グラフ分割による広域分散並列ワークフローの効率的な実行

田中昌宏† 建部修見†

地理的に離れた複数組織の計算機資源を連携して大規模データの解析を行う e-サイエンスにおいて，広域分散並列ワークフローを実行する際の効率的なファイルシステムへのアクセスが課題の 1 つである．本研究では，ワークフローのタスクをどの拠点に割り当てて実行するかにより，性能低下の要因となる別拠点へのファイルアクセスを減らすことができることを示した上で，最適なタスク割り当てを達成するため，ワークフローのグラフ分割に基づく手法について提案する．この提案手法に基づき，実際に複数拠点において並列分散ワークフローを実行した結果，別拠点へのファイルアクセスが全ファイルアクセスのうちの約 5-20%に抑えられ，それにより実行時間が 16-35%短縮されることを実証した．

Efficient Execution of Large-area Distributed Parallel Workflows by Graph Partitioning

Masahiro Tanaka† and Osamu Tatebe†

Efficient file access is one of issues for executing wide-area distributed parallel workflows in e-Science conducted in research collaboration with multiple distant organizations. This paper discusses how the allocation of workflow tasks to execution sites affects distant file accesses which degrade the performance of workflow. We propose a method based on graph partitioning to achieve optimum task allocation. The performance of distributed parallel workflow based on the proposed method is evaluated. The result shows that file access to different sites is reduced to 5-20% of the total sizes of file accesses in the workflow, and demonstrate that this method reduces elapsed time of workflows.

1. はじめに

天文学の分野では，SDSS[1] や 2MASS[2] など，サーベイに特化した観測により，大量の均質な観測データが得られるようになった．そのような観測データを利用して行う研究は，新しい研究分野の一つとして認知されてきている．また，観測装置の進化により，アーカイブされる観測データの量も増加している．すばる望遠鏡の主焦点カメラ Suprime-Cam は 1 年に約 1.5 TB でデータを生み出しているが，開発中の Hyper Suprime-Cam に置き換えられれば，一度に約 10 倍の広さの撮像が可能となり，データ発生量も約 10 倍となる．こうした大量のデータに対してデータ解析を行うには，並列分散処理が必要であり，特にスケーラビリティが高いファイルシステムが鍵になる．NFS のようなサーバ集中型のファイルシステムは，アクセス集中による性能劣化が問題である．そのため，並列分散処理では Lustre[3], PVFS[4], Gfarm[5] などの並列ファイルシステムが用いられる．

処理するデータが大規模になり，1 拠点では計算機資源が不足する場合は，複数拠点の計算機資源を利用したい場合もある．このような複数拠点の情報基盤を連携してインターネットを通じて連携させることにより，新たな研究分野を創出することを目指した e-サイエンス基盤の研究開発が行われている．e-サイエンス基盤の 1 つである広域分散ファイルシステム Gfarm は，複数拠点による広域データ共有・データ解析に対応したファイルシステムである．

複数拠点による大規模なデータ解析を行う場合，ストレージの性能に加えて，ワークフローを効率的に実行するための基盤が必要である．ワークフローを構成するタスクの依存関係に基づき，分散計算機に適切にタスクを割り当てるといったスケジューリングの機能が必要である．ワークフローの処理系として，グリッドについては TeraGrid[6] で用いられる Pegasus[7] などがある．それらの処理系では，ワークフローを DAG (Directed Acyclic Graph, 有向非循環グラフ) により

† 筑波大学
University of Tsukuba

表現し，その DAG に基づいてスケジューリングを行うことが多い．グリッドが導入されていないクラスタに対してワークフローを容易に実行できるツールとして，GXP make[8] がある．

一方，Gfarm ファイルシステムの特長を生かしてワークフローを実行するためには，ワークフローのタスクをどのノードで実行するかというスケジューリングに対する工夫が必要となる．Gfarm の特徴として，ストレージの実体を持つファイルシステムノードが計算ノードを兼ねることができる．そのため，ファイルが置かれたノード，またはネットワーク的に近いノードで処理を行うことにより，ファイルアクセスの効率を高めることが可能である．しかし，ローカリティを考慮したタスク実行の機能は Gfarm バージョン 2 にはない．この Gfarm の特徴を生かしたワークフロー実行システムはこれまで存在しなかった．

そこで我々は，Ruby 版 make である Rake を拡張した Pwrake[9] を開発し，Gfarm におけるローカリティを考慮したワークフロー実行を可能にした．この先行研究により，複数拠点のクラスタを用いてワークフローを実行した場合の問題点が判明した．その問題とは，別の拠点のストレージに置かれたファイルへのアクセスが多い場合，拠点間のファイル転送により，ワークフローの実行性能が悪くなることである．

そこで，本研究では，拠点間のファイル転送を最小化するための手法として，タスクを実行するサイトを決める際に，ワークフローのタスクの依存関係のグラフに対してグラフ分割を適用する手法を提案する．その提案手法を天文画像合成ソフトウェア Montage のワークフローに適用し，実行性能や拠点をまたぐファイルアクセスの削減について評価する．

本稿の構成は以下の通りである．2 節で関連研究について述べ，3 節で広域分散ワークフローについて考察する．4 節で実装について述べ，評価対象のワークフローについて 5 節で述べる．6 節で性能評価を行い，7 節でまとめと今後の課題について述べる．

2. 関 連 研 究

タスクをグループ化する問題は，並列プロセッサのスケジューリングに関して，古くから研究されている[10]．グリッドにおいては，Pegasus にはワークフローをクラスタリングする機能がある[11]．これは並列タスクを部分的にまとめて抽象化ワークフローとするものであり，本研究のようにファイルアクセスを効率化するためのものではない．

データに空間的な座標の情報が付随していれば，その座標に基づいてグループ化する手法が考えられる．天文データ解析のワークフロー関しては，入力画像の座標に基づいてワークフローをクラスター化する手法[12] が行われている．その論文ではファイル転送量についても議論されている．しかし，この手法では座標を明示的に指定する必要があり，自動的にクラスター化を行うものではない．本研究で提案する手法は，ワークフローのグラフに基づく手法である．ワークフローの実行を目的とした座標情報の取得を必要とせず，より一般的なワークフローにも適用することを目指す．

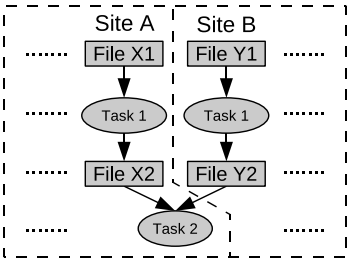

図 1 拠点間のファイルアクセスが発生するワークフロー．Task 1 が異なる拠点に割り当てられると，File X2, Y2 が異なるサイトに書き込まれ，Task 2 で別拠点へのアクセスが生じる．

3. 広域分散ワークフローについての考察

3.1 タスクごとのローカリティ考慮の問題

先行研究[9] では，ワークフロー実行の際にタスクを実行するノードを決める手法として，入力ファイルのローカリティのみを基準とする方法を採った．本節では，同様な手法を複数拠点で行った場合に起こる問題点について述べる．

前提条件として，入力ファイルは各拠点のストレージに複製されているものとする．利用頻度の高い公開天文データの場合は，そのような運用が行われることが予想される．

次に，**図 1** のようなワークフローを複数拠点のクラスタで実行する場合を考える．1 段目の Task 1 の入力ファイル X1 と Y1 はどの拠点にも複製されているから，Task 1 をどの計算ノードで実行しても，拠点内のファイルアクセスとなる．Task 1 が出力した中間ファイル X2, Y2 は，Gfarm ファイルシステムの場合，空き容量などに問題がなければ，プロセスが実行された計算ノードのストレージに書き込まれる．したがって，処理を実行したノードと，出力ファイルが書き込まれたノードは同じとみなすことができる．

次の Task 2 は，Task 1 が出力した中間ファイル 2 つを入力とする．その 2 つの中間ファイル X2, Y2 が，

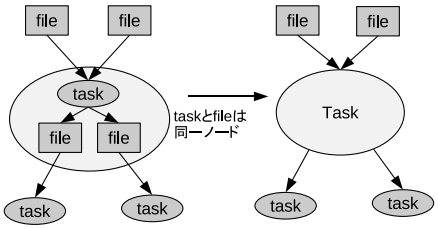

図 2 ワークフローをグラフに表現

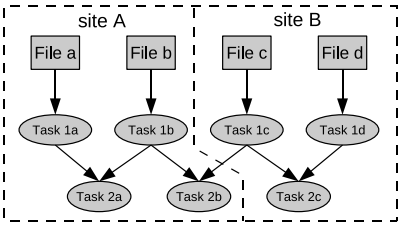

図 3 ワークフローのグラフ分割の例

もし同じ拠点にあるならば，その拠点の計算ノードで Task 2 を実行すれば，Task 2 のファイルアクセスは両方とも拠点内のアクセスとなる．一方，中間ファイル X2, Y2 が異なる拠点に置かれた場合，Task 2 をどちらの拠点へ発行したとしても，片方のファイルへのアクセスが効率の悪い別拠点へのアクセスとなる．このように，中間ファイル X2, Y2 が同じ拠点に書き込まれたかどうかによって，Task 2 の効率が左右される．その中間ファイル X2, Y2 が書き込まれた場所は，前述のとおり，Task 1 が実行された場所と同じである．したがって，最初の Task 1 がどの拠点で実行されたかによって，後で実行される Task 2 の効率が左右される，というわけである．

この例が示すように，複数拠点でワークフローを効率的に実行するためには，タスク単位でローカリティを考慮するだけでは不十分であり，後で実行されるタスクの依存関係も含めたワークフロー全体の情報に基づく最適化が必要である．

3.2 ワークフローのグラフ表記

前節の図 1 ではワークフローをグラフで表す際にタスクと入出力ファイルを分けて描いた．本稿では，以下に述べる理由から，ワークフローのグラフを**図 2** のようにタスクと出力ファイルをまとめて表記する．

まず，効率的なストレージアクセスを追求する場合，タスクを実行したローカルのストレージに書き込む方法が考えられ，本稿でもそのような状況を想定する．実際，Gfarm ファイルシステムでは，高い書き込み性能を得るため，空き容量が必要以上あるなどの条件を満たしていれば，プロセスが実行された計算ノードのストレージに出力ファイルが書き込まれる．そこで，未実行のタスクについては，実行ノードのストレージに出力ファイルが書き込まれると仮定する．タスクと出力ファイルが同じ計算ノードにあるならば，タスクの拠点配置という観点からは，図 2 に示したように，それらを 1 つのグラフノードで表してよい．そこで，これ以降は，ワークフローをグラフに表記する場合には，簡略化のため，タスクのみを頂点で描き，その依存関係についてエッジで結ぶという表記方法を採用する．

3.3 ワークフローのグラフ分割

3.1 節で，ワークフローの効率的な実行のためには，後の方で実行されるタスクの依存関係を含めたワークフロー全体の情報に基づいて，タスクを実行する拠点を決める必要があることについて述べた．本節では，その決定方法がどのような問題に帰着できるかについて考察する．

ここで**図 3** のようなワークフローを 2 つの拠点で実行する場合について考える．最初の 4 つの Task 1a-1d を 2 つの拠点に割り振る方法は $2^4 = 16$ 通り存在する．しかしどのように割り振っても，Task 2a-2c において拠点間のアクセスが発生する．その中で，拠点間アクセスが少ないケースは，図のように Task 1a, 1b と Task 1c, 1d をそれぞれグループ化するケースである．それにより，拠点間のファイルアクセスは，Task 2b の片方の入力エッジの 1 回のみとなる．もし Taks 1a-1d を交互に違う拠点に割り振ると，Task 2a-2c の 3 つのタスクすべてにおいて片方の入力エッジが拠点間のファイルアクセスとなるため，ワークフロー全体で拠点間ファイルアクセスが 3 回となる．これは最適な場合に比べて 2 回多い．

このグループ化問題は，グラフ分割問題と同等である．グラフ分割問題は，頂点を複数のグループに分ける際に，エッジカットコストが小さい，すなわち，異なるグループに属する頂点を結ぶエッジができるだけ少なくなるように頂点をグループに割り当てる問題である．この問題は NP 完全であることが知られている．グラフ分割問題の解法については，多くの研究が行われており，高速に解くオープンソースの実装として METIS[13] がある．MEITS を利用することにより，オーバーヘッドが小さい実装が可能である．

3.4 グラフ分割が適用できない場合

ここで，タスクを複数の拠点に配分する目的として，グラフ分割が有効でないケースについて考察する．

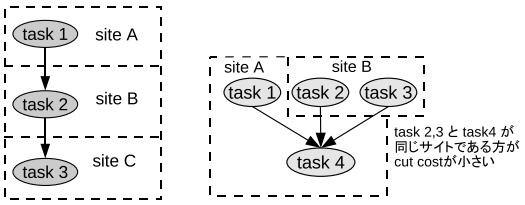

図 4 グラフ分割が有効でない例：逐次ワークフロー (左)，集約ワークフロー (右)

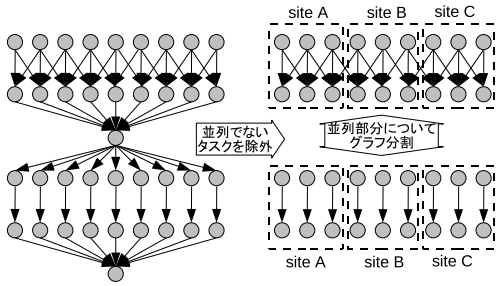

図 5 提案手法による，タスク実行サイトの決定

3.4.1 逐次ワークフロー

図 4 左のように，task 1, 2, 3 の順に逐次実行するような逐次ワークフローの場合，3 つのグループにグラフ分割を行うと，頂点がそれぞれ異なるグループに分けられる．しかしこれでは拠点間のファイル転送が起こり，非効率である．このような場合には 1 拠点で実行する方が効率が良い．このように逐次ワークフローから成るワークフローに対しては，グラフ分割は有効ではない．

3.4.2 集約ワークフロー

図 4 右のように，並列タスクの出力ファイルすべてを入力ファイルとするような集約ワークフローの場合，エッジが集まるタスク (task 4) と同じグループに属する並列タスクが多いほどエッジカットのコストが低くなる．そのため，集約ワークフローの存在は，並列タスクを別々のグループに分配するのを妨げる方向に作用する．この状況は，1 つのファイルが並列タスクの入力ファイルとなる場合にもあてはまる．

逐次・集約ワークフローはたいていのワークフローに含まれているが，そうしたワークフローのグラフに対してそのままグラフ分割を適用すると，意図しない結果となる．

3.5 提 案 手 法

以上のように，タスクを拠点に配分する目的としては，グラフ分割があらゆるワークフローに対して有効というわけではない．もし全てのワークフローに適用するならば，グラフ分割を発展させた新たなアルゴリズムが必要となる．しかし，本研究では，実用性の観点から，既存のグラフ分割ソルバの実装を利用するため，ワークフローグラフを変形してグラフ分割問題に帰着する手法を提案する．その手法とは，

Step 1 ワークフローのグラフから，逐次・集約ワークフローのエッジを除き，並列タスクのみから成るグラフにする．

Step 2 Step 1 で得た並列タスクのグラフそれぞれに対して，グラフ分割を行う．

本提案手法の具体例を**図 5** に示す．図 5 の例では，集約ワークフローを除くことにより，前半と後半の 2 つのグラフに分断される．前半と後半それぞれのグラフについて，グラフ分割を適用することにより，タスクを実行する拠点を決定する．

図 5 の前半のグラフは，エッジによってひとつながりになっている．このグラフを分割すると，いずれかのエッジがカットされることになる．どのエッジを切ればよいかはグラフ分割問題を適用でき，それにより最適なタスクの拠点配分が実現する．後半のグラフのように，エッジカットを行わなくてもグラフ分割できる場合は，任意の拠点にタスクを分配すればよい．

グラフ分割では，エッジや頂点に対して重みをつけることができるが，ここでは重みは等しいとする．ファイルサイズに応じてエッジに重みを付ける手法も考えられるが，それには中間ファイルのサイズを得る手法が別途必要となる．本稿ではそれは対象とせず，今後の課題とする．

4. 実　　装

本研究で使用するワークフロー実行ツール Pwrake は 9) で述べ，ここではその概要について述べる．Pwrake は，Ruby 版の make ツールである Rake に対して，クラスタにおける分散並列処理の機能を拡張したワークフロー実行ツールである．Rake においてタスクを記述する文法は Ruby そのものであり，これは言語内 DSL (Domain Specific Language) と呼ばれる特徴である．これにより Rake では Ruby の機能を使った高度なタスクの記述が可能である．

Pwrake で拡張した部分は，SSH による遠隔接続，スレッドプールによる並列実行の他，Gfarm 用のマウント機能，ローカリティ機能などである．Ruby は機能拡張が容易であるため，タスクのスケジューリング機能も容易に追加できる．Rake ではタスクの記述を Ruby スクリプトとして実行し，Rake::Task クラスまたはそのサブクラスのオブジェクトを作成する．

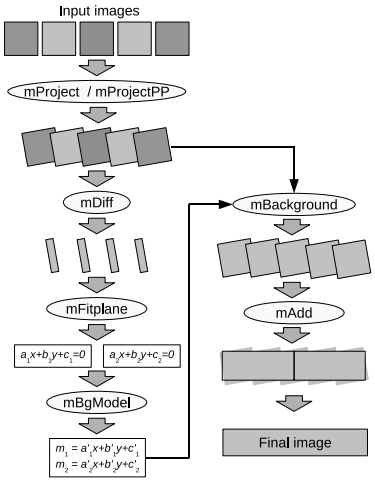

図 6 Montage ワークフロー

そのオブジェクトがタスクの依存関係などの情報を保持しており，その情報を解析することにより，ワークフローのグラフを取得できる．さらに提案手法にしたがって逐次・集約ワークフローを除いたグラフを作成し，METIS によりグラフ分割を行う．評価目的として，今回は METIS のコマンド インタフェース pmetis を用いた．pmetis に与えることができるパラメータは，頂点とエッジ，およびそれらの重みである．METIS のコマンド インタフェースからは，各グループに属する頂点の割合のパラメータを指定できず（C の API からは可能），グラフの頂点は各グループ均等になるように分割される．

5. 評価対象のワークフロー

5.1 Montage

本提案手法を評価する対象のワークフローは，天文画像の合成（モザイキング）を行うソフトウェア Montage[14] である．Montage は，多くのワークフロー研究において実例として取り上げられており，ベンチマーク的な側面もある．天文観測において画像を撮像する場合，1 つのカメラが 1 度に撮影できる空の大きさは固定であり，それより広い領域を撮影する場合には複数回のショットを行う．こうして撮影された複数の画像を合成し，1 枚の画像にするためのソフトウェアの 1 つが Montage である．

Montage は処理ごとにプログラムが分かれており，最小単位は画像 1 枚に対する処理である．そのため画像の枚数分の並列化が可能である．**図 6** にこのワークフローを模式的に示す．Montage のワークフローでは，まず最初に mProjectPP というプログラムによって入力画像を出力画像の座標系へ投影する．次に明るさの補正を行う．天文画像は，撮影条件によって，星がない部分の空の明るさが変わるため，明るさの変化をスムーズにする必要がある．それにはまず mDiff というプログラムによって，画像間で重なった部分の「差」の画像を抽出する．次に mFitplane によって差の値を 1 次成分でフィットする．その結果に基づき，画像全体でスムーズにつながるように，mBgModel で各々の画像について補正パラメータを計算する．mBackground ではその補正パラメータを用いてそれぞれの画像について明るさ補正を施す．こうして座標系と明るさが変換された画像を，次のように 2 段階に分けてつなぎ合わせる．まず，全体の領域を等分割した部分領域それぞれについて，その部分領域に含まれる画像を mAdd でつなぎ合わせる．次につなぎ合わされた部分画像を mShrink というプログラムによって縮小する．こうして得られた部分画像すべてを最後にもう一度 mAdd によってつなぎ合わせることで最終的な画像が得られる．

表 1 2MASS 画像データセットの諸元

画像ファイル 1 枚	
ファイルサイズ	2.1 MB or 1.7 MB
画素数	512 × (1024 or 830)
画像領域	8.5′ × (17.1′ or 13.8′)
データセット 1	
ファイル数	28
全ファイルサイズ	57 MB
合成画像領域	38′ × 37′
データセット 2	
ファイル数	309
全ファイルサイズ	639 MB
合成画像領域	167′ × 167′

5.2 入力データ

Montage ワークフローに対する入力データとして，2MASS の画像データを用いた．本稿では，**表 1** に示したファイル数の異なる 2 つのデータセットを用いる．後で述べる性能評価では，データセット 2 を用いる．

5.3 Montage ワークフローのグラフ分割

Montage ワークフローに対して，提案手法の Step 1 に基づき逐次・集約部分のエッジを除くと，並列タスクに関しては，最初の mProjectPP から最後の mAdd の手前の mShrink まで，エッジで結ばれたグラフとなる．データセット 1 を入力ファイルとするワークフローについて，Step 1 を適用した後のワークフローグラフについて図示したものが**図 7** である．タスクの種類は，上から順に mProjectPP, mDiff, mFitplane, mBackground, mAdd, mShrink である．図 7 では，グラフ分割によって頂点を 4 つのグループに分けた結

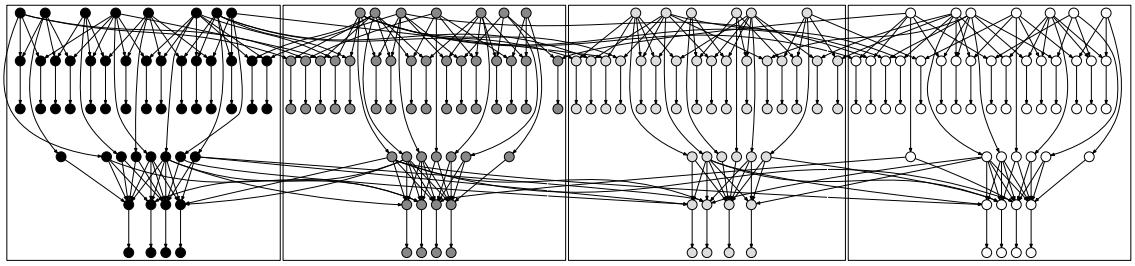

図 7 Montage ワークフローのグラフ分割

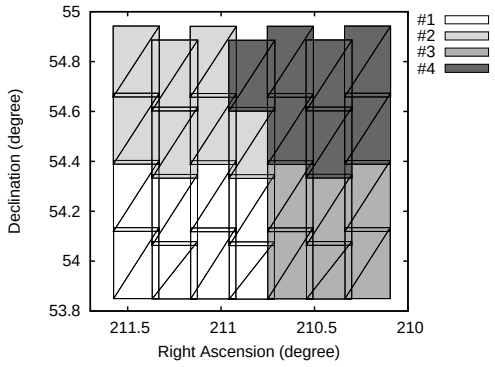

図 8 データセット 1 についてグループ化した画像ファイルの空間分布．赤道座標系での画像の範囲を示す．

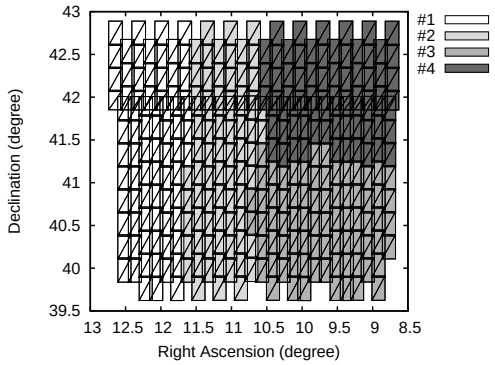

図 9 データセット 2 についてグループ化した画像ファイルの空間分布．赤道座標系での画像の範囲を示す．

果を四角の枠で示す．このグラフのエッジの数は，全部で 315 であり，そのうち 40 のエッジがカットエッジ，すなわち異なるグループに属する頂点を結ぶエッジである．また，データセット 2 について同様にワークフローからグラフを構築すると，エッジの数が 3615 になり，4 つのグループに分割すると，155 のエッジがカットエッジとなる．

5.4 画像ファイルの空間分布

ここで，提案手法による Montage ワークフローの分割結果について，画像の空間分布から検証する．データセット 1 の入力画像ファイルについて，座標を赤道座標系で示した図が**図 8** である．それぞれの画像に対して最初に mProjectPP タスクが処理を行う．本提案手法によるグラフ分割の結果，mProjectPP タスクに割り当てられたグループを，図 8 の対応する画像内の色の濃さによって示した．この図から，画像の座標に関しても，空間的に近い画像同士でグループ化されることがわかる．

データセット 2 の画像ファイルについて同様に示したものが**図 9** である．図 9 の結果は十字に区切られてはおらず，縦で区切られた部分がある．これは，画像 1 枚が縦長であり，横方向より縦方向のファイルの数が少ないことが影響している．このように，ワークフローのグラフ分割に基づく本提案手法により，空間的な情報を明示的に与えなくても，適切にタスクをグループ化できることが確認できる．

2 節で述べた，座標によってワークフローのクラスタリングを行う手法[12] では，まずグループの座標範囲を設定し，グループ分けの際にファイルから座標を読む必要がある．本提案のワークフローのグラフ分割による手法では，そうした座標情報を必要としない点で優れている．

6. 性能評価

提案手法によって拠点間のファイルアクセスがどのように削減されるかについて調べるため，実際にワークフローを実行し，ファイルアクセスサイズと実行時間について測定を行った．

6.1 測定環境

測定に使用した環境は，InTrigger プラットフォーム[15] である．InTrigger は全国 17 拠点に配置されたクラスタにより構成される．本測定で使用した拠点は，千葉，早稲田，筑波，広島の各拠点である．それぞれ拠点の計算ノードの仕様を**表 2** に示す．測定は拠点数が 1 から 4 までの各場合について行った．それぞれの拠点数の場合にどの拠点を使用したかを**表 3** に示

表 2 測定環境

拠点	千葉	筑波	広島	早稲田
CPU	Xeon E5410 (2.33GHz)			Core2 6400 (2.13GHz)
使用コア数/ノード	4	4	4	2
使用ノード数 (最大)	12	6	6	12
主記憶容量 (GB)	32	32	16	4
千葉からの RTT (ms)		9.4	24.6	4.4

表 3 測定に使用した拠点の内訳

1 拠点	千葉
2 拠点	千葉，早稲田
3 拠点	千葉，早稲田，筑波
4 拠点	千葉，早稲田，筑波，広島

表 4 拠点間ファイルアクセス量の削減

拠点数, コア数	スケジューリングなし	提案手法	削減データ量
4, 96	7.3 GB (60%)	2.3 GB (19%)	4.9 GB (41%)
3, 72	6.6 GB (54%)	1.7 GB (14%)	4.9 GB (40%)
3, 48	6.8 GB (56%)	1.4 GB (12%)	5.3 GB (44%)
2, 48	4.6 GB (38%)	1.3 GB (10%)	3.3 GB (27%)
2, 24	4.8 GB (39%)	0.6 GB (5%)	4.2 GB (34%)

百分率は，ワークフローにおける全ファイルアクセスに対する割合．

表 5 ワークフロー経過時間

拠点数, コア数	スケジューリングなし	提案手法	削減 実行時間	削減 割合
4, 96	365 秒	276 秒	89 秒	24%
3, 72	256 秒	188 秒	68 秒	30%
3, 48	290 秒	187 秒	103 秒	35%
2, 48	189 秒	160 秒	29 秒	16%
2, 24	292 秒	234 秒	58 秒	20%

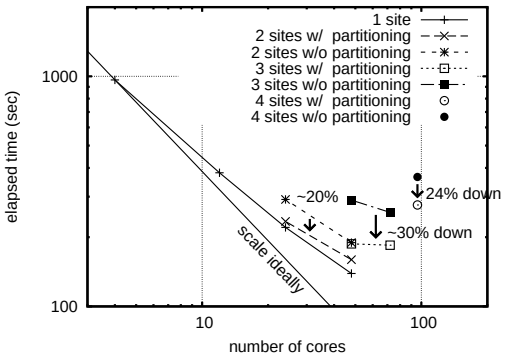

図 10 ワークフロー経過時間

す．各拠点で同時に起動する並列プロセス数は同じとした．ただし，早稲田拠点のみ 1 ノードあたりのコア数は 2 であるのに対し，他の拠点では 1 ノードあたりのコア数は 4 以上である．そこで，早稲田拠点では 1 ノードあたり同時に 2 プロセス起動し，他の拠点では同時に 4 プロセスを起動した．そのため，早稲田拠点では他の拠点の倍の数のノードを使用した．

ファイルシステムには Gfarm を用いた．Pwrake の機能により，ワークフローの開始時に，使用コア数と同じ数の gfarm2fs によるマウントポイントが用意される．タスクはそれぞれのマウントポイントに移動して実行される．Gfarm のメタデータサーバは筑波拠点に設置されている．

入力ファイルは表 1 のデータセット 2 である．入力ファイルについては，各拠点がそれぞれデータセットを 1 つずつ持つように複製配置した．各拠点の中では，全てのファイルを 1 つのノードに集めて置くのではなく，file1 は node1，file2 は node2 というように，拠点内のノードで分担するようにファイルを配置した．

6.2 結果と評価

本測定では，スケジューリングを行わない場合，すなわち，タスクを無作為に各拠点のノードに割り当てた場合と，提案手法を適用した場合についてワークフローを実行し，別拠点へのファイルにアクセスしたデータ量，および実行にかかった時間について集計し，手法の差による性能の違いを調べる．

6.2.1 拠点間ファイルアクセスの削減

ワークフローを複数拠点で実行した際に，拠点間でアクセスされたファイルサイズの合計を**表 4** に示す．スケジューリングなしでワークフローを実行した場合，全ファイルアクセスに対して，拠点間アクセスの割合は約 40-60%であった．提案手法の導入により，この割合は約 5-20%にまで減少した．拠点間ファイルアクセスの削減は，全ファイルアクセスに対する割合では約 30-40%になり，削減データ量は約 3-4 GB である．このように，提案手法の導入によって実際に拠点間アクセスが減少することを確認できた．

6.2.2 実 行 時 間

複数拠点で実行したワークフローの経過時間を測定した結果を**表 5** に示す．拠点間ファイルアクセスの削減に対応して，実行時間も約 30-100 秒短縮され，割合にして 16-35%減少した．なお，グラフ分割に用いたプログラム pmetis の経過時間は約 0.3 秒である．

ワークフローのスケーラビリティを調べるため，使用コア数に対する，表 5 のワークフロー経過時間の両対数プロットを**図 10** に示す．この図では，表 5 の複数拠点での結果に加えて，理想的な場合として千葉 1 拠点のみの結果についてもプロットした．1 拠点での結果は，コア数の増加にしたがって実行時間が減少することを示している．

一方，2 拠点での結果は，スケジューリングを行わない場合に対して，提案手法により実行時間が 16-20%減少し，1 拠点での結果に近づいている．2 拠点の場合でも，本提案手法により，1 拠点に近いスケーラビリティが得られることがわかる．

さらに，3, 4 拠点を用いた場合でも，提案手法の適用により，ワークフローの実行時間は 24-35%減少し，本提案手法の有効性を示している．ただし，提案手法を適用しても，1 拠点 48 コアと比較すると，3 拠点 72 コアおよび 4 拠点 96 コアでは，コア数の増加にもかかわらず，実行時間は増加している．複数拠点に対してスケーラビリティを得るためには，提案手法の他に課題があると考えられる．とはいえ，グラフ分割に基づく本提案手法が，複数拠点の計算機を用いた広域分散ワークフローを実行する際に，必要かつ有効な手法の一つであるといえる．

7. まとめと今後の課題

本研究では，e-サイエンス基盤 Gfarm ファイルシステムによる複数拠点の計算機資源を利用した広域分散並列ワークフローを効率的に実行するため，グラフ分割によりタスクを各拠点に配分する手法について提案した．本手法に基づいて実際に複数拠点を用いてワークフローを実行した結果，拠点をまたぐファイルアクセスが全ファイルアクセスの約 5-20% に抑えられ，それによりワークフローの実行時間が 16-35% 短縮されることを実証した．ただし複数拠点でのワークフローの実行においてスケーラビリティを得るためには，さらに別の手法が必要である．

その他に考えられる今後の課題を次に挙げる．

(1) 拠点だけでなく，計算ノードごとのグループ化も考慮した多段グループ化．
(2) ファイルがすでにある特定の拠点に存在する場合を考慮して，特定の頂点のみグループに割り当てられた状態でグラフ分割を行う手法．
(3) ファイルサイズによる重みの考慮．
(4) 複数の種類のワークフローに対する検証．

謝辞 本研究は，文部科学省の科学技術試験研究委託事業による委託業務：次世代ＩＴ基盤構築のための研究開発「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」における研究課題「研究コミュニティ形成のための資源連携技術に関する研究」（データ共有技術に関する研究）の支援を受けました．また本研究は，科学技術研究費特定領域 「情報爆発時代に向けた新しい IT 基盤技術の研究」において構築された研究用プラットフォーム InTrigger を利用しました．謹んで感謝の意を表します．

参 考 文 献

1) SDSS: http://www.sdss.org/.
2) Skrutskie, M. F., Cutri, R. M., Stiening, R., Weinberg, M. D. et al.: The Two Micron All Sky Survey (2MASS), *Astronomical Journal*, Vol. 131, pp. 1163–1183 (2006).
3) Lustre: http://www.lustre.org/.
4) PVFS: http://www.pvfs.org/.
5) Gfarm: http://datafarm.apgrid.org/.
6) TeraGrid: http://www.teragrid.org/.
7) Deelman, E., Singh, G., Su, M.-H., Blythe, J. et al.: Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems, *Scientific Programming Journal*, Vol. 13, No. 3, pp. 219–237 (2005).
8) Taura, K.: Grid Explorer : A Tool for Discovering, Selecting, and Using Distributed Resources Efficiently, 情報処理学会研究報告 2004-HPC-99, pp. 235–240 (2004).
9) Tanaka, M. and Tatebe, O.: Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing, *19th ACM International Symposium on High Performance Distributed Systems (HPDC 2010)* (2010). accepted.
10) Sarkar, V.: *Partitioning and Scheduling Parallel Programs for Multiprocessors*, MIT Press, Cambridge, MA, USA (1989).
11) Deelman, E., Blythe, J., Gil, A., Kesselman, C., Mehta, G., Patil, S., hui Su, M., Vahi, K. and Livny, M.: Pegasus: Mapping scientific workflows onto the grid, pp. 11–20 (2004).
12) Meyer, L., Annis, J., Wilde, M., Mattoso, M. and Foster, I.: Planning spatial workflows to optimize grid performance, *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, New York, NY, USA, ACM, pp. 786–790 (2006).
13) METIS: http://www.cs.umn.edu/~metis.
14) Montage: http://montage.ipac.caltech.edu/.
15) 斎藤秀雄, 鴨志田良和, 澤井省吾, 弘中健ほか: InTrigger: 柔軟な構成変化を考慮した多拠点に渡る分散計算機環境, 情報処理学会研究報告 2007-HPC-111, pp. 237–242 (2007).