Vol.2010-HPC-126 No.6
2010/8/3

大規模ワークフローを対象とする動的スケジューリング手法における静的情報の利用

松本真樹[†1]　佐々木敬泰[†1]　大野和彦[†1]
近藤利夫[†1]　中島浩[†2]

ワークフロー型の大規模並列処理において実行効率を高めるにはタスクスケジューリングが重要であり，様々な手法が提案されている．静的スケジューリング手法は最適解に近い結果を得ることができるが，計算ホストの負荷等による性能変動が発生した場合，ワークフローの実行効率が低下する可能性がある．一方デマンドドリブン型の動的スケジューリング手法は，ワークフローの依存関係がボトルネックになり十分な効果を得られない場合がある．そこで，実行環境の性能変動が発生する毎に静的スケジューリング手法で再スケジューリングを行う手法が提案されている．しかし，静的スケジューリング手法の実行回数が増加することで計算コストが膨大となる．

そこで本稿では，性能変動やタスクの終了といった一定の条件に従って静的スケジューリング手法を実行する動的スケジューリング手法を提案する．抽象シミュレーションにより本手法を評価した結果，タスク数が 1000 規模のワークフローの場合で，再スケジューリングの実行回数を 1/4〜1/100 に減らしつつワークフローの実行効率を維持することができた．

Dynamic Scheduling Scheme using Static Information for Large-Scale Workflows

Masaki Matsumoto,[†1] Takahiro Sasaki,[†1]
Kazuhiko Ohno,[†1] Toshio Kondo[†1]
and Hiroshi Nakashima [†2]

Task scheduling is very important for efficient execution of large-scale workflows on distributed computing environments. Various scheduling schemes are proposed, including dynamic rescheduling schemes for environments which performance change dynamically. In dynamic rescheduling schemes, static scheduling schemes are executed if the performance is changed. Thus they achieve better performance in workflow applications. However, the scheduling cost increases if the performance is changed frequently.

Therefore, we propose a dynamic scheduling scheme, reducing the number of rescheduling. The evaluation using an abstract simulation shows that the execution times are reduced to approximately 1/4 to 1/100 without decrease in execution efficiency.

1. はじめに

近年, 大規模計算の需要がますます増大する一方で, 単一プロセッサでの性能向上は頭打ちになりつつあり, 並列処理への期待が高まっている．特にコストパフォーマンスやスケーラビリティの面で大きな利点があるため，広域ネットワーク上に分散しているクラスタ群を利用することに注目が集まっている．

大規模なワークフロー型の並列処理で高いスループットを得るには，実行単位となる各タスクをどの計算機でどの順番で実行するかというタスクスケジューリングが重要になる．こういったタスク間の依存関係を考慮したスケジューリング手法は多く提案されている．しかし，ワークフロー型並列処理全体の実行時間であるスケジューリング長において良い結果を得る従来手法は計算コストが高く，タスク数・ホスト数が大規模な並列処理では現実的な時間で解くことは難しい．そこで我々は, 複数のスケジューラを階層的に配置し, 上位層と下位層で異なるスケジューリング処理を行う階層型スケジューリング手法 [1] や，下位層のスケジューラで扱うタスク間の依存関係に着目し, 適応的にスケジューリング処理を切り替える, 適応型スケジューリング手法[2] を提案している．これらの手法により，スケジューリングを現実的な時間で行うことができる．

しかし，これらの手法はワークフローを実行する前にスケジューリングを行う静的スケジューリング手法であり，実行時のホストの性能の変化については対応できていない．性能の変化に対応している手法としては，デマンドドリブン型のスケジューリング手法が提案されているが，これらはワークフローの依存関係によるボトルネックを考慮していないため，静的スケジューリング手法と比較して実行効率が低下する恐れがある．一方で，最適解に近い答えを出す静的スケジューリング手法を性能変化が発生する毎に呼び出し再スケジューリ

†1 三重大学大学院 工学研究科
Mie University
†2 京都大学 学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

© 2010 Information Processing Society of Japan

ングを行うことで，ワークフローの実行効率を高める手法が提案されている．しかしこれらは計算コストが非常に高く，大規模ワークフローで利用することは現実的でない．

そこで我々は，高速な適応型スケジューリング手法[2] を再スケジューリングの処理に用いた，再スケジューリングを行う動的スケジューリング手法を提案する．そして，大規模なワークフロー型の並列処理を目的とするタスク並列スクリプト言語 MegaScript[3)–5)] に実装し抽象シミュレーションで評価を行った．

以下，2 章では背景である MegaScript の概要とスケジューリングについて述べる．3 章では従来のスケジューリング手法を紹介し，我々の目的との適合性について議論する．4 章では提案手法を述べ，5 章でシミュレーションによる提案手法の評価結果を示す．最後に，6 章でまとめを行う．

2. 背　　景

2.1 並列スクリプト言語 MegaScript

MegaScript は 2 階層並列モデルの上位層を記述する言語である．逐次や並列の独立したプログラムを計算タスクとして扱い，これらのタスクを並行・並列に実行させることで並列処理を行う．また，ストリームと呼ばれる通信路を介することで，各タスク間のデータ通信を行う．計算の中心となる各タスクはネイティブプログラムとして用意するため，MegaScript プログラムでは，タスクやストリームから成るタスクネットワークと呼ばれるワークフローやタスクの実行に必要な情報等を記述する．MegaScript ではこれらの情報を解析し，スケジューリング結果に従って各タスクを指定された計算機で実行する．

ここで未知の病原菌の DNA 配列群を既知の病原菌のデータと比較し特定を行い，またその結果の相互比較を行うタスクネットワークの例（図 1）を示す．まず未知の DNA 配列群に対し *BLAST*[6] や *FASTA*[7] を使い既知のシーケンスデータベースとの相同性検索を行い，その結果に対して *ClustalW* [8] でマルチプルアライメントを行い樹形図を得る (図 1 の太枠)．これを複数のシーケンスデータベースで行い，シーケンスデータベースごとに得られた樹形図を *ClustalW Panel* [8] に読み込ませ相互比較を行う．

2.2 タスクスケジューリング

一般のワークフローは Directed Acyclic Graph(DAG) で表現ができるが，MegaScript のようにタスクの動的生成が可能なシステムもある．しかし，本稿では議論を簡単にするために，ワークフローは予め与えられており動的に変化しないものとする．従って本稿で扱うスケジューリング手法は，DAG のタスクスケジューリングの一種になる．一般の DAG の

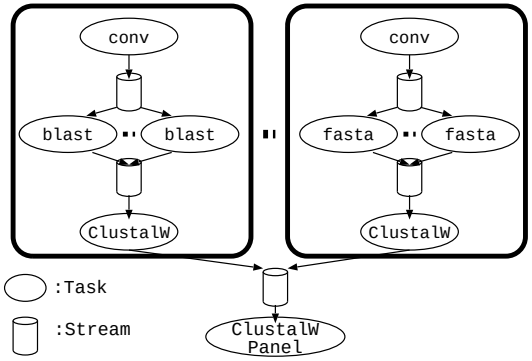

図 1 タスクネットワークの例
Fig. 1 Example of Tasknetwork

形を取るタスクスケジューリングとして，3 章で述べるように，様々な従来手法が提案されている．大規模ワークフローを対象とした場合，これらの手法は一長一短であるが，実用化においてはそれぞれの利点を両立させることが必須である．

ワークフロー型の大規模並列処理を容易に実行でき，また効率よく自動で実行するようなシステムの需要は高い．しかし本稿で扱うタスクスケジューリングでは，以下の特徴を考慮する必要がある．

(1) 一般のシステムでは MegaScript のようにネイティブプログラムを扱うことも多く，タスクの分割やタスク実行中のマイグレーションは行えない．
(2) 実行環境として広域分散するホスト群を想定しており，計算性能・通信性能は非均質である．
(3) 大規模な並列処理を目的としており，10 万 ∼100 万規模のタスクやホストを扱えることが求められる．
(4) 独立したプログラムを粗粒度のタスクとして組み合わせ，ユーザが明示的にワークフローを記述する．このためデータの分散・収集や処理の流れの分岐といった，比較的単純なパターンの組み合わせになる．

ワークフロー型の大規模並列処理では DAG のタスクスケジューリングが必要になり，(2), (3) の特徴により計算コストは非常に大きくなる．対して (4) の特徴により複雑な DAG の記述が難しいため，直並列グラフに近い単純なワークフローになることが多いと考えられる．そこで適応型スケジューリング手法では，同一の先行/後続タスクを持ち互いに依存関

© 2010 Information Processing Society of Japan

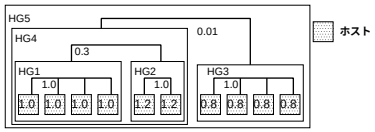

図 2 ホストの階層化
Fig. 2 Hierarchy of Host Groups

係の無いタスクの集合 (以降**独立型タスク群**と呼ぶ) に着目し，これらの処理を簡略化しスケジューリングの計算コストを削減している．

2.3 階層型・適応型スケジューリング手法

我々は 2.2 節であげた特徴を考慮した階層型スケジューリング手法や適応型スケジューリング手法を提案し，スケジューリング時間の大幅な削減を達成している [1),2)]．本節では階層型スケジューリング手法や適応型スケジューリング手法について説明する．

2.3.1 実行モデル

大規模な広域分散環境の利用例として，SETI@home[9)] 等がある．これは家庭などに存在する PC を集めて利用するもので，各ホストの計算性能や通信性能は完全に非均質である．こうした環境は大量の計算機を確保できるが，ほとんどのホスト間で高速な通信を期待できず，また所有者の都合により頻繁かつ予告なくホストが利用できなくなる問題がある．このため，完全に独立した大規模問題には有効ではあるが，タスク間通信が頻繁に起きる並列処理には適していない．

我々が想定するワークフロー型の大規模並列処理では，PC クラスタのようにある程度の規模・品質の計算資源が多数，広域に分散した環境が現実的な実行環境と考えられる．この場合，各クラスタ内の通信性能は高速かつ均質である．したがって，我々が提案した階層型スケジューリング手法はこのような環境を想定している．

実行環境のモデル化について，相互通信性能の高いホスト同士が内側へ来るように階層的にグルーピングを行う．したがって上位層のホストグループほど通信性能は低くなる．

10 台のホストからなる 3 つのクラスタを階層化した例を，図 2 に示す．ホストの中の値は計算性能を，ネットワーク付近の値は通信性能を，それぞれ表す．

2.3.2 階層型スケジューリング手法

階層型スケジューリング手法では，最内側のクラスタを対象に実行する局所スケジューリング手法と，それ以外の上位層に対して行う大域スケジューリング手法の二種類を用いる．これにより，全体のタスクネットワークを高速にスケジューリングすることができる．大域スケジューリング手法では直下のホストグループに対するタスクの分配のみを行う．一方，局所スケジューリング手法では扱うタスク数やホスト数が大幅に削減されているため，従来手法のような良いスケジューリング長を得ることができる．

2.3.3 適応型スケジューリング手法

2.2 節で示したようにタスクネットワーク中には多くの独立型タスク群を含むと考えられる．したがって，各クラスタに分配されるタスクの多くも独立型タスク群の一部である可能性は高い．そこで各独立型タスク群を疑似タスクに縮約し，従来の DAG のタスクスケジューリング手法を利用する．本手法の局所スケジューラでは，DAG のタスクスケジューリング手法として比較的精度が高く，スケジューリング時間の短い HEFT を用いた．

2.2 節で述べたように，独立型タスク群は互いに依存関係は無い．したがって，このタスク群をスケジューリングする際，DAG タスクスケジューリング手法ではなく，独立タスクスケジューリング手法を利用することができる．本手法の適応型スケジューリング手法では，MinMin ヒューリスティック法を簡略化し利用している[2)]．

3. 関 連 研 究

本章では DAG のタスクスケジューリングについて概観し，大規模ワークフローへの適用可能性について検討する．

3.1 静的スケジューリング手法

階層型スケジューリング手法や適応型スケジューリング手法は DAG の静的スケジューリング手法に分類される．DAG の静的スケジューリング手法は多くの研究がされており，その計算コストを無視すれば最適解に近い答えを出す．従来の DAG の静的スケジューリングで引用回数の多い手法に，Heterogenous Earliest-Finish-Time(HEFT)[10)] や Levelized Duplication Based Scheduling(LDBS)[11)] 等がある．しかし大規模ワークフローのタスクは基本的に静的スケジューリングが可能であるが，MegaScript が想定するような実行環境では計算性能や通信性能が頻繁に変化する．このため，静的スケジューリング手法で良いスケジューリング長を得ても実際の実行時間と一致するとは限らず，高い実行効率を得ることが難しい．また大規模ワークフローを想定した場合，HEFT や LDBS のような静的スケジューリング手法はスケジューリングにかかる計算コストが高くなってしまい，現実時間でスケジューリングを行うことが困難であるという問題がある．

3.2 動的スケジューリング手法

タスク間が完全独立な場合には，マスタワーカー型の単純な動的スケジューリング[12)] が

© 2010 Information Processing Society of Japan

ある．これらは各ワーカーホストがアイドル状態となるたびにマスタホストへタスクを要求するため，実行環境における計算性能や通信性能の変化に柔軟に対応できる．これに対しランダムスチール（RS）法[13]では，あらかじめタスクを全ホストに分配し，実行タスクのなくなったホストが，ランダムに選択したホストにタスクの分配を要求する．この手法ではマスタホストが不要かつ通信回数が最小限で済むため，安定して高性能が得られる．しかしこれらの手法では，ワークフローを扱う場合は大域的なボトルネックを考慮できないため，静的スケジューリング手法と比較して実行効率が低下する恐れがある．

一方，ワークフローを考慮した動的スケジューリング手法として，AHEFT[14]や Blythe らの手法[15]などが提案されている．性能が変化するような実行環境においても静的スケジューリング手法のような最適解を求める手法である．これらは初めに，静的スケジューリング手法のような最適解に近いスケジューリング結果を求める．その後，そのスケジューリング結果にしたがって各タスクを実行していくが，ホストの計算性能や通信性能に大きな変化が発生した場合，未実行のタスクに対して性能変化を考慮した上で再度スケジューリングを行う．そしてこの再スケジューリングの結果に従って，残りのタスクの実行を行う．これにより，ワークフローのボトルネックを考慮しつつ，性能の変化に対応してワークフローを実行することができる．しかしこのような再スケジューリングを行う手法は，デマンドドリブン型の単純な動的スケジューリングと比較して，スケジューリングにかかる計算コストが大きくなり，これがワークフローの実行において大きなオーバーヘッドとなる恐れがある．

本稿では再スケジューリングを行う手法について提案するが，再スケジューリングにおける計算コストを抑えるために，大規模ワークフローでも計算コストが小さい適応型スケジューリング手法をベースに用いた．

4. 提案手法

4.1 概　　要

ワークフロー型の並列処理で実行効率を高めるには，AHEFT[14]のように依存関係を考慮しつつ，性能の変化に柔軟に対応する必要がある．しかし，大規模なワークフロー全体を一つの問題として扱うと，再スケジューリングにかかるコストが非常に大きくなってしまう．そこで階層型スケジューリング手法と同様に上位層と下位層を分け，それぞれ特徴を考慮した異なる動的スケジューリング手法を併用していく．本稿では，3.2 節で紹介したような再スケジューリングを行う手法を下位層の局所スケジューラとして用いた．上位層の大域スケジューラにどのような動的スケジューリング手法を用いるかについては今後の課題である．

局所スケジューラで再スケジューリングを行う手法を用いた場合，実行環境の性能の変化に柔軟に対応しつつ，静的スケジューリング手法のような最適解に近い結果を得ることができる．一方で以下のような欠点が考えられる．

欠点 1　再スケジューリング毎にオーバーヘッドが発生する．

欠点 2　再スケジューリングを繰り返すことで全体のスケジューリングコストが大きくなる．

欠点 1 について，本手法ではランタイムの要求に対してスケジューラが再スケジューリング処理を行った場合，その処理が完了するまでランタイムは結果待ちの状態となり，これがオーバーヘッドとなる．従って大規模ワークフローを想定した場合，HEFT や LDBS のような計算コストの高い手法を再スケジューリングに用いることはオーバーヘッドが大きくなってしまい現実的に難しい．そこで提案手法ではこのオーバーヘッドを小さくするために，スケジューリングの計算コストが小さい適応型スケジューリング手法を用いた．

欠点 2 について，ランタイムの要求に対して頻繁に再スケジューリング処理を行った場合，再スケジューリングに計算コストが小さい手法を用いてもオーバーヘッドが大きくなってしまう．そこで我々は再スケジューリングを行う回数を減らすことを目的とした手法を提案する．

4.2 実行モデル

本節では提案手法の実行モデルについて述べる．本手法の実行モデルでは，各計算ホストでランタイムが起動しており，また各クラスタに一個のスケジューラが起動している．本手法で用いるランタイムは，同時に実行できるタスク数を 1 とした．ランタイムはスケジューラからタスク T_k の実行開始命令を受け取ると，直ちにタスク T_k の実行プロセスを起動させる．またランタイムは計算ホストの性能変動を検知することができ，その情報をスケジューラに通知する．

一方，スケジューラは内部に保持しているワークフローの情報と実行環境の情報からスケジューリングを行う．各タスクの計算量や通信量，タスク間の依存関係といった情報は，ワークフローの情報として扱う．各計算ホストの計算性能や互いの通信性能については，実行環境の情報として扱う．また，スケジューリングを行うためにスケジューラには次のような関数も用意されている．

ChangeTerminated(T_j)　ワークフローの情報のタスク T_j について，実行終了フラグを立てる．このフラグの立っているタスクはスケジューリングの対象から除かれる．

ChangePerformance(H_i, pfm)　実行環境の情報の計算ホスト H_i の計算性能を pfm に設定する．

© 2010 Information Processing Society of Japan

CanStart(T_k)　タスク T_k の全ての先行タスクについて実行が終了していた場合に $true$ を返す．また，先行タスクが存在しない場合も $true$ を返す．上記以外の場合 $false$ を返す．

NextTask(H_i)　ホスト H_i で次に実行するタスクを返す．

CanStart 関数はスケジューラが各計算ホストを効率良く利用するために用いる．一般的な実装では，ランタイムは計算ホスト H_i で実行中のタスク T_k が終了した時点で，スケジューラに次に実行するタスク T_l を問い合わせる．ランタイムはタスク T_l の実行に必要な全てのデータがそろっていれば直ちにタスク T_l をホスト H_i で実行するが，全てのデータがそろっていない場合はタスク T_l の実行に必要なデータがそろうまでホスト H_i はアイドル状態となる．従って長時間計算ホストがアイドル状態になってしまう可能性がある．この問題に対処するため我々が用いるスケジューラでは，ランタイムに次に実行するタスク T_l を通知する前に，CanStart 関数でそのタスク T_l がすぐに実行可能かを調査する．もし CanStart 関数でそのタスクがすぐに実行できない場合，ランタイムにタスク T_l の実行開始命令は送らない．これにより，タスク T_l は再スケジューリングの対象に含まれるため，ワークフローをより効率的に実行することが可能となる．

4.3　アルゴリズム

提案手法のアルゴリズムを図 3 に示す．

このアルゴリズムでは最初に静的スケジューリングを行う．その後，ワークフロー内の全タスクの実行が開始されるまで 8–25 を繰り返す．9 で任意のランタイムからメッセージを受信するまで待機する．タスク T_l の実行完了を知らせるメッセージを 9 で受信した場合，10–11 でスケジューラが保持しているワークフローの情報のタスク T_l の実行終了フラグを立てる．計算ホスト H_j の性能が変動したことを知らせるメッセージを 9 で受信した場合，12–15 でスケジューラが保持している実行環境の情報の計算ホスト H_j の計算性能を変動後の計算性能に変更する．16–18 では後で示す条件に 9 のメッセージが一致するか調べ，条件に当てはまった場合静的スケジューラを起動させ再スケジューリングを行う．19–24 では内部に保持したスケジューリング結果から，各ホストで次に実行するタスク T_k を取得し，タスク T_k が実行可能ならば各ホスト上で動いているランタイムにタスク T_k の実行開始命令を送信する．

大規模ワークフローの実行効率は 16 の条件に大きく依存する．そこで，以下の 3 つの条件を用いて提案手法の実装を行った．

条件 A　任意のホストから性能変動のメッセージを受信した時．

```
 1. 静的スケジューリングアルゴリズムを実行しスケジューリング結果を取得
 2.foreach ホスト Hi in 全ホスト
 3.   タスク Tk = NextTask(ホスト Hi)
 4.   if CanStart(タスク Tk) == true then
 5.     ホスト Hi のランタイムにタスク Tk の実行開始命令を送信
 6.   end
 7.end
 8.while 未実行タスクが存在
 9.   任意のホストからメッセージを受信するまで待機
10.   if タスク Tl の終了メッセージを受信 then
11.     ChangeTerminated(Tl)
12.   else if ホスト Hj の性能変動を知らせるメッセージを受信 then
13.     pfm = ホスト Hj の性能変動後の計算性能
14.     ChangePerformance(Hj, pfm)
15.   end
16.   if 再スケジューリングを行う条件に合致 then
17.     静的スケジューリングアルゴリズムを実行しスケジューリング結果を取得
18.   end
19.   foreach ホスト Hi in 全ホスト
20.     タスク Tk = NextTask(ホスト Hi)
21.     if CanStart(タスク Tk) == true then
22.       ホスト Hi のランタイムにタスク Tk の実行開始命令を送信
23.     end
24.   end
25.end
```

図 3　局所スケジューラのアルゴリズム
Fig. 3　Algorithm of Local Scheduler

条件 B　任意のホストからタスクの終了メッセージを受信した時．

条件 C　任意のホストからタスク T_l の終了メッセージを受信した時．ただしタスク T_l が独立型タスク群 ITS_m に含まれており，ITS_m に未実行のタスクが存在した場合は，再スケジューリングを行わない．

ホストの性能変動の頻度がタスクの実行回数と比較して多い場合，再スケジューリング後にすぐに再スケジューリングが行われる可能性がある．この場合，初めに行った再スケジューリングは無駄になる可能性が高く，スケジューリングのオーバーヘッドが大きくなってしまう．そこで条件 B は性能変動毎に再スケジューリングは行わず，任意のホストからタスクの終了メッセージを受信した時に再スケジューリングを行うことで，オーバーヘッド

© 2010 Information Processing Society of Japan

を抑えることを期待している．

独立型タスク群の各タスクは依存関係が同じである．そこで独立型タスク群を構成するタスク T_i をワークフローから除外しても，元のワークフローと大きな違いはない．またタスク T_i と依存関係が同じ同種のタスクが存在していることから，クリティカルパス上のタスクとなる可能性も低い．従って，タスク T_i が終了した時点で再スケジューリングを行っても，タスク T_i が終了する前に行った再スケジューリングの結果と比較して大きな違いがないことが予想される．そこで条件 C では条件 B よりさらに再スケジューリングを行う回数を減らした．これにより条件 B よりもオーベーヘッドを抑えることを期待している．

5. 評　　価

5.1 シミュレーション方法

本評価では実際にタスクの実行を行わず，以下の条件によりイベントドリブン型の抽象シミュレーションを行う．各タスクの計算時間はそのタスクの計算量に従い，通信時間はそのタスクの通信量に従う．

- 各タスクの通信は，そのタスクの実行終了時に行う．
- 各ホストでは一度に一つのタスクしか実行せず，スケジューリングされた順序は追い越さないものとする．つまり，次に実行するべきタスクが依存関係により実行できない場合，そのホストは当該タスクが実行可能になるまでアイドル状態となる．
- 実行時間は最初のタスクの実行開始から全てのタスクが終了するまでとし，スケジューリング自体に要する時間は考えない．
- タスク間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．

また，ホストの性能変動をシミュレーションするために，ホストの半数に疑似タスクを発生させた．各ホストの計算性能は疑似タスクの数に応じて低下するようにした．また各疑似タスクは予め指定された値を疑似タスクの平均生存時間として，ポアソン分布にしたがって消滅するようにした．一方今回の評価では通信性能の変化は考慮していない．今後は通信性能の変化も含めて，より様々な条件において評価を行う必要がある．

5.2 シミュレーション条件

本評価では以下の手順により，ランダムなタスクネットワークを生成した．

(1) 初期構造として，2 個のタスクを 1 本のストリームで接続したタスクネットワークを生成する．

(2) 以下のいずれかの操作を等確率で適用する．

連結　ランダムにタスクまたは独立型タスク群 T_i を選択する．ただし，タスクネットワークの先頭のタスクは除外する．新しくタスク T_j とストリーム S_k を生成し，T_i を，T_i と T_j を S_k で接続した構造で置き換える．

コントロール並列　ランダムにタスクまたは独立型タスク群 T_i を選択する．ただし，タスクネットワークの先頭と末尾のタスクは除外する．新しくタスク T_j を生成し，T_i と同じ入力側・出力側ストリームを持つように接続する．

データ並列　ランダムにタスク T_i を選択する．ただし，タスクネットワークの先頭と末尾のタスクは除外する．新しく独立型タスク群 T_j を生成し T_i と置き換える．

(3) タスクの総数が設定値以下であれば，(2) から繰り返す．

各タスクの計算量や通信量について，表 1 の範囲でランダムに選んだ値を利用した．タスクの通信量については各評価では用いる CCR 値を使い，

$$\text{タスクの通信量} = \text{タスクの基本通信量} * \text{CCR 値} \quad (1)$$

とする．これにより CCR 値の大きい場合，全体的にタスクの計算量より通信量が大きいタスクネットワークとなり，CCR 値が小さい場合はその逆となる．

実行環境については表 2 の条件に従ってランダムに生成した．BLAST[6] や AMBER[16] の実行時間が通常数百秒以内であり，その出力結果は数百 MB 程度であることが多い．一般的に各クラスタ内については，ギガビット・イーサネット等の高速回線で接続しており，数百 MB のデータ転送時間は数秒程度である．そこで上で設定した計算・通信量から，計算・通信の時間比が BLAST や AMBER の場合に近くなるように設定した．一方，広域ネットワークにおける通信性能はクラスタ内の通信性能より低いことが多く，数百 MB のデータ転送に数十秒から数分程度かかることが多い．また場合によってはそれ以上の時間を要することもある．よって今回の各クラスタ間の通信性能においては，クラスタ内の通信性能の 1/10 ～ 1/1000 に設定した．

それぞれの評価で示すデータは，同じ条件でランダムに生成した 10 種類のデータセットに対しそれぞれシミュレーションを行った平均値である．今後，より広い範囲におけるデータセットで評価を行い，様々な実アプリケーションや実行環境における効果を確認していく必要がある．

5.3 評　　価

5.3.1 性 能 変 動

計算ホストの性能変動の頻度による各条件の効果の違いを明らかにするために，スケジューリング長と再スケジューリングの呼び出し回数による比較評価を行った．図 4 は各条件で

© 2010 Information Processing Society of Japan

Vol.2010-HPC-126 No.6
2010/8/3

表 1 タスクネットワークの生成条件
Table 1 Attributes of Task Networks

タスク数	約 1000
独立型タスク群の大きさ	100
タスクの計算量	100 - 200
タスクの基本通信量	100 - 200

表 2 非均質環境の生成条件
Table 2 Attributes of Heterogeneous Environments

同一グループ内のホスト間通信性能	100
グループの異なるホスト間通信性能	0.1 - 10
ホストの計算性能	1.0 - 5.0
クラスタ当たりのホストの台数	16
クラスタ数	10

スケジューリングを行った時のスケジューリング長を，再スケジューリングを行わない場合のスケジューリング長を 1 として正規化した値である．また各局所スケジューラが再スケジューリングを行った回数の和を表 3 に示す．計算ホストの性能変動をシミュレーションするために，疑似タスクを計算ホストの半数で発生させた．疑似タスクの生成頻度は単位時間に平均 λ 回のポアソン分布に従っている．一方，疑似タスクの消滅は指数分布に従っている．

図 4 の評価結果から，全ての条件において $1/\lambda$ が大きいほどワークフローの実行時間が短くなった．計算性能の変動の頻度が少ないほど計算ホストの性能低下の偏りが大きくなり，静的スケジューリングの性能は大きく低下する．一方提案手法では，再スケジューリングを行うことで実行効率の改善がされたと考えられる。

表 3 の評価結果から，条件 A は性能変動の頻度が高いほど静的スケジューリングの実行回数は多いが，条件 B や条件 C の実行回数には違いが無かった。これは，条件 A はワークフローのタスク数に依存せず，条件 B や条件 C は計算ホストの性能変動に依存しないためである．一方スケジューリング長の増加は図 4 の結果から約 5% に収まっている．従って，条件 B や条件 C は約 5% のワークフローの実行効率の低下で，再スケジューリングの実行回数を大幅に減らすことができるといえる．

また条件 B と条件 C の再スケジューリングの実行回数に約 10 倍の差があった．しかし，図 4 の結果からスケジューリング長が同じか良い結果であったため，条件 C は条件 B と比較して優れていると考えられる．

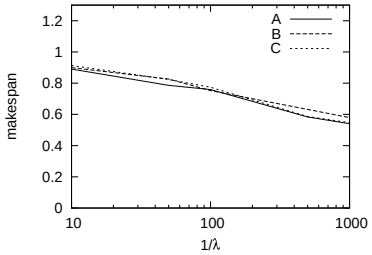

図 4 スケジューリング長 (性能変動)
Fig. 4 Makespans with Performance Change

表 3 静的スケジューリングの実行回数 (性能変動)
Table 3 The Number of Static Scheduling Execution with Performance Change

$1/\lambda$	10	50	100	500	1000
条件 A	18932	4011	2406	694	459
条件 B	1029	1023	1020	1025	1022
条件 C	162	148	164	152	148

5.3.2 CCR

CCR 値の違いによる各条件の効果の違いを明らかにするために，スケジューリング長と再スケジューリングの呼び出し回数による比較評価を行った．図 5 は各条件でスケジューリングを行った時のスケジューリング長を，再スケジューリングを行わない場合のスケジューリング長を 1 として正規化した値である．また各局所スケジューラが再スケジューリングを行った回数の和を表 4 に示す．本評価ではワークフローの実行前と実行中で計算ホストの計算性能に違いを持たせるために，疑似タスクは半数のホストで生成した．また，疑似タスクはシミュレーション開始時に生成し，ワークフローの実行中に疑似タスクの生成は行わなかった。各疑似タスクは指数分布に従って消滅するようにした．

図 5 の評価結果から，全ての条件において CCR の値が低いほど高い効果が得られた．タスクの通信量が計算量より大きい場合，ワークフローの実行時間の内，タスク間のデータ通信時間が大部分を占める．従って，動的スケジューリングによってタスクの実行時間が短くなっても大きな効果が得られないと考えられる．

表 4 から，条件 B と条件 C で再スケジューリングの実行回数に約 10 倍の差があった．条件 B では独立型タスク群の各タスクが終了する毎に再スケジューリングが行われる．一方，条件 C では独立型タスク群の全てのタスクが終了した時点で一度だけ再スケジューリングが行われるので，大幅に呼び出し回数が減った．従って条件 C は条件 B より再スケジューリングの計算コストが小さくなると思われる．図 5 から条件 C は条件 B のスケジューリング長よりも良い結果が得られたので，性能変動の時と同様，条件 C は条件 B と比較して優

© 2010 Information Processing Society of Japan

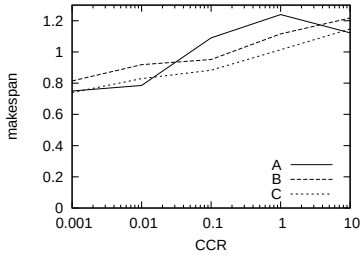

図 5 スケジューリング長 (CCR)
Fig. 5 Makespans with CCR

表 4 静的スケジューリングの実行回数 (CCR)
Table 4 The Number of Static Scheduling Execution with CCR

CCR	0.001	0.01	0.1	1.0	10.0
条件 A	141	142	141	138	147
条件 B	1022	1022	1017	1025	1023
条件 C	164	147	161	162	152

れていると考えられる．

6. おわりに

本稿では，計算ホストの性能変動やワークフローの実行状態を条件とし静的スケジューリングを実行する手法を提案した．また，静的スケジューリングを実行する条件について３種類提案した．抽象シミュレーションの結果から，計算ホストの性能変動が頻繁に発生する場合，条件 C は静的スケジューリング実行回数が大幅に少ないにもかかわらず，ワークフローの実行効率は条件 B と比較して同等か良い結果になり，条件 A と比較して 5% 程度の低下に留まった．

今後は，ワークフローの実行効率を高めつつ再スケジューリングの実行回数減らす条件の開発をしていく．そのために，本手法は階層型スケジューリング手法の局所スケジューラのみを改良したが，大域スケジューラに対しても動的手法の開発する必要がある．また評価においては，ネットワークの挙動をより現実的なモデルにするなど，精度を高めたシミュレーション評価を行うと共に，広域分散環境での実評価を行う必要がある．

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 21013025, 「タスクと実行環境の高精度モデルに基づくスケーラブルなタスクスケジューリング技術」) による．

参考文献

1) 松本真樹，片野聡，佐々木敬泰，大野和彦，近藤利夫，中島浩：ヘテロ型大規模並列環境の階層型タスクスケジューリングの提案と評価，情報処理学会論文誌:プログラミング， Vol.2, No.1, pp.1–17 (2008).
2) 松本真樹，片野聡，佐々木敬泰，大野和彦，近藤利夫，中島浩：非均質環境における適応型スケジューリング手法の提案と評価，電子情報通信学会論文誌， No.6, pp.693–704 (2010).
3) 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.SIG 12 (ACS 11), pp.181–193 (2005).
4) 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).
5) 阪口祐輔，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ユーザーレベル機能拡張機構，情報処理学会論文誌， Vol.47, No.SIG 12(ACS 15), pp.296–307 (2006).
6) S.F., A., W., G., W., M., E.W., M. and D.J., L.: Basic local alignment search tool, *Journal of Molecular Biolpgy*, Vol.215, pp.403–410 (1990).
7) J., L.D. and W.R., P.: Rapid and sensitive protein similarity searches., *Science*, Vol.227, pp.1435–1441 (1985).
8) J.D., T., D.G., H. and T.J., G.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, *Nucleic Acids Reseach*, Vol.22, pp.4673–4680 (1994).
9) http://setiathome.berkeley.edu/: SETI@home.
10) H.Topcuoglu, S.Hariri and Wu, M.-Y.: A high-performance mapping algorithm for heterogeneous computing system, *IPPS/SPDP Workshop on Heterogeneous Computing*, pp.3–14 (1999).
11) A., D. and R, O.: LDBS:a duplication based scheduling algorithm for heterogeneous computing systems, *Proc. Int'l Conf. Par. Proc.*, pp.352–359 (2002).
12) R.Armstrong, D.Hensgen and T.Kidd: The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions., *7th IEEE Heterogenous Computing Workshop*, pp.87–97 (1998).
13) R. D. Blumofe and C.E.Leiserson: Scheduling Multithreaded Computations by Work Stealing., *35th Annual Symposium on Foundations of Computer Science (FOCS'94)*, pp.34–43 (2001).
14) Yu, Z. and Shi, W.: An adaptive rescheduling strategy for grid workflow applications., *Parallel and Distributed Processing Symposium,International*, Vol.0, pp.1–8 (2007).
15) Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A. and Kennedy, K.: Task scheduling strategies for workflow-based applications in grids., *Fifth IEEE*

© 2010 Information Processing Society of Japan

Vol.2010-HPC-126 No.6
2010/8/3

International Symposium on Cluster Computing and the Grid, Vol.2, pp.759–767 (2005).

16) Weiner, J., S., Kollman, A., P., Nguyan, T., D. and A., C.D.: Rapid and sensitive protein similarity searches., *Science*, Vol.227, pp.1435–1441 (1985).

© 2010 Information Processing Society of Japan