システムソフトウェアと
オペレーティング・システム　71－11
マルチメディア通信と分散処理　73－11
（１９９５．１２．１）

分散処理環境でのタスクスケジューリング

－プログラムの特性と実行環境を考慮した分散処理システム－

天野慎太郎　岡本秀輔　曽和将容
電気通信大学情報システム学研究科

本論文では、分散処理環境において、並列プログラムを効率良く実行させるための手法について述べる。このシステムでは、分散処理環境で並列プログラムを実行する時に、プログラムの特性とシステムの静的および動的な情報の両方を考慮して、より効率良く実行するということを行なっている。これを行なうために、実行環境についての静的及び動的な情報を利用できる機構を提供している。この情報を基づいて実行時システムは環境に応じた効率的なプログラムの実行を行なう。それと同時に、並列プログラムのタスクに関する情報（ここではタスクテーブルと呼ぶ）をプログラマが提供し実行時システムが利用できるようにしている。

Task Scheduling on Distributed Computing

- Distributed Computing taking account of Program Characteristics and System Configuration -

Shintaro AMANO, Shusuke OKAMOTO and Masahiro SOWA
Graduate School of Information Systems
The University of Electro-Communications

We propose the method to make the parallel programs run efficiently in the Distributed Computing. When the parallel programs are executed on the distributed computer, this facility can enable the run-time system to utilize the characteristics of the parallel programs as well as the static and dynamic characteristics of the distributed system. For this purpose, our system has the mechanism that provides the static and dynamic information of the system where the program is running, then our system optimize the program performance by modification of the task allocation. Additionally the task characteristics, called Task Table, is defined by the programmer and can be utilized.

1 はじめに

近年技術の発展により、高速なコンピュータや高速なネットワークを比較的に容易に利用できるようになってきた。また、それに伴いワークステーションやパーソナルコンピュータを使っての分散処理も行なわれるようになってきた。分散処理にはさまざまな特徴があるが、この研究ではプログラムの効率的な実行や負荷の分散について注目する。

これまで分散あるいは並列プログラムを実行する場合、あらかじめ静的に決められたスケジューリングに基づいて実行する方式や実行時に動的にスケジューリングを行なう方法が採用されてきた。しかし、従来の方式ではプログラムの持つ特性と実行環境の状況の両方を反映させることは困難である。なぜなら、プログラマが並列プログラムを作成している時点では、必ずしも実行時の環境が分かっているわけではない。また、実行時システムは特別な機構がない限り、並列プログラムの個々のタスクに関する特性を得ることは無理である。

そこで、プログラムと実行環境がタスクに関する情報を共有できるようにインターフェース（ここではタスクテーブルと呼ぶ）を定義し、それを通して実行時システムがタスクの情報を取得する。さらに実行時システムは、システムの静的及び動的な実行環境に関する情報を併せて利用することによって、効率的でかつシステムの変化に柔軟に対応できるはずである。具体的には、プログラマがその並列プログラムの持つ特性をプログラム中に記述できるようにしておき、システム側では実行環境についての情報を収集しておくことで、プログラムの持つ特性と実行環境の特性の両方を反映した分散実行が可能となる。本研究では、このようなことを実現するために必要な機構と、並列プログラムと実行時システムがタスク情報を共有するための汎用的なインターフェースとその利用方法について検討する。

研究を進めるための前提として、ここでは複数のワークステーションがネットワークで接続された仮想的な並列コンピュータ（ワークステーション・クラスタと呼ばれることもある）を想定している。最近ではPVM[1]やMPI[2]等によって、このような環境での分散処理が実現されている。

2章では、本研究が前提としているドメインやユーザスケジューラについて定義する。3章では、並列プログラムと実行時システムとのインタフェースとなるタスクテーブルの詳細について定義する。4章ではドメインマネージャやユーザスケジューラがどのようにユーザのタスクを割り当てて実行するかについて説明する。

2 システムの構成

本研究では、並列プログラムを実行するための環境として、複数のワークステーションがネットワークにより接続されたものを前提としている。この仮想的な並列コンピュータをここではドメインと呼ぶ。このドメイン上では、並列プログラムを分散させて実行することができる。その構成要素にはドメインマネージャ、ユーザスケジューラというものがあり、これらが協調してプログラムの効率的な実行のために働く。この章では、そのドメイン、ドメインマネージャ、ユーザスケジューラについて説明する。

2.1 ドメイン

ドメインとは、ネットワークで接続された複数のコンピュータを、仮想的な1台のコンピュータとして扱うための機構である（図1）。このドメインはノードと呼ばれるコンピュータから構成されていて、それぞれのノードには最低限プロセッサ、ローカルメモリ、ネットワークインター

フェースを持つものとする。

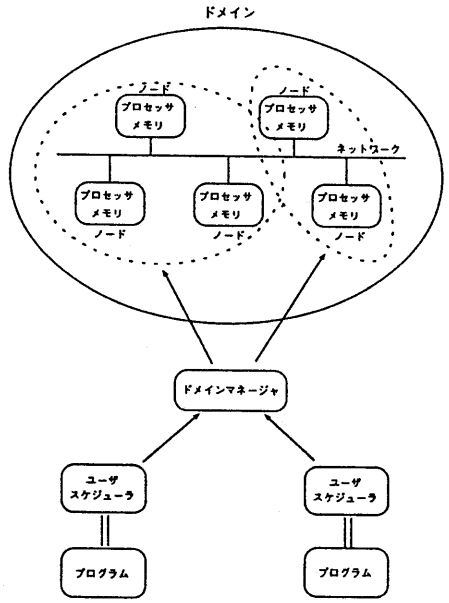


図 1: ドメイン

このドメイン上で並列プログラムを実行する場合、タスクを任意のノード上に割り当てて実行することができる。ドメイン内では、複数のユーザが同時に並列プログラムを実行することができるが、1つのノードには1つのタスクしか割り当てることができないものとする。またタスク間の通信にはメッセージ・パッシングを利用する。

ドメインは人間（システム管理者）の指示によって構成が決定され、ドメインからノードを取り除いたり、ノードを追加したりすることが可能である。この変更によりドメイン内のノード数が変化し、ユーザが利用できるノード数も変化することとなる。その場合、ドメインマネージャとユーザスケジューラとの協調によりタスクの実行を新しい状況に対応させることになるが、詳細については次の章以降で議論する。

2.2 ドメインマネージャ

ドメインマネージャの主な機能は、ドメイン内の静的及び動的な情報を収集しユーザスケジューラに提供することである。さらに、ユーザが実行しているタスクや、ユーザ毎の制限や優先度を管理することも含まれる。

ドメインマネージャが管理する情報には、次のようなものがある。

- ハードウェアに関する静的な情報。プロセッサの性能、メモリ容量、ネットワークの性能、アーキテクチャの種類。
- ハードウェアに関する動的な情報。ドメインを構成するノードの数。各ノードで実行中のタスク。各ノードの負荷情報。
- ユーザ毎の制限。利用できるプロセッサ数、メモリ容量。優先度。

ドメインマネージャはこれらの情報を保持または定期的に収集し、ユーザスケジューラに提供する。

ドメインマネージャのタスク管理の機能には次のようなものがある。

- **タスクの生成** プログラムの起動時やプログラムが新しいタスクを生成する場合にユーザスケジューラからの要求によって行なわれる。
- **タスクの消滅** ユーザのタスクが正常終了した場合。
- **タスクの移動** あるノードからタスクを移動させる必要がある場合に、ユーザスケジューラにその旨通知し、ユーザがスケジューラが新しいノードを決定した場合そこにタスクを移動させる。

- **タスクの停止** タスク移動の必要があるにもかかわらず、他に利用できるノードがない場合は、そのタスクを停止して、後で再開できるように退避用の領域に退避する。

ドメインマネージャのもう1つの機能として、ユーザ毎の制限や優先度の管理がある。この機能は、ユーザ間での不公平をなくしたり、逆にユーザ毎に差を付けるために利用される。これは従来のシステムのプライオリティやリソースに対するリミットをそのまま流用したものである。

2.3 ユーザスケジューラ

ユーザスケジューラは、ユーザが並列プログラムを起動する時に、そのプログラムの複数のタスクをどのノードに割り当てるかということと、実行するタスクの選択を行なう。そして、この決定に基づきドメインマネージャに実際のタスク生成の要求を発行する。この並列プログラムの実行に関する決定には、後述のタスクテーブルとドメインマネージャからの情報を利用する。タスクテーブルの詳細については次の章以降で議論する。

ユーザスケジューラは、基本的にユーザ毎に存在しているプログラムで、概念的にはUNIXのシェルに近いものである。また、ドメインマネージャとは違い、ユーザ毎やプログラム毎に対応するユーザスケジューラを用意することも可能である。これによりタスク割り当てやタスク実行順序のポリシーを自由に変更できる。

2.4 ドメインマネージャとユーザスケジューラの関係

図2はドメインマネージャとユーザスケジューラの関係を表したものである。ドメインマネージャには実行環境についての情報収集を行ない、ユーザスケジューラに提供するという役割がある。一方ユーザスケジューラはこの情報を利用することで、目的の並列プログラムを実行環境に対して最適な状態で実行できるように、タスクの実行順序とノードへの割当を決定する。

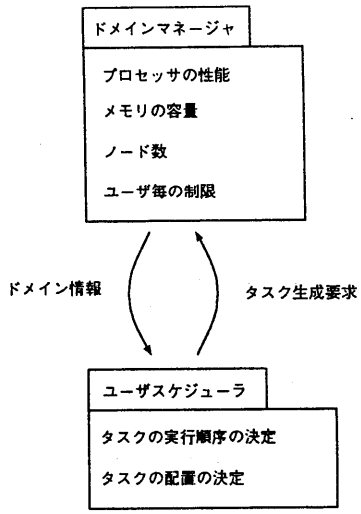


図 2: ドメインマネージャとユーザスケジューラ

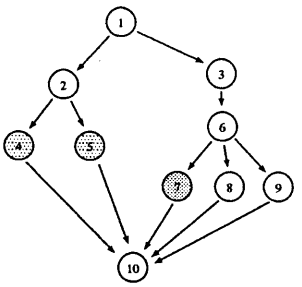
3 タスクテーブル

タスクテーブルとは、並列プログラムの持つ特性を表形式で表したものである。このタスクテーブルはプログラマが用意すべきもので、何らかの方法でユーザスケジューラが読めるような状態になっている。例えば、プログラム内のあらかじめ定められた記憶領域に書き込まれたり、別のデータファイルとしてユーザスケジューラに渡されると行った方法が考えられる。

3.1 タスクテーブルの内容

図3は、典型的な並列プログラムのタスクグラフとそれに対応するタスクテーブルである。タスクテーブルの具体的な内容は、次のようになっている。

タスク番号 並列プログラム中のタスクの識別子。



タスク番号	タスクの順番	計算量	必要メモリ量	その他の情報
1	1	0	0	0
2	2	0	0	0
3	2	0	0	0
4	3	1	0	0
5	3	1	0	0
6	3	0	0	0
7	4	2	1	0
8	4	0	0	0
9	4	0	0	0
10	5	0	0	0

図 3: タスクテーブル

タスクの順番 静的なスケジューリングの結果得られたタスクの実行順序。

計算量 そのタスクで処理される計算量。この値は他のタスクとの比較による相対的な値。

必要メモリ量 そのタスクが必要としているメモリの量。同じく相対値。

その他の情報 現時点では未定。

上記の情報全てが使われるとは限らないし、また情報を省略することも可能であるが、効率の良い実行のためにはプログラマはできるだけ多くの情報を提供することが望ましい。

4 プログラムの実行とタスクの割り当て方法

この章では、ドメイン内で並列プログラムを実行する場合に、具体的にどのような手順でプログラムが起動され、ユーザスケジューラによりタスクの割当が決定され、プログラムが実行状態となるかについて説明する。

ユーザがプログラムを実行する場合はユーザスケジューラを経てドメインマネージャの管理下でタスクが実行される。図4 では、プログラムの起動から実行状態になるまでと、実行中に利用可能なノードの数が変化した場合の動作を表している。

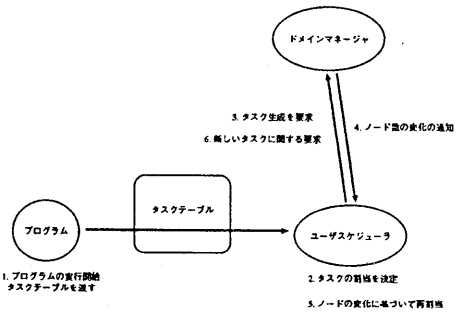


図 4: タスクの割り当て方法

プログラムの起動時に、ユーザスケジューラにはタスクテーブルと呼ばれるものが渡される。これには、実行されるプログラムのタスクに関する情報が含まれていて、ユーザスケジューラはこのタスクテーブルとドメインマネージャからの実行環境に関する情報を基にプロセッサの割り当て等を決定する。ユーザスケジューラは自分の決定をドメインマネージャに伝え、その要求に基づいてドメインマネージャは各ノードにタスクの配置を行ないタスクを実行状態にする。

プログラムが実行中に何らかの原因でノード数が変更することが考えられる。例えば、ドメインの管理者の操作によりノードの数が増加する場合や、優先度の高いユーザがプログラムを実行しようとした場合などである。その場合、ドメインマネージャは対象となるノードへのタスクの再配置

や、そのノードからのタスクの移動などを行なう必要がある。

4.1 タスクテーブルの評価の方法

ここでは、タスクテーブルをどのようにして利用して、タスクの割り当てと実行順序を決定するかを説明する。プログラムの起動時や実行途中の手順は次のようなものとなる。

1. プログラムの起動時には、ドメインマネージャからの情報により利用可能なノードの数を知る。
2. タスクの順番の小さい順に利用可能なノードの数だけ割り当て対象とする。
3. 2. で選択されたタスク間で計算量に差があり、なおかつ利用可能なプロセッサの性能にも差がある場合は、計算量が多いタスクを性能の良いノードに割り当てる。
4. 3. と同様に、必要メモリ量やその他の情報についても検討を行なう。
5. 利用可能なノード数が増加した場合やあるタスクの終了により空きノードができた場合などは、2 から 4 までの手順を繰り返す。

この方法により、並列プログラムの持つ特性を実行環境と共に利用して、実行環境に柔軟に対応できる。

5 まとめ

分散処理を効率的に行なうために、プログラムとシステムとのインターフェースとなるタスクテーブルを定義し、その利用方法を提案した。これによりプログラムの特性と実行環境の特性の両方を考慮した効率の良い実行が可能となるはずである。また、研究の前提としているドメインはワークステーション・クラスタだが、分散メモリ方式の Multicomputer 等にも適用できると考えられる。今後、より最適に近い実行のためのタスクテーブルの追加情報や利用の方法について検討する。また、アプリケーション・プログラムを使ってシミュレーションも行なう予定である。

参考文献

[1] G.A. Geist and V.S.Sunderam : Network-Based Concurrent Computing on the PVM System. CONCURRENCY: PRACTICE AND EXPERIENCE, VOL. 4(4) (1992)

[2] William Gropp, Ewing Lusk and Anthony Skjellum : USING MPI. The MIT Press (1994)

[3] Vibha A. Dixit-Radiya and Dhabaleswar K. Panda : Clustering and Intra-Processor Scheduling for Explicitly-Parallel Programs on Distributed-Memory Systems. Proc. 8th Intl. Parallel Processing Symposium, Mexico (1994)

[4] Ishfaq Ahmad, Arif Ghafoor and Geoffrey C. Fox : Hierarchical Scheduling of Dynamic Parallel Computations on Hypercube Multicomputers. Journal of Parallel and Distributed Computing, 20 (1994)

[5] 天野慎太郎 : 並列分散オペレーティングシステムに関する研究, SLL940053, 曽和研究室テクニカルレポート (1995 年 1 月 27 日)