社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－HPC－111 (7)
2007／8／1

不均質環境におけるタスクネットワークの静的スケジューリング手法

片野 聡[†] 森 英一郎[†,☆] 大野和彦[†]
佐々木 敬泰[†] 近藤利夫[†] 中島 浩[††]

我々はメガスケールの並列処理を想定したタスク並列スクリプト言語 MegaScript を開発している．不均質な環境において最大限に性能を発揮するには，各ホストの負荷バランスや通信時間の削減などを考慮したスケジューリングが必要である．しかし，MegaScript で扱うプログラムはタスク数が膨大であると想定されるため，真に最適なスケジューリングを求めるのは現実的ではない．そこで，膨大なタスク数でも効率よくスケジューリングを行う手法を提案する．提案手法では，まず，同一ドメイン内のホストをホストグループとし，関連の強いタスクをタスクグループとしてクラスタリングを行う．そして，計算コストなどの静的な情報をもとに，タスクグループ単位でホストグループに効率よく割り当てる．大規模なタスク数とホスト数を想定したシミュレーションを行った結果，単純な動的負荷分散方式と比べて，実行速度が向上することを確認した．

A Static Scheduling Scheme of Task Network on Heterogeneous Environment

Satoshi Katano,[†] Eiichirou Mori,[†,☆] Kazuhiko Ohno,[†]
Takahiro Sasaki,[†] Toshio Kondo[†] and Hiroshi Nakashima[††]

We are developing a task-parallel script language named MegaScript for mega-scale computation. To obtain high performance in heterogeneous environment, a scheduling scheme considering load-balancing and communication cost is required. However, optimal scheduling is difficult because the number of scheduled tasks is extremely large. In our scheduling scheme, hosts in the same domain are regarded as a host group, and tight-coupled tasks are regarded as a task group. Using computing cost, we allocate these task groups to the host groups. The result of large scale simulation shows that our scheme improves the execution efficiency compared to the dynamic load balancing scheme.

1. はじめに

近年，複雑な物理系が絡み合う環境・気象シミュレーションや災害シミュレーション等では Pflops 以上の計算能力が求められている．そのため，100 万台規模のプロセッサを用いたメガスケールコンピューティングが必要である．現在，専用並列計算機では多くのプロセッサを利用して数百 Tflops の能力を実現しているが，この延長でメガスケールコンピューティングを実現するためには膨大な電力と巨大な施設が必要となる．それらの資源を効率よく利用するためのプログラミング環境として，「タスク並列スクリプト言語 MegaScript」を開発している[1)~4)]．

MegaScript では，並列処理の単位となるタスクを大量に生成し，並列・並行に実行する．このため，システムを構成する各ホストに対しどのようにタスクを割り当て，どのような順序で実行するかという，スケジューリングを考える必要がある．メガスケール環境において最大限に性能を発揮するには，各ホストの負荷バランスやホスト間通信量の削減などを考慮したスケジューリングが不可欠である．しかし，MegaScript で扱うプログラムはタスク数が膨大であると想定されるため，あらゆる割り当て方を探索して真に最適なスケジューリングを行うのは要する演算時間の観点から現実的ではない．

そこで，我々はタスクネットワーク構造のモデル化を行い，その上で並列実行可能部分を抽出し，それらを並列実行するスケジューリングを提案している[5),6)]．しかし，広域分散などの不均質な環境について考慮していなかったため，プログラムによっては適切なスケ

† 三重大学
Mie University
†† 京都大学
Kyoto University
☆ 現在，船井電機株式会社
Presently with Funai Electric Co., Ltd

ジューリングを行えない場合がある．

本稿では，広域分散環境に対応したスケジューリング手法を提案する．まず，同一ドメイン内に存在する同一性能の複数のホストをホストグループとし，関連の強いタスク群をタスクグループとしてクラスタリングを行う．そして，計算コストやホストの性能などの静的な情報をもとに，タスクをホストグループに割り当てるスケジューリング手法を提案する．

2. MegaScript の概要

MegaScript は，外部プログラムをタスクとして定義し，その間をストリームと呼ばれる通信路でつないだタスクネットワーク構造を記述するための言語である．タスクを並列実行させるのに必要なオーバヘッドは全体に対してわずかであるため，実行効率より記述のしやすさを優先し，Ruby[7] をベースとするスクリプト言語としている．

2.1 タスクとストリーム

タスクは MegaScript の並列実行単位である．タスクの内部の処理に MegaScript は関与しない．

ストリームは，あるタスクの標準出力の内容を，他のタスクの標準入力に流し込むための通信路である．一つのストリームの入出力にそれぞれ複数のタスクを接続することで，一対多，多対多などの通信を簡潔に実現できる．ストリームの入力端に複数のタスクを接続した場合，メッセージはマージされる．また，出力単に複数のタスクを接続した場合，メッセージは接続した全タスクにマルチキャストされる．

タスクとストリームは，それぞれ Task と Stream クラスのオブジェクトとして表される．スケジューリングは，このオブジェクトを操作して行う．

また，同じ種類のタスクを複数生成する場合は，タスク配列として生成できる．それぞれのタスクに別々の引数を与えることも可能であり，TaskArray クラスのオブジェクトとして表される．

複数のストリームをまとめて生成する場合は，ストリーム配列として生成できる．同じ数のタスク配列をストリーム配列に接続すると，個々のタスクが個々のストリームに接続される．StreamArray クラスのオブジェクトとして表される．

タスク配列やストリーム配列は，オブジェクト 1 つで複数のタスクやストリームの情報が縮退されているため，同種のオブジェクトを生成する際は，これらを用いて記憶資源の節約を行うことが可能である．

MegaScript のプログラム (のタスクネットワーク定義部分) の例を**図 1** に挙げる．この記述により，**図 2** のようなタスクネットワークが生成される．

```
t1 = Task.new("T1")
t2 = Task.new_array(N,"T2")
t3 = Task.new_array(N,"T3")
t4 = Task.new("T4")
s1 = Stream.new()
s2 = Stream.new_array(N)
s3 = Stream.new()

s1.connect(t1, IN)
s1.connect(t2, OUT)
s2.connect(t2, IN)
s2.connect(t3, OUT)
s3.connect(t3, IN)
s3.connect(t4, OUT)
```

図 1 タスクネットワークの定義例

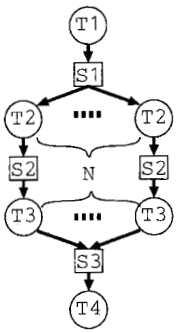

図 2 タスクネットワーク

2.2 メタプログラムとメタ情報

適切なタスクスケジューリングを行うには，タスクの計算時間や通信量などの情報をできるだけ正確に知る必要がある．しかし，タスクは任意の言語で記述された外部プログラムなので，MegaScript が常にこれらの情報を取得できるとは限らない．

そこで，MegaScript ではタスクに関する情報の記述方式としてメタプログラムを用いる．メタプログラムは元のプログラムの制御構造と計算処理，入出力処理を抽象化して記述するものである[2),4)]．メタプログラムは，記述方式としてプログラムを用いているため高い記述性を持ち，ユーザのタスクプログラムに関する知識や，かけられる手間に応じて，詳細な記述もトップレベルの大雑把な構造のみの記述も可能である．**図 3** に，メタプログラムの記述例を示した．

メタプログラムを静的解析して得られた情報から，メタモデルが生成される．メタモデルはタスク毎に存在し，対応するタスクのメタ情報を持つ．具体的には，計算コスト C_c，入力コスト C_i，出力コスト C_o を得る．これらのコストは静的に求まる場合は数値となる

```
input(1)
FOR @arg[0]
  compute(100)
END
output(10)
```

図 3　メタプログラムの記述例

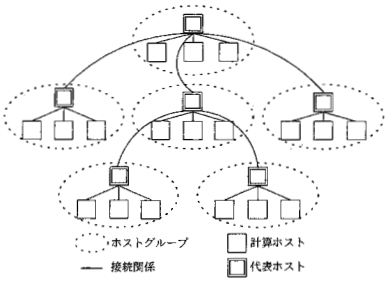

図 4　実行環境の階層型モデル

が，静的には求まらないケースもある．例えば，ループの回数がタスクの実行時引数によって決まるケースや，入力があるたびに処理を行うようなタスクのケースである．この場合，コスト値はコスト関数として表される．従って C_c，C_o は，数値か，実行時引数と入力コスト C_i のコスト関数である．

図 3 の場合，C_c は $100 \times arg[0]$ として実行時引数の関数として得られ，C_o は 10 という数値が得られる．

2.3　実行環境の階層型モデル

MegaScript の実行環境は，階層型モデルを採用している[8]．図 4 のように，いくつかのホストをまとめたものがホストグループとなっており，各グループの代表ホストが直下の計算ホストとホストグループを制御している．

2.4　スケジューリング

スケジューラはタスクネットワークの解析情報を利用してスケジューリングを行う．MegaScript プログラムで生成されたタスク／ストリームオブジェクトは，ユーザが create メソッドを呼ぶとスケジューリング待ちキューに移動する．その後，schedule メソッドを呼ぶと，スケジューラはスケジューリング待ちキューにあるオブジェクト全ての配置ホストと実行順を決定する．配置されたオブジェクトは，決定した順番通りに，配置先のホストで実行される．

3.　従来手法の問題点と解決方針

従来手法[6] は，タスクネットワーク上の並列実行可能なタスク群をグループ化する．そして，計算コストをタスクネットワーク上で伝搬させて依存関係により待たされる時間である依存コストを算出する．最後に，その結果を用いてホストの最終完了時刻を短くする静的スケジューリングを行っている．しかし，以下のような問題が存在する．

(1)　実行環境の階層型モデルを意識せず，大規模なタスク数やホスト数に対して，タスクネットワークの解析・スケジューリングを行うため，非常に時間がかかる．

(2)　ホスト性能や通信性能は均質であると仮定してスケジューリングを行う．そのため，不均質な環境では配置後の実行時間が予測と異なり，依存関係により待たされる時間が変わるため，最終完了時刻が必ずしも短くはならない．

そこで，(1) を解決するために，タスクの情報を縮退したタスク配列のまま解析を行い，ホストグループに対してスケジューリングを行う．そして，(2) を解決するために，ホストの性能など環境を考慮する．

提案手法は，通信による結合度の高いタスク順からグループ化し，タスクグループ単位でホストグループに配置することで，結合度の高いタスクが分割されて，低速な通信路を通るのを防ぐ．さらに，不均質な性能では依存コストを正しく算出することが難しいため，タスクネットワークを縦方向に分割することで，依存による待ち時間を生じにくくする．

4.　提 案 手 法

今回提案する手法は，実行環境の階層型モデルを意識して，タスクグループをホストグループに対し割り振るスケジューリングを行う．

階層的にスケジューリングを行い，制御下にあるホストまたはホストグループのみに，スケジューリングを行うことで，スケジューリングコストの分散・削減を行う．ホストの性能と台数を考慮して，ホストグループごとの計算負荷が均等になる手法を提案する．

4.1　通信コスト

通信コストは，ストリームに流れる出力コストを合計したものである．出力コストが実行時引数や入力コストの関数で決まる場合は，あらかじめタスクネットワーク上で伝搬させていき求める．

4.2　タスクのグループ化

MegaScript では扱うタスク数が多い．しかし，大部分はタスク配列を用いた同一種のタスクである可能性が高いため，タスク配列をそのまま扱うことで，スケジューリングコストの削減を行うことができる．また，スケジューリングの際に計算負荷の均等化のため，タスクグループを分割するが，通信コストの大きい部

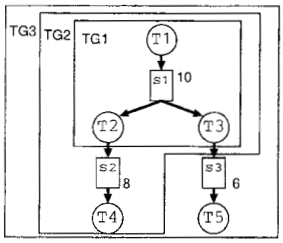

図 5　タスクのグループ化

分からグループとしてまとめ，外側のグループから分割することで，通信コストの大きい部分は分割されにくくなる．

グループ化は以下のように行う．

(1)　ストリームに流れる通信コストの最も多いストリーム，ストリーム配列を探す．

(2)　そのストリームに繋がれているタスク，タスク配列またはタスクグループとストリームのすべてをまとめて 1 つのグループとする．

(1) と (2) をタスクネットワーク全体が 1 つのグループになるまで繰り返す．**図 5** はタスクネットワークにこのアルゴリズムを適応した例である．ストリームの右の数字を通信コストとする．MegaScript では，各タスクの入力側，出力側に 1 本しかストリームを接続できないため，このような手順を繰り返し行うと，通信量の多い部分が内側のグループに入り，少ない部分が外側のグループに入る．

これにより，スケジューリングでタスクグループを分割する際，外側の部分を優先して分割するようにすれば，通信量の多い部分をまとめて同一のホストグループに配置できるため通信の効率を上げることが可能になる．また，グループ化する際に，タスクグループ内のタスクの計算コストの合計を求めておき，タスクグループの計算コストとする．

ストリーム配列がネットワーク構造に含まれている場合は，**図 6** のように，ストリーム配列に接続されているタスク配列の個々のタスクとストリームのタスクグループが一つの大きなタスクグループになったものになっていると考える．

4.3　ホストのグループ化

スケジューリングコスト削減のために，ホストのグループ化を行う．まず，ドメイン単位で 1 つのホストグループとし，階層的にグループ化を行う．その上で，同一性能の計算ホストをグループ化し，最下層が均質環境になるようにする．また，ホストをグループ化する際は，ホストグループの性能の平均を求めておく．一般に上層になるほどネットワークは低速になるため，

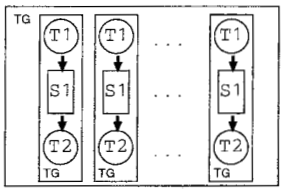

図 6　ストリーム配列のグループ化

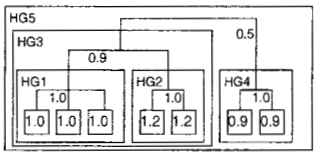

図 7　ホストのグループ化

タスクグループ単位でホストグループに配置することにより，頻繁に行われる通信が高速なネットワークを通るようになり，全体の通信時間を短くすることができる．**図 7** は 7 台の計算ホストをグループ化した例である．ホストの数字は計算性能を表し，ネットワークの近くの数字は，通信性能を表している．

また，同一性能の計算ホストをグループ化することにより，計算ホストへの配置の際に，従来手法の依存コストを用いて，スケジューリング性能の向上を狙うことが可能になる．

4.4　スケジューリング手法

本スケジューリング手法は，計算コストをホストの性能や台数に応じて均等に分割することを目指している．さらに，タスク間の依存がある部分や通信の多い部分がホストグループ間にまたがって配置されにくいように工夫している．なお，

- 性能や計算コストなどの静的な情報の精度は完全なものである．
- タスクはマイグレーションしない．
- 同じホストで複数のタスクを並行実行しない．
- ホストと通信の性能は変化せず，安定している．
- タスクネットワークにループ構造が存在しない．

を前提条件としている．また，本手法では，ホストグループの各々にスケジューラを配置して，階層型スケジューリングを行い，各スケジューラは直下のホストグループへのスケジューリングのみ担当する．直下のホストグループの集合を $G = \{G_1....G_n\}$ とする．ホストグループ G_i の性能 P_i はホストグループ内のホストの性能をすべて合計したものである．G のすべてのホストグループの合計 $P_1 + P_2 + ... + P_n$ を P_{sum} とする，ホストの性能の平均 $P_{ave,i}$ は，P_i を G_i 内

の全計算ホストの台数で割ったものとする．

(1) ホストグループに割り振る計算コストの算出
  (a) $P_{ave,i}$ が大きい順にホストグループをソートする．
  (b) タスクグループの計算コストを C として，各ホストグループに割り振る計算コスト $H_i = C * P_i / P_{sum}$ を求める．
  (c) C のストリームの入力側のタスクの合計コスト C_{in}，出力側の合計のコスト C_{out} を計算する．

(2) ソート結果にしたがい，$P_{ave,i}$ が最も大きいホストグループを一つ取り出す (G_i とする)．これにタスクグループを割り振る．
  (a) ストリームの IN 側のタスクグループの集合を $I = \{I_1....I_m\}$，ストリームの OUT 側のタスクグループの集合を $O = \{O_1....O_l\}$ とする．
  (b) $H_{in} = C_{in}/C * H_i, H_{out} = C_{out}/C * H_i$ を求め，入力側と出力側それぞれに割り振るべき計算コストを求める．
  (c) ストリームの入力側について割り当てる．以下，出力側についても同様に割り当てる．微小な正数である閾値 L を設定する．
  (d) I から計算コストが最大のタスクグループを取り出す (I_j とする)．
  (e) H_{in} から I_j の計算コストを引く．
  (f) $H_{in} > L$ ならば，割り当てて (d) に戻る．
  (g) $-L \leq H_{in} \leq L$ ならば割り当て，次のホストグループに対して (2) から実行する．
  (h) $H_{in} < -L$ ならば，割り当てずに，そのタスクグループの一つ内側に対して (2-a) から実行する．
  (i) 最も内側のタスクグループになれば，次のホストグループに対して (2) から実行する．

これにより，ホストグループの性能と台数に応じた分の計算コストを割り振ることが可能になる．性能の大きい順に計算コストが大きいものから割り振ることによって，負荷を分散しやすくしてある．また，通信の多い部分はタスクグループの内側にあるため，分割せずに済み，通信を同じホストグループ内に閉じこめることが可能になる．さらに，ストリームの入力側タスク，出力側タスクをホストグループの性能に応じ，同じ比率で配分されるため，タスクネットワークを縦方向に分割していることになる．このため，依存関係のある部分が他のホストグループに配置されることを防いでいる．

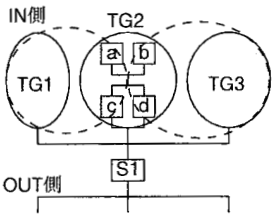

図 8 スケジューリングの例

図 8 を用いて説明する．ホストグループが 2 つあって，ストリームの IN 側の計算コストを 1:1 で割り振りたいとする．TG1 を割り振り，TG2 も割り振ると閾値の範囲を超えてしまう場合，TG2 の一つ内側のタスクグループに対し，縦方向に分割することで，より均等に割り振ることができる．OUT 側も同様に行う．

5. シミュレーション評価

提案手法の効果を確かめるために，大規模なタスク数とホスト数を想定したシミュレーションを行った．今回は，ホストグループへのスケジューリングまでの性能を確かめるため，全体を動的負荷分散で行ったものと，提案手法でホストグループまで配置し，計算ホストへの配置は動的負荷分散で行ったものを比較する．

5.1 シミュレーションモデル

4.4 節の前提条件に以下を加える．

- 計算・通信性能の単位は [コスト/s] とする．
- タスクは実行終了時にすべての通信を行う．
- タスクは依存しているタスクのすべてが終了し，通信にかかる時間が経過後，実行可能になる．
- 依存関係によりタスクが実行できない場合は，ホストは実行可能になるまでアイドル状態である．
- ホストとスケジューラの間の通信時間・タスク生成時間を考慮しない．
- 通信負荷などのオーバーヘッドは考慮しない．
- タスクの通信が同一ホスト上で行われる場合の通信時間は 0 とする．

動的負荷分散方式は，依存により実行できなくなることを防ぐため，あらかじめ，タスクフローにそってソートしておく．タスクの実行が終了したら，次のタスクの実行を行う．

5.2 シミュレーション環境

1024 台のホストが 19 のホストグループに分かれ，これらのグループが 3 階層に接続された環境を想定した．計算性能は，0.5 から 1.7 の間で，通信性能は上

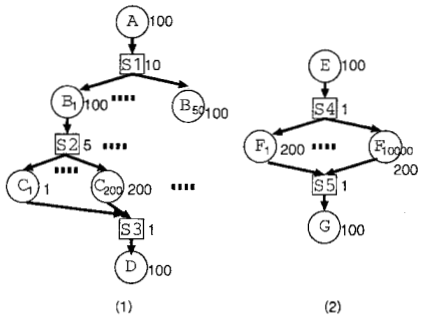

図 9　シミュレーション対象のタスクネットワーク

表 1　評価結果

タスク	動的負荷分散	提案手法	提案手法/動的負荷分散
(1)	2062.79s	1532.28s	0.74
(2)	2255.90s	2157.65s	0.96

層に行くほど遅くなるように設定し，最下層では 1.0 で最上層では 0.05 となっている．

5.3　結果と考察

図 9 の 2 つのタスクネットワークについてシミュレーションを行った．タスクの右の値が計算コストで，ストリームの右の値が通信コストとなる．(1) では，タスク B は 50 個生成し，計算コストはすべて 100 である．タスク C はタスク B 1 つにつき 200 個ずつ計 10000 個生成し，計算コストは 1 から 200 とそれぞれ異なる値になっている．(2) は単純で通信の少ないタスクネットワークで，F を 10000 個生成している．結果は，表 1 の通りである．

(1) では，提案手法が動的負荷分散手法にくらべ 26 %程度高速になっている．単純な動的負荷分散方式では，通信の多い B から C の通信が別のホストグループに配置されて，通信に時間がかかり，その分遅くなっている．逆に，提案手法では，B と C の依存部分は同じホストグループに配置される可能性が高く，その差が現れたと考えられる．

(2) では，計算ホストと台数を考慮し，計算コストが配分されており，動的負荷分散と比べ，同等以上の性能が出ている．提案手法では G のタスクが高速なホストに配置される分，速くなっていると思われる．

大域的な動的負荷分散では，スケジューリングを行うホストがボトルネックになりやすい．しかし，提案手法はスケジューリングが階層化されており，スケジューリングコストは，直下の計算ホストとホストグループに対して，タスクグループを割り振るだけでよい．したがって，スケジューリングコストが削減・分散されておりボトルネックになりにくく効率がよい．

6. おわりに

本稿では，タスク並列スクリプト言語 MegaScript 用のタスクスケジューリング手法として，階層モデルに適した，不均質環境向けのスケジューリング手法について述べた．ホストの計算性能とタスクの計算コストに応じたホストグループへの配置や，通信をホストグループの内側に閉じこめることで，不均質な大規模環境で大量のタスクを効率よく並行実行できる．シミュレーションによる評価の結果，本手法が動的負荷分散方式より高速であることを確認できた．

今後は，計算ホストへの配置に従来手法の依存コストを考慮した静的スケジューリングを行い，今回の提案手法と組み合わせたハイブリッドなスケジューリングを課題に取り組んで行きたい．

謝辞　本研究の一部は文部科学省科学研究費補助金（特定領域研究，研究課題番号 19024041,「高性能計算の高精度モデル化技術」）による．

参考文献

1) 大塚保紀, 深野佑公, 西里一史, 大野和彦, 中島浩: タスク並列スクリプト言語 MegaScript の構想, 先進的計算基盤システムシンポジウム SACSIS2003, pp. 73–76 (2003).
2) 大塚保紀, 大野和彦, 中島浩: タスク並列スクリプト言語 MegaScript によるタスク動作モデル記述, 情処研報 2003-HPC-95, pp. 113–118 (2003).
3) 西里一史, 大野和彦, 中島浩: タスク並列スクリプト言語 MegaScript のランタイムシステムの設計と実装, 情報研報 2003-HPC-95, pp. 119–124 (2003).
4) 湯山紘史, 津邑公暁, 中島浩: タスク並列言語 MegaScript 向け高精度実行モデルの構築, 情報処理学会論文誌:コンピューティングシステム,Vol.46, No.SIG12(ACS11),pp. 181–193 (2005).
5) 森 英一郎, 大野 和彦, 佐々木 敬泰, 近藤 利夫, 中島浩: タスクネットワークの形状に基づく並列スクリプト言語のスケジューラ, 情処研報 2004-HPC-100, pp. 19–24 (2004).
6) 片野 聡, 森 英一郎, 大野 和彦, 佐々木 敬泰, 近藤 利夫, 中島浩: タスクネットワークの解析情報を用いたスケジューリング手法, 情処研報 2006-HPC-107, pp. 61–66 (2006).
7) まつもとゆきひろ, 石塚圭樹: オブジェクト指向スクリプト言語 Ruby, ASCII (1999).
8) 高木 祐志, 西川 雄彦, 大野 和彦, 佐々木 敬泰, 近藤 利夫, 中島浩: タスク並列スクリプト言語処理系における広域分散実行方式, 情処研報 2006-HPC-107, pp. 79–84 (2006).