福山大学工学部紀要
第26巻2002年9月

PCクラスタにおける並列計算の動的負荷分散

石川 洋* 尾関 孝史* 渡邊 栄治**

Dynamic Load Balancing for Parallel Computing in PC Cluster

Hiroshi ISHIKAWA*, Takashi OZEKI*, and Eiji Watanabe**

ABSTRACT

A PC cluster is one of the parallel computers and it is constructed with a number of commodity PCs and free software. Therefore we can obtain such a system with ease and reasonable cost. Dynamic load balancing is one of the methods to execute a program for parallel computing in PC clusters effectively. We introduce a general method for dynamic load balancing and propose a new one.
This paper is constructed with two parts. In the first part, we introduce some concepts of PC Clusters, Parallel Computing, Dynamic Load Balancing, and Computational Reflection. In the second part, we propose a method of dynamic load balancing based on the framework of computational reflection.

キーワード：PCクラスタ，並列計算，動的負荷分散，自己反映計算

Keywords: PC Cluster, Parallel Computing, Dynamic Load Balancing, Reflection

1. はじめに

PCクラスタとは一般的に入手可能なパーソナルコンピュータを複数台ネットワークで接続して構築するメモリ分散型の並列計算機である．近年のCPUやメモリの飛躍的なコストパフォーマンス向上や，異機種混合なマシンの利用が可能であることから，比較的廉価で数十台規模のシステムが容易に構築できる．

PCクラスタ上で並列計算を実現しようとするとき，C, C++, Fortranなどのコンパイラの他に，各PCのCPUの支配下にあるメモリ内のデータをアクセスするための並列ライブラリが必要となる．代表的な並列ライブラリにはPVM(Parallel Virtual Machine), MPI(Message Passing Interface)などがある．

PCクラスタを効率良く利用して並列計算を実行するためには，すべてのコンピュータに対し，その処理能力に見合った作業量を分担させる仕組みが必要である．この仕組みを負荷分散という．プログラムの実行前にあらかじめ仕事量がわかっていないときには特にこの仕組みは有用である．負荷分散には，プログラムの実行前に，仕事をプロセッサに割り付けておく静的なものと，プログラム実行中にプロセッサに仕事を割り付ける動的なものがある．本稿では，一般的な動的負荷分散のしくみを紹介したのち，自己反映計算の概念を利用した動的負荷分散の実現について提案する．

本稿の構成は以下の通りである．第2節において，PCクラスタ，並列計算ライブラリ，動的負荷分散，自己反映計算などの諸概念について述べる．第3節では，自己反映計算の概念を利用した動的負荷分散の実現について提案する．最後に本提案に関する議論と今後の課題について述べる．

2. 諸概念の概要

本節では，PCクラスタ，並列計算ライブラリ，動的負荷分散，自己反映計算の諸概念について簡単に説明しておく．

* 情報処理工学科 ** 甲南大学

2.1 PCクラスタ

PCクラスタ [3,4,6] とは，一般的に使用されているパーソナルコンピュータ (Personal Computer, PC) を複数台ネットワークで接続して構築するメモリ分散型の並列計算機である．近年のCPUやメモリの飛躍的なコストパフォーマンス向上や，異機種混合なマシンの利用が可能であることから，比較的廉価で数十台規模のシステムが容易に構築できる．このようなことはハードウェアだけでなく，ソフトウェアに関しても同様である．PCクラスタを構成する各コンピュータにはFree BSDやLinuxといった，UNIX系のフリーなオペレーティングシステムをインストールすることが多い．

クラスタを構成するコンピュータはネットワーク接続されており，ネットワークトポロジーはリング型，スター型などさまざまであるが，並列計算を行う場合は，仕事を割り振り，計算結果を受け取るコンピュータ (Master)1台と，Masterから割り振られた仕事を実行するコンピュータ (Slave) 数台という構成が最も基本的である (図 1)．

2.2 並列計算ライブラリ

クラスタのような分散メモリシステムにおいては，並列化を実現する方法は大きく2つに分類される．一つは，メッセージパッシング方式でありもう一つはデータ並列方式である．現在よく利用されている並列ライブラリであるPVM(Parallel Virtual Machine) [6] やMPI(Message Passing Interface) [3,6] は前者の方式を採用している．PVMはさまざまな並列計算機ベンダーが独自にチューニングを施した独自のPVMを開発しており，MPIに比べて移植性に乏しいという欠点がある．MPIはPVMより後で開発されたため，PVMが持っている問題点をすべてではないがクリアしている．また，MPIは第三者的機関が仕様を決定しており，汎用性，移植性がPVMより優れている．したがって，本稿では並列計算ライブラリとしてMPIの使用を前提とする．

以降はメッセージパッシング方式およびMPIについて説明する．

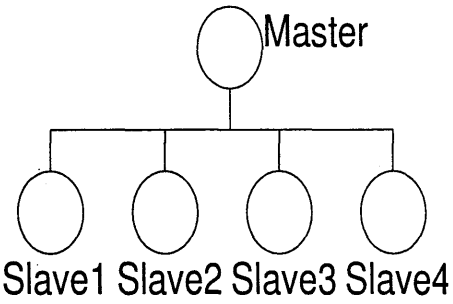

図1 PCクラスタモデル (マスタ・スレーブ方式)
Fig.1 PC Cluster Model(Master-Slave Method)

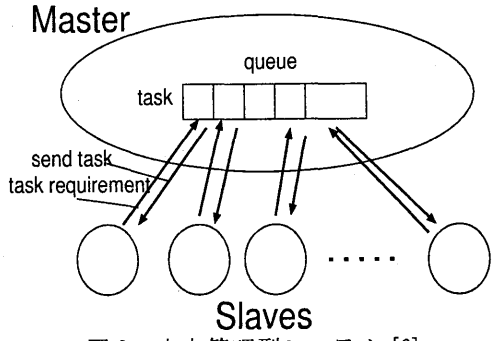

図2 中央管理型システム [6]
Fig.2 Centralized Control System

メッセージパッシング方式は，プロセッサ間でメッセージをやりとりしながら並列処理を実現するものである．この方式の場合，複数のプロセッサ間での通信をお互いのデータの送受信によって実現している．したがって，プロセッサ A 上でプロセッサ B にデータを送信することと，プロセッサ B 上でプロセッサ A からのデータを受信することが対で実現されなければならない．

MPIにおいては，データ交換を行うための通信関数として，任意の2つのプロセス間 (1対1) 通信と，任意のグループに属するプロセス間 (グループ) 通信が提供されている．またそれぞれにおいて，操作が完了するまで手続きから戻らないブロッキング通信と，操作が完了する前に手続きから戻ることがあるノンブロッキング通信がある．これらにより，処理時間の軽減化や同期，非同期通信が可能になる．

2.3 動的負荷分散

PCクラスタを利用した並列計算を効率良く利用する方法の一つに，各コンピュータに作業が偏らないよう，負荷を分散するものがある．負荷を分散方法には，プログラム実行前に各クラスタの作業割当を決定しておく静的負荷分散と，プログラム実行中に負荷のない (または少ない) クラスタに作業を割り当てる動的負荷分散がある．

静的負荷分散は，通常マッピング問題もしくはスケジューリング問題と呼ばれており [6]，プログラムの実行時間やモジュールの依存関係の見積りを行う．この方法はあくまでもプログラムの実行時間や，各コンピュータへの負荷を予想するものであり，効果的な負荷分散を実現するのは困難である．

これに対し，動的負荷分散はプログラムモジュールの実行に応じて負荷を配分していくため，実行中に通信によるオーバーヘッドが生じるが計算資源を有効かつ適切に利用するという観点においては効果的である．

動的負荷分散はプログラムの実行中にプロセッサにタスク (作業) を割り付けるが，その方法は，中央管理型シ

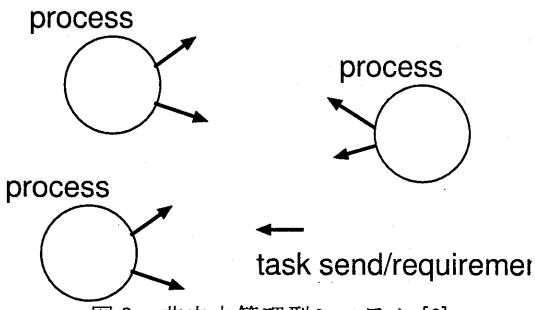

図 3　非中央管理型システム [6]
Fig.3 Non Centralized Control System

ステム (図 2) と非中央管理型システム (図 3) のいずれかに分類できる.

中央管理型システムにおいては，マスター (Master) にタスクの管理場所があり，ここから取り出したタスクをスレーブ (Slave) 側に割り付ける．スレーブは一つのタスクが終了するとマスタに他のタスクを要求する.

非中央管理型システムでは，タスクは任意のプロセス間で受け渡される．それぞれ自分のタスクを持ち，タスクの生成も行うようなプロセスに対し，最初に仕事を割り当てたら，プロセス同士が互いに作用しあってタスク群を実行することも可能である.

2.4　自己反映計算

計算システムが，自分自身の表現を計算システム内部に持ち，それを操作することによって自分自身の構造や振舞いを変更することを自己反映計算 [5] と呼ぶ．自己反映計算を実装する方法としては，メタインタープリタの無限の自己反映階層による方法がある．これは，通常計算を記述するプログラムの解釈，実行をメタレベルのインタープリタで行い，メタレベルのインタープリタはメタメタレベルのインタープリタで行われるという階層を無限に構成して行くというものである．このような構造を持つ計算システムを，メタレベルアーキテクチャを持つ計算システムと呼ぶ.

本来の目的である通常計算をベースレベルの計算とみなせば，計算システムの負荷分散計算は，計算の主体であるシステムを変更するという意味でメタレベルの計算である．また，計算システム自身で，負荷の偏った計算システムを，負荷の分散されたものへと動的に変化させるという操作をすることになるので，自己反映計算を行っていると考えることができる.

3.　自己反映計算を利用した動的負荷分散の提案

動的負荷分散の一形態である非中央管理型システムは，

- タスクは任意のプロセス間で受け渡される.
- それぞれ自分のタスクを持ち，タスクの生成も行うようなプロセスに対し，最初に仕事を割り当てたら，プロセス同士が互いに作用しあってタスク群を実行することも可能である.

という特徴を備えている．この場合，プロセス同士のメッセージ送受信や，タスクの負荷調整を各プロセスで実行しなければならない．これらの処理を実現するために，自己反映計算の概念を導入する.

メッセージ送受信やタスクの負荷調整といった計算資源管理はアプリケーション (通常の計算，ベースレベル) の計算過程の制御を行うわけであり，アプリケーションからみればメタレベルの計算を行っていると考えられる．アプリケーションにあわせた計算資源管理機構をモジュールとして与える枠組として，並行・分散計算システムにおける自己反映計算が自然に利用できる [1,7].

計算資源管理のようなメタレベルアーキテクチャを持つ計算システムは，動的負荷分散計算に関して，以下のような利点を持っていると考えられる.

- 負荷分散をメタレベルで行うことにより，これを自然に導入できる.
- 自己反映計算を行うことによって，計算システムの負荷の状況に，計算システムを動的に適合させることができる.
- プログラミング言語の枠内で，通常計算とのモジュール分離が容易に実現できる.
- メタレベルの計算をプログラミング言語で記述することにより，アプリケーションプログラムに依存するような負荷分散方式を柔軟に実装することができる.

以降では，計算資源管理機構のモデル化について述べる.

3.1　計算資源管理機構のモデル化

自己反映計算を並列計算に利用するために一般の計算と計算資源管理機構を実現するためのモデルを提案する．次のことを前提とする.

> 各 PC クラスタにおいて，図 3でいうプロセスが一つ存在し，それが計算資源管理機構を持っており，メタレベルの計算を行う.. 通常の計算は，そのプロセスが子プロセスを生成することでベースレベルの計算を実現する.

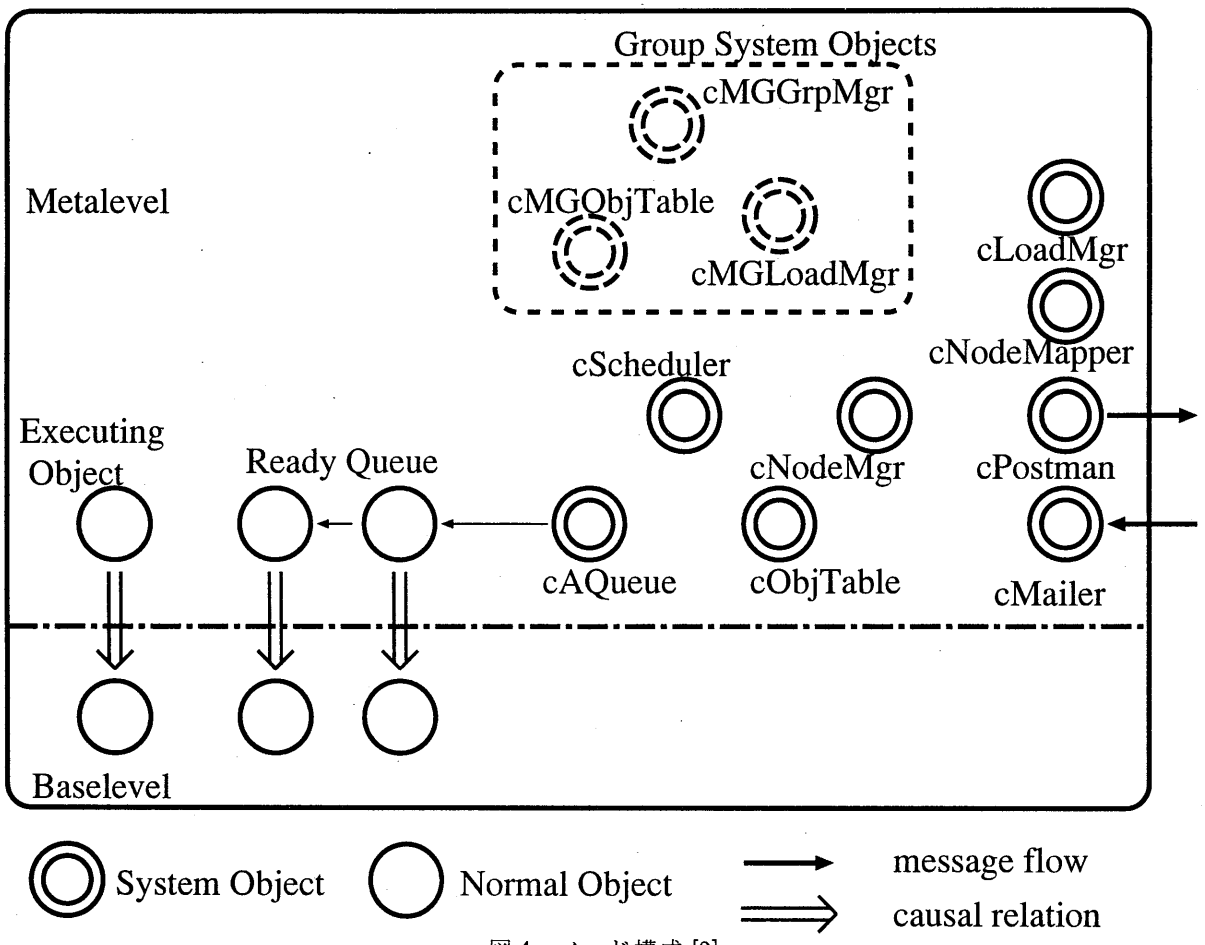

図4　ノード構成 [2]

Fig.4 Structure of Nodes

自己反映計算では，通常の計算はベースレベルで実現される．ベースレベルで行われる計算や計算資源管理をメタレベルで実行する．そのためには，

- 計算主体の管理
- データの送受信
- 計算主体の情報の管理
- 計算環境情報の管理

を行う機構が必要となる．

計算主体の管理では，通常の計算の開始，入出力データの管理，通常の計算の終了の感知を行う．

データの送受信では，通常の計算に必要なデータの送受信と，メタレベル間 (クラスタ間) で必要なデータの送受信が必要となる．メタレベル間で必要なデータとは，計算環境情報であり，

- 自分自身 (PC クラスタ) のアドレス
- 送信可能なアドレス一覧
- メッセージを送信すべき相手
- グループ管理
- メッセージ受信機構
- 自分自身の計算資源情報 (CPU，搭載メモリ，ハードディスク容量など)
- 分担作業の管理
- 自分自身の作業負荷状況

などを指す．計算主体の情報の管理は，負荷情報，割り付けられた作業情報のことを指す．このような機構の概念を図4に示す．ここでノードは図3のプロセスを意味する．ノードはシステムオブジェクトとその他の一般オブジェクトから構成される．システムオブジェクトは，ノードシステムオブジェクトとグループシステムオブジェクトに分類している．ノードシステムオブジェクトはそのノード自身の管理と，他ノードとのメッセージ送受信を行うオブジェクトからなる．また，並列計算を実行するノードの全体または一部分を管理するオブジェクトがグループシステムオブジェクトである．

図4の各オブジェクトとそれらの機能を表1にまとめた．

表1　主要オブジェクトとその機能 [2]
Table1 The significant objects and their function.

オブジェクト名	機能
cNodeMgr	ノードの管理.
cAQueue	実行待ちオブジェクトキューの管理.
cGenObject	オブジェクト生成用のシステムオブジェクト.
cLoadMgr	ノードの負荷マネージャ.
cMailer	他ノードからのメッセージ受信.
cNodeMapper	オブジェクト識別子とノード間通信用のアドレス間の変換.
cObjTable	ノード上のオブジェクトの登録.
cPostman	他ノードへのメッセージ送信.
cSOT	システムオブジェクトの登録.
cScheduler	ノード上のオブジェクトの並行実行.
cMGGrpMgr	グループの管理.
cMGLoadMgr	グループの負荷マネージャ.
cMGMailer	グループのオブジェクトへのメッセージ送信.
cMGObjTable	グループに属するオブジェクトの登録.
cMGPostman	グループのシステムオブジェクトからのメッセージ受信.
cMGScheduler	グループシステムオブジェクト群の並行実行.

3.2　実現方法について

提案したモデルにしたがって，非中央管理型システムにおける動的負荷分散の実現方法について述べる．

システムオブジェクトや他の一般オブジェクトは関数で実現する．通常の計算は，各 PC クラスタで稼働しているノード (プロセス) が子プロセスを生成して実現する．様々な情報を管理するために適切な構造体を用意する．メタレベルおよびベースレベルを操作するプログラムは同一言語で記述するので，それらを構成する関数名が明確に区別できるように注意を払う必要がある．実装においては C 言語と並列計算ライブラリ MPI を利用することを想定している．MPI には並列計算を実現するための豊富なプリミティブ関数が提供されており (表 2)，これらを活用することで提案した機構が実現できる．

例えば，特定のノード間の通信であれば，`MPI_Send()`, `MPI_Recv()` を利用すればよい．またグループを構成するのであれば，`MPI_Comm_size()` を利用し，そのグループ間の通信は`MPI_Barrier()` を用いる．

4.　議論と今後の課題

動的負荷分散に自己反映計算の概念を導入したことにより，

- 計算資源を有効かつ効率良く利用できる
- 異機種，性能 (CPU，メモリ容量，通信速度などの) 格差で構成されるクラスタ上で，それらの差異を考慮した動的負荷分散が可能
- 機種ごとの性能をメタレベルで管理し，クラスタ間での情報交換が実現可能

などの利点が挙げられる．しかしながら，データや計算環境の情報を頻繁に送受信することが予想され，そのため通信による負荷が従来方法よりは増加する懸念があるといった問題点もある．

本稿では，PC クラスタにおける並列計算の動的負荷分散に自己反映計算を導入することを提案し，そのモデルを示した．今後はこのモデルに基づき，非中央管理型システムにおける動的負荷分散を実現し，具体的な例題を用いてその可能性を考察する．

参考文献

[1] Matsuoka, S., Watanabe, T., Ichisugi, Y., and Yonezawa, A. : Object-Oriented Concurrent Reflective Architectures, *Object-Based Councurrent Computing*, LNCS, Vol.612, pp.211–226, 1992.

[2] 宮内，渡部：メタレベルアーキテクチャを用いた動的負荷分散の実現，北陸先端科学技術大学院大学 Technical Report IS-RR-94-0006S, 1994.

表2　主なMPI関数 [4]
Table2 The significant MPI's primitive functions.

サブルーチン	タイプ	内容
MPI_Init()	環境管理	MPIの実行環境の初期化.
MPI_Comm_rank()	連絡機構	コミュニケータ内のランクを取得.
MPI_Comm_size()	連絡機構	コミュニケータ内のプロセス数を取得.
MPI_Finalize()	環境管理	MPIの実行環境の終了.
MPI_Send()	2地点間	ブロッキング送信. 最も標準的な送信関数.
MPI_Recv()	2地点間	ブロッキング受信. 最も標準的な受信関数.
MPI_Sendrecv()	2地点間	ブロッキング送受信.
MPI_Isend()	2地点間	ノンブロッキング送信.
MPI_Irecv()	2地点間	ノンブロッキング受信.
MPI_Iprobe()	2地点間	source, tag, および comm と一致するメッセージの着信検査.
MPI_Probe()	2地点間	souce, tag, および comm と一致するメッセージが着信するまで待機.
MPI_TEST()	2地点間	ノンブロッキング送受信操作の完了検査.
MPI_Wait()	2地点間	特定の非ブロック送受信の完了待ち.
MPI_Waitall()	2地点間	すべての非ブロック送受信の完了待ち.
MPI_Get_count()	2地点間	メッセージの要素数を返す.
MPI_Barrier()	集合通信	コミュニケータ内でバリア同期を取る.
MPI_Bcast()	集合通信	メッセージをブロードキャストする.
MPI_Reduce()	集合通信	コミュニケータ内で通信と同時に指定された演算を実施.
MPI_Type_extent()	派生データタイプ	特定のデータタイプのサイズを取得.
MPI_Type_struct()	派生データタイプ	新しいデータタイプを作成.
MPI_Type_commit()	派生データタイプ	システムに新しいデータタイプを委ねる.
MPI_Type_free()	派生データタイプ	データタイプの削除.

[3] Pacheco, P. S.: Parallel Programming with MPI, Morgan Kaufmann, 1997. (秋葉訳：MPI並列プログラミング，培風館，2001).

[4] 超並列計算研究会：PCクラスタ超入門，2000.

[5] 渡部：リフレクション，コンピュータソフトウェア, Vol.11, No.3, pp.5–14, 1994.

[6] Wilkinson, B. and Allen, M.: Parallel Programming, Prentice-Hall, 1999. (飯塚，緑川訳：並列プログラミング入門，丸善，2000).

[7] Yonezawa, A. and Watanabe, T.: An Introduction to Object-Based Reflective Concurrent Computation, *Proceedings of the ACM SIGPLAN Workshop on Object-Based Councurrent Programming*, SIGPLAN Notices, Vol.24, No.4, pp.50–54, 1989.