理スクリプトの文法

分散処理スクリプトの文法は，UNIX に標準的に搭載されている Bourne Shell [10] の文法を拡張することにより実装する．このような実装方法を選択したのは，Bourne Shell で記述された既存のプログラム資産の再利用性を確保するとともに，分散処理スクリプトの文法学習にかかるユーザの負担を軽くし，また，Shell を経由することによって，UNIX 上の他プログラムと容易に連携可能にするためである．分散処理スクリプトの文法は，Bourne Shell の文法に対して，計算資源をオブジェクト化するための記述方法と，オブジェクトを構造化するための記述方法を拡張することによって実現する．本章では，これらの記述文法について述べる．

4.1 計算資源のオブジェクト化記述

スクリプト構造化モデルでは，仮想的な計算資源を

```
1:  class_object    : 'class' class_name class_id '{' class_body '}'
2:  class_id        : 'Rclass' | 'Pclass' | 'Dclass'
3:  class_body      : class_slot class_method
4:  instance_object : program_name | data_name | hostname ...
```

図 4 クラスオブジェクトとインスタンスオブジェクトの記述形式
Fig. 4 Description form for class objects and instance objects.

クラスオブジェクトとして，実際の計算資源をインスタンスオブジェクトとして扱う．このうち，インスタンスオブジェクトは，実際の計算資源を直接表しているため，実計算資源の名称をそのまま分散処理スクリプトの記述において用いる．これにより，仮想的な計算資源を用いない場合には，従来の Bourne Shell と同様の記述方法になるため，これまで Bourne Shell で実行されていたスクリプト（シェルプロシージャ）は，変更することなくそのまま本システムで処理可能となる．

クラスオブジェクトは，複数の計算資源を透過的に扱える必要があり，また，計算資源の不確定性を解決するためのメソッドを記述できる必要がある．これらの記述形式を BNF 記法を用いて表したものを**図 4** に示す．

クラスオブジェクト *class_object* は，クラス名 *class_name*，クラス種別 *class_id*，クラス構造の定義 *class_body* から構成される．*class_name* は任意の文字列で構成される文字列であり，分散処理スクリプトの記述において仮想計算資源の名称として利用される．*class_id* は，オブジェクトクラスを示すキーワードが設定される．キーワードは，オブジェクトのクラスがプログラムに属する場合は *Pclass*，データに属する場合は *Dclass*，リソースに属する場合は *Rclass* が指定される．*class_body* は，クラスオブジェクト内に保持する情報とその操作方法であるメソッドから構成される．クラスが内包する計算資源の名称は *class_body* に保持される．これによりクラス名を指定した際に透過的に扱う複数の計算資源が定義される．また，クラスオブジェクトからインスタンスオブジェクトを生成する場合の方法をメソッドとして *class_body* に定義する．これにより，計算資源の不確定性を解消し，実際の計算資源を決定するための処理が表現される．

4.2 オブジェクトの構造化記述

分散処理スクリプトは，オブジェクト化された複数の計算資源を関連付けることによって表現される．オブジェクト間の関連は，**図 5** に示すオブジェクト構造化の記述形式を用いて表現する．

```
1:  スクリプト : RPD組群
2:  RPD組群    : RPD組群 RPD組
3:  RPD組      : '(' PD組群 ')' '@' R ';'
4:  PD組群     : PD組群 PD組
5:  PD組       : P D群 ';' | P ';'
6:  D群        : D群 D
7:  P          : Pclass | Pinstance
8:  D          : Dclass | Dinstance
9:  R          : Rclass | Rinstance
```

図 5 オブジェクトの構造化記述形式
Fig. 5 Description form for object structure.

```
1:  class multi_computer Rclass {
2:          hostname="Cc1, Cc2, ... Cc10"
3:  }
4:  class  graphic_computer  Rclass {
5:          hostname="Cg1, Cg2, Cg3"
6:          best(){
7:              instance=`select_cpu $hostname`
8:          }
9:  }
10: ( Pc1 )@multi_computer | ( Pv1 )@graphic_computer.best
```

図 6 スクリプトの記述例
Fig. 6 An example of script.

個々の計算資源は，オブジェクト化された時点で透過性が保証されるため，分散処理スクリプトの記述において，仮想的な計算資源か実際の計算資源かを区別することなく利用することができる（図 5 の 7〜9 行目）．実行プログラムとデータの組を表す PD 組に関しては，Bourne Shell と同様の文法規則であり拡張する必要がない（図 5 の 4〜6 行目）．本スクリプトでは，スクリプトを構成する要素として，プログラムとデータに加え，リソースを導入しており，これを表現するために，R と PD 組との関係を記述する必要がある．この関係を記述するために，@演算子を導入し，これを用いて PD 組と R を関連付ける（図 5 の 3 行目）．@演算子で関連付けられたリソースオブジェクト R は，() とともに用いられた場合実行計算資源を表し，() を用いずにプログラムオブジェクト P，またはデータオブジェクト D と関連付けられた場合は，それらのオブジェクトが格納されているリソースオブジェクトを表す．スクリプトは，複数の RPD 組から構成される（図 5 の 1，2 行目）．

4.3 スクリプト記述例

図 6 にスクリプトの記述例を示す．例では，まずはじめに，仮想計算機 *multi_computer* を，10 台の実計算機 Cc_1 から Cc_{10} で構成し（図 6 の 1〜3 行目），仮想計算機 *graphic_computer* を，3 台の実計

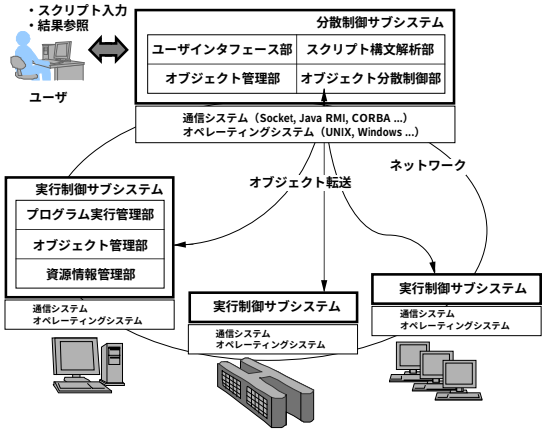


図 7 システム構成
Fig. 7 System architecture.

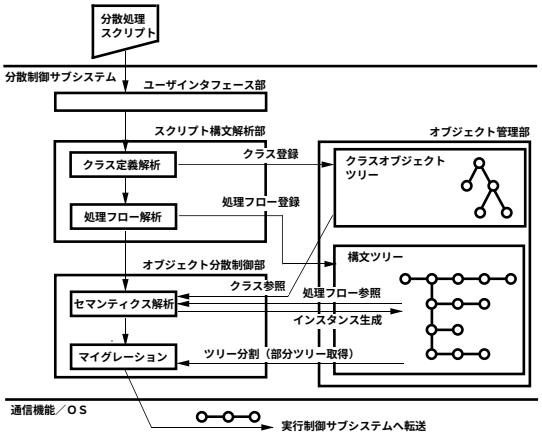


図 8 分散制御サブシステム
Fig. 8 The distribution control sub-system.

算機 Cg_1 から Cg_3 で構成する（図 6 の 4〜9 行目）．次に計算処理本体として，プログラム Pc_1 を，仮想計算機 *multi_computer* 上で実行し，その結果を仮想計算機 *graphic_computer* 上のプログラム Pv_1 で処理し結果を表示する（図 6 の 10 行目）．この計算処理では，結果を表示する実計算機は 1 台でよいため，仮想計算機 *graphic_computer* の best メソッドを実行し，実計算機を 1 台選択している．

5. システムアーキテクチャ

分散処理フロー制御システムは，分散制御サブシステムと実行制御サブシステムから構成される．これらサブシステムを，OS および通信機能（ソケット，Java RMI など）上に実装することで実現する．(**図 7**)．

本章では，各サブシステムの動作と制御方法について述べる．

5.1 分散制御サブシステム

分散制御サブシステムは，ユーザインタフェース部，スクリプト構文解析部，オブジェクト分散制御部，オブジェクト管理部から構成される（図 7）．

ユーザインタフェース部は，ユーザとのインタラクションを制御する機能である．ユーザから分散処理スクリプトの文字列を入力し，処理結果と分散処理状態の表示などを行う．スクリプト構文解析部は，ユーザインタフェース部を通して入力された分散処理スクリプトの構文を解析し，スクリプトを構成する各要素の情報および要素間の関係情報をオブジェクト管理部へ転送する．オブジェクト管理部は，スクリプト構文解析部から転送された解析情報を基にクラスオブジェクトやインスタンスオブジェクトの生成と管理，およびこれらオブジェクトから構成されるクラスオブジェクトツリーと分散処理フローの構文ツリーを管理する．クラスオブジェクトツリーは，クラス間の継承関係をツリー状に表現したものである．また，構文ツリーは，スクリプトの解析結果をツリー状に表現したものである．オブジェクト分散制御部は，分散処理の準備として，オブジェクト管理部が管理しているクラスオブジェクトツリーと構文ツリーの内容を解釈し，個々の計算機へ分散する部分ツリーの切り出し処理を行う．部分ツリーは，*RPD* 組の単位で認識され切り出される．また，切り出した部分ツリーを，各計算機上の実行制御サブシステムへ転送する．

分散制御サブシステムの各部は，さらに複数の機能によって構成される（**図 8**）．以下，各機能の制御方法について述べる．スクリプトの内容は，仮想計算資源の定義であるクラス定義と，仮想計算資源と実計算資源を要素として記述される分散処理フロー定義の 2 種類の情報から構成される．これらの情報に対して，まずはじめに，スクリプト構文解析部のクラス定義解析機能がクラス定義を解析し，仮想計算資源の構成情報をオブジェクト管理部へ渡す．オブジェクト管理部では，仮想計算資源の構成情報をクラスオブジェクト化し，クラスオブジェクトツリーを構成する．次にスクリプト構文解析部の処理フロー解析機能が分散処理フロー定義を解析し，リソースオブジェクト，プログラムオブジェクト，データオブジェクトに分解し，これら要素の定義内容と，各要素間の関連情報をオブジェクト管理部へ報告する．オブジェクト管理部では，各オブジェクトの情報とオブジェクト間の関連情報を基に分散処理フローの構文ツリーを構成する．

次にオブジェクト分散制御部のセマンティクス解析機能によって，構文ツリーのセマンティクス評価を行

い，インスタンスオブジェクトを生成する．分散処理スクリプトは，実計算資源を示すインスタンスオブジェクトだけでなく，仮想的な計算資源を示すクラスオブジェクトもスクリプト内に含んでいる．構文ツリーのセマンティクス評価では，クラスオブジェクトに対してインスタンス生成指示を出すことで，実際に処理で利用される実計算資源を表すインスタンスオブジェクトを生成する．この処理により，分散処理で実際に利用されるすべての実計算資源が生成されるため，分散可能な状態になる．スクリプト構造化モデルに従った場合，実行の最小単位は，リソースオブジェクト，プログラムオブジェクト，データオブジェクトで構成される RPD 組である．よって，オブジェクト分散制御部のマイグレーション機能では，分散処理をするために構文ツリーをトレースし，RPD 組を認識し，部分構文ツリーとして切り出す．最後に切り出された部分構文ツリーが，各計算機上の実行制御サブシステムへ転送される．

5.2 実行制御サブシステム

実行制御サブシステムは，オブジェクト管理部，プログラム実行管理部，資源情報管理部から構成される（図 7）．

オブジェクト管理部は，分散制御サブシステムから転送されてきた部分構文ツリーの情報に従い，オブジェクトツリーを再構成し，管理する機能である．プログラム実行管理部は，オブジェクト管理部が管理するオブジェクトツリーの情報に従い，実際の計算資源へアクセスしプログラムを実行する機能である．資源情報管理部は，リソースオブジェクトに対応する計算資源（計算機など）の状態を管理する機能である．

実行制御サブシステムの各部は，さらに複数の機能によって構成される（**図 9**）．以下，各機能の制御方法について述べる．まずはじめに，分散制御サブシステムから転送されてきた部分構文ツリーの情報がオブジェクト管理部へ渡され，オブジェクトツリーが再構成される．次に，プログラム実行管理部がオブジェクトツリーの情報を解析し，起動制御を行う．起動制御は，データ駆動型実行制御[11]の原理をオブジェクトに対して適用し，オブジェクトの到着によってプログラムの起動を行う[12]．すなわち，部分構文ツリーを構成するオブジェクトがすべて揃ったものから起動手続きが開始される．起動手続きでは，各オブジェクトに対応する実計算資源が存在するかどうかを資源情報管理部へ問い合わせて確認し，部分ツリーを構成するすべてのオブジェクトに対応する実計算資源が存在する場合にはプログラムを起動し，存在しない場合には起動を保留するように制御される．プログラムが起動された後は，プログラムの実行状態と実行結果が分散制御サブシステムのユーザインタフェース部へ送信され，最終的にユーザに提供される．

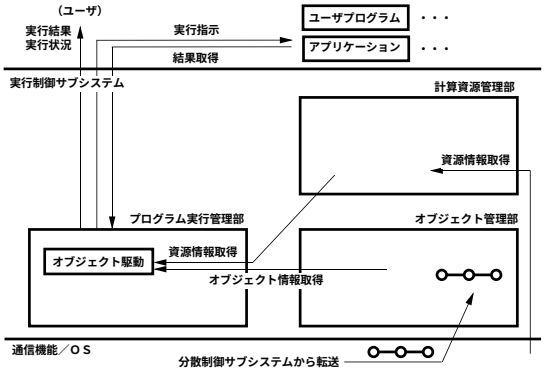


図 9 実行制御サブシステム
Fig. 9 The run control sub-system.

6. 評　　価

本章では，分散処理スクリプトの記述性を評価するために，仮想的な計算資源を用いた場合と用いなかった場合の記述量を測定し，その結果を考察する．また，分散環境の状態変化に対する適応性を評価するために，計算資源の状態を変化させた場合の分散処理フローの状態を観測し，動的なインスタンスの生成とその構造化によって，状態変化に適応していることを確認する．

6.1 評 価 方 法

スクリプトの記述性評価では，分散処理スクリプトを記述する際に，仮想的な計算資源を用いた場合と用いなかった場合について，スクリプトの総ステップ数と，仮想計算資源の定義に要するステップ数，計算処理のステップ数，および，計算処理の中で実計算資源に依存するステップ数を測定する．次に分散環境の状態や構成に変化を与えるために，各計算機の負荷状態や稼動状態を変化させ，インスタンスの生成とその関連付けの状態を確認する．

評価に用いる計算問題として，分子動力学計算などで一般的に行われているパラメータサーベイと呼ばれる計算形態を用いる．この計算形態では，ある計算プログラムに対して，いくつかの計算パラメータのセットを用意し，個々のセットに対して計算処理を実行していく．本評価では，次の実行手順の分散処理スクリプトを記述し，記述性を評価する．

(1) 任意の計算機上に分子動力学に基づく計算プログラム Ps_1 を用意する．また，計算プログラム Ps_1 の処理に用いる計算パラメータのセットを

10 セット（Dp_1 から Dp_{10} まで）用意する．

(2) 各計算機上に計算プログラム Ps_1 を転送しコンパイルする．

(3) 各計算機のうち，計算処理が実行可能な計算機を選択し，計算パラメータを 1 セット転送し，計算処理を開始する．実行可能な計算機の選択は，計算機の負荷に一定の閾値を設け，この閾値以下の計算機を実行可能な計算機と判断する．

(4) 各計算機上の計算結果の集計処理をするために，処理の実行時点で計算機負荷があらかじめ定めた閾値より小さく，かつ計算性能の高い計算機を選択し，集計処理を行う．

(5) 可視化処理用プログラムを実行可能な計算機において，計算結果データの可視化処理を行う．

(6) 次に Ps_1 と計算内容の異なる計算プログラム Ps_2 を用意し，Ps_1 と同じ計算資源を用いて (1)〜(5) の手順で処理する．

以上のステップで処理を行うスクリプトを作成する．作成するスクリプトは，(1)〜(5) を実計算資源を用いて処理するスクリプト Sr_1，(1)〜(5) を仮想計算資源を用いて処理するスクリプト Sv_1，(6) を実計算資源を用いて処理するスクリプト Sr_2，(6) を仮想計算資源を用いて処理するスクリプト Sv_2 の 4 種類を作成する．定義する仮想計算資源は，全計算機を表す仮想計算機 Vc_1，可視化処理可能な計算機を表す仮想計算機 Vc_2，複数の計算機上の Fortran コンパイラを表す Vfc の 3 種類を用いる．

評価で利用する計算機環境は次のとおりである．これら計算機のうち可視化処理可能な計算機は (4)，(5)，Fortran コンパイラを持つ計算機は (2)，(3)，(4) である．

(1) Sun SparcStation 10，Memory: 64 MB，OS: Solaris 2.6

(2) Sun SparcStation 20，Memory: 512 MB，OS: Solaris 2.6

(3) IBM 互換 PC（Pentium 200 MHz），Memory: 128 MB，OS: BSD/OS 4.0.1

(4) SGI INDY（MIPS R5000 150 MHz），Memory: 256 MB，OS: IRIX 6.2

(5) IBM 互換 PC（PentiumIII 500 MHz），Memory: 96 MB，OS: Windows 98

これらを 10Base-T で接続した LAN 環境を用いる．

6.2 測定結果と評価

表 1 にスクリプト記述量の測定結果を示す．また，実行例を**図 10** に示す．図 10 では右下のウィンドウで計算結果データの可視化結果を表示し，背後のウィンドウが各計算機の実行状態を表示している．

表 1 測定結果
Table 1 Results of measurement.

	Nts	Nvc	Nps	Ndr
Sr_1	133	0	133	50
Sv_1	99	18	81	0
Sr_2	108	0	108	36
Sv_2	91	18	73	0

Sr_i: 仮想計算資源未使用スクリプト
Sv_i: 仮想計算資源使用スクリプト
Nts: 総ステップ数
Nvc: 仮想資源定義のステップ数
Nps: 計算処理のステップ数
Ndr: 実資源依存ステップ数

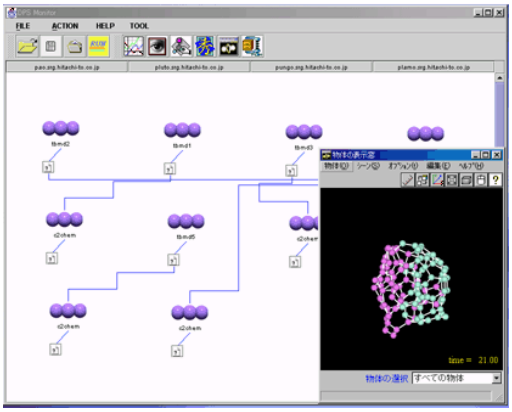


図 10 実行例
Fig. 10 An example of execution.

6.2.1 記述性の評価

分散処理スクリプトの記述において，従来のように複数の実計算資源を用いて記述した場合，各計算機に処理を分散させるための割付け処理を記述しなければならないため処理スクリプトが複雑化していた．

これに対し仮想的な計算機を用いた場合，複数の計算機へのプログラムの割付けを 1 台の仮想計算機への割付けとして表現できるため，処理スクリプトが単純化され記述量が減少する．これは仮想計算資源を用いなかった Sr_i の Nts（総ステップ数）より仮想計算資源を用いた Sv_i の Nts が減少していることから確認することができる．また，計算機をプログラムに割り当てる際に，計算機の物理的な特性や構成を意識するのではなく，計算処理の特性に適した計算資源を用いて分散処理フローを表現することができるため，ユーザに要求される計算資源の物理的な知識の量が少なくて済むことを確認した．

6.2.2 動的な状態変化への対応の評価

従来のように実計算資源を分散スクリプトの記述に用いた場合，処理の実行時まで計算資源が決定されな

い特性（不確定性）を意識した記述が必要であるため記述難易度が高くなっていた．また，分散処理スクリプトが実計算資源と静的に関連付けられるため，実計算資源の状態（負荷や稼動数）が変化した場合，その変化を受けて処理スクリプトを再構成することが困難であった．

これに対して，スクリプトの記述実験の結果より，実計算資源に依存したスクリプト量（Ndr）が 0 になっていることから，不確定性を仮想計算資源内に限定できていることが確認され，ユーザが分散処理フローの記述において不確定性を意識しなくてよいことが確認された．したがって，スクリプトの記述難易度を低くすることができたと考えることができる．

また，記述された処理スクリプトが分散環境の状態変化に適応しながら実行可能である点を確認するために，各計算機の負荷や稼動状態をランダムに変化させた環境において分散処理スクリプトを実行した．その結果，必要な数のインスタンス（実計算機や実プログラムなど）が生成され，これらインスタンスが動的に関連付けられることによって，分散環境の状態に適応していることを確認した．

7. 関 連 研 究

分散処理を行う際に，複数の計算機を選択的に利用し，処理のパフォーマンスを向上させる研究はこれまでいくつか行われてきている．しかしながら，分散処理の内容に適した仮想的な計算資源を構成し，これらの組合せで分散処理を表現することによって，ユーザの記述量や記述難易度を低減したうえで，仮想計算資源の組合せ情報から，実行に利用する計算資源の数やその関係を決定することによって，分散環境の動的な変化に，より柔軟に対応しようとする研究はあまり例を見ない．

Condor[6]は，分散する複数の計算機群の中から空いている計算機を選択しプログラムを実行したり，プログラムを実行中の計算機が何らかの理由により利用できなくなった場合に，他の計算機へプログラムを移動させ再開させたりする仕組みの提供を目的としている．この仕組みを用いることで，ユーザは容易に実行に適した計算機を選択し利用することができる．また，分散環境の動的な状態変化に対応しながら実行のスループットを向上させることができる．しかしながら，複数の計算資源を論理的に 1 つの計算資源として扱えるわけではなく，また，計算機の選択時には，計算資源の物理的な構成や特性を指定しなければならないため，ユーザはこれらに関する知識を十分に持っている必要がある．よって分散プログラムを作成する際の記述量や記述難易度は低減されない．

Legion[7]は，計算資源の構造モデルであるオブジェクトモデルにおいて，CPU やメモリ，ハードディスクのストレージなど，計算機を構成する要素の単位をオブジェクトの属性として扱えるようにしており，これらを組み合わせて利用することで，CPU 性能やメモリ量などの計算資源の物理特性に関して，より細かくプログラムの要求に応えることができる．また，利用するオブジェクトの選択においては，他のリソース管理システムやスケジューラ，ユーザが記述したスケジューリング方法などを複数組み合わせて利用することができるため，多様なプログラムに適応する計算資源を得やすいと考えられる．

しかしながら，クラスオブジェクト間の組合せ情報を用いてインスタンスの組合せを導出しないため，分散環境の状態変化を分散処理フローの構造（インスタンスの組合せ構造）に反映することが困難である．たとえば，利用可能な計算機の数に合わせて分散するプログラム数を調整し，これらを関連付けて分散処理フローを構成し実行するといった実行制御が困難である．

以上のように，他に比べ本論文の方式では，記述量や記述難易度の低減，分散処理フローの再構成による分散環境の状態変化への動的な適応などが実現されている．

8. お わ り に

本論文では，ユーザが複数の計算機を用いて分散処理を行う場合に，ユーザの作業の内容や特性に合わせて仮想的な計算資源を定義し，これを用いて分散処理スクリプトを記述し実行するシステムについて述べた．

仮想的な計算資源を用いることで，作業に適した計算資源が提供され分散処理フローが単純化されることから，記述量を減少させることができた．また，ユーザに要求される計算資源に関する知識が少なくなるとともに，計算資源の不確定性が隠蔽されるため，記述難易度を低くすることができた．これらよりユーザのスクリプト記述効率を向上させることが可能である点を確認することができた．また，動的なインスタンスの生成と分散処理フローの構成によって，分散環境の状態変化への動的な適応などが実現された．

仮想的な計算資源を用いることで，さらに次の効果が期待できる．

(1) 計算資源利用のノウハウを持つユーザが仮想計算資源を定義し，これを他のユーザが利用することで，計算資源利用のノウハウを共有するこ

とができる．

(2) 既存の計算資源を組み合わせて，より高度な機能を持つ計算資源を定義し利用することができる．

今回試作したシステムでは，分散処理スクリプト実行時のプログラム間通信に，ソケットまたはJava RMIを選択し利用することができる．また現在，CORBAやGlobus Toolkitも利用できるよう機能の拡張を行っている．これらの通信機能は，他の計算資源と同様，1つの通信機能として抽象表現されているため，分散処理プログラムを複雑にすることなく，プログラムの実行時に，ネットワークやユーザ要求に適した通信機能を動的に選択し利用することが可能である．これは，インターネットのように複数のネットワークが相互接続された広域ネットワークにおいて，個々のネットワークごとにセキュリティポリシーや利用可能な通信機能が違う場合に有効に機能すると考えられる．今後はこの有効性について評価する予定である．

参 考 文 献

1) Coulouris, G., Dollimore, J. and Kindberg, T.: *Distributed Systems*, Addison-Wesley Publishing Company, Inc. (1994).
2) International Standards Organization (ISO): Basic Reference Model of Open Distributed Processing. ISO/IEC JTC1/SC212/WG7 CD 10746-1 (1992).
3) Object Management Group (OMG): CORBA 2.2/IIOP Specification. http://www.omg.org/library/c2indx.html.
4) Sun Microsystems, Inc.: Java (TM) Remote Method Invocation. http://java.sun.com/products/jdk/1.2/docs/guide/rmi/index.html.
5) Foster, I. and Kesselman, C.: *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Inc. (1999).
6) Epema, D., Livny, M., Dantzig, R., Evers, X. and Pruyne, J.: Worldwide Flock of Condors: Load Sharing among Workstation Clusters, *Journal on Future Generations of Computer Systems* (1996).
7) Chapin, S., Karpovich, J. and Grimshaw, A.: The Legion Resource Management System, *Proc. 5th Workshop on Job Scheduling Strategies for Parallel Processing* (1999).
8) Casanova, H., Legrand, A., Zagorodnov, D. and Berman, F.: Heuristics for Scheduling Parameter Sweep applications in Grid environments, *Proc. 9th Heterogeneous Computing workshop*, pp.349–363 (2000).
9) Rumbaugh, J.: *Object-Oriented Modeling and Design*, Prentice Hall, Inc. (1991).
10) Gilly, D.：UNIX クイックリファレンス，オライリー・ジャパン (1998).
11) Sohn, A. and Gaudiot, J.: A macro actor/token implementation of production systems on a data-flow multiprocessor, *Proc. 12th Int. Joint Conf. Artificial Intelligence*, AI, pp.36–41 (1990).
12) Kikuchi, K., Shiratori, N. and Miyazaki, M.: The process control environment for the dynamic structured distributed system, *Proc. 5th International Conf. ISAS'99*, International Institute of Informatics and Systemics (IIIS) (1999).

(平成12年10月31日受付)
(平成13年 4 月 6 日採録)

菊池 一彦（正会員）

1985年東北工業大学工学部電子工学科卒業．同年日立東北ソフトウェア（株）入社．1997年東北大学大学院情報科学研究科博士前期課程修了．言語プロセッサ，知識処理システム，コミュニケーション支援システム等の研究開発に従事．電子情報通信学会，ソフトウェア科学会各会員．

菅沼 拓夫（正会員）

1997年千葉工業大学大学院博士後期課程情報工学専攻修了．現在，東北大学電気通信研究所助手．博士（工学）．やわらかいネットワーク，エージェントプログラミングに興味を持つ．8th JWCCベストプレゼンテーション賞受賞．情報処理学会第54回全国大会奨励賞受賞．電子情報通信学会会員．

白鳥 則郎（正会員）

1977年東北大学大学院博士課程修了．1984年同大学助教授（電気通信研究所）．1990年同大学教授（工学部情報工学科）．1993年同大学教授（電気通信研究所）．情報通信システム，ソフトウェア開発環境，ヒューマンインタフェースの研究に従事．1993年本会マルチメディア通信と分散処理研究会主査．1985年本会25周年記念論文賞受賞．情報処理学会理事（1996〜1998），情報処理学会フェロー．IEEE Fellow．

宮崎 正俊（正会員）

1962年東北大学工学部電気学科卒業．大型計算機センター講師，助教授，教養部情報科学科教授，大学院情報科学研究科教授を経て，現在，岩手県立大学ソフトウェア情報学部教授および同学部長．1972年より1年間MIT客員研究員．オペレーティングシステム，情報システム構築法，並列分散処理，システム評価，データベースに関する研究に従事．ACM，日本ME学会，日本教育工学会，電子情報通信学会，日本ロジスティクスシステム学会各会員．東北大学名誉教授，工学博士．