ワークフロー実行中のデータ移動を最小化するタスク配置方式

田 中 昌 宏[†1,†2] 建 部 修 見[†1,†2]

膨大な科学データの処理が伴うデータインテンシブサイエンスでは，高い並列 I/O 性能が求められる．そのためには，できるだけ処理タスクをデータが保管されているノードに割り当て，ノード間のデータ移動を少なくすることが必要である．ワークフローにおいてデータ移動を最小化するためには，ワークフロー全体の依存関係に基づき，タスクをどの計算ノードに配置するかを決める必要がある．最適なタスク配置方式をワークフローのグラフから自動的に求めるため，本研究では，ワークフローのグラフに対して，多制約グラフ分割を適用する手法を提案する．この提案手法を Montage ワークフローに適用した結果，ノード間のデータ移動がデータアクセス全体の 14% に削減され，これにより実行時間が 31% 短縮された．

New Method of Task Assignment for Mininum Data Movement in Workflow

Masahiro Tanaka[†1,†2] and Osamu Tatebe[†1,†2]

High-performance parallel I/O is one of key issues for data-intensive sciences which handles huge amount of science data. In order to achieve high-performance parallel I/O, it is necessary to exploit data locality, that is, closeness of computation and storage. As for the workflow execution, the overall dependency of workflow tasks determines which node to be selected for each task. In order to achieve an optimal assignment of tasks to compute nodes, we propose a new method of task assignment by applying multi-constraint graph partitioning to workflow graphs. 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 14% of the total sizes of file accesses in the workflow, and that the elapsed time is reduced by 31%.

1. はじめに

天文学，素粒子物理学，生命科学などの様々な科学分野において，観測装置の進化とともに，扱うデータ量は年々増加している．こうした大量のデータ処理が必要となるデータインテンシブサイエンスを推進するには，大量のデータに対する並列分散処理が必要である．優れた並列 I/O 性能を持つファイルシステムとして，Lustre[1)], PVFS[2)], Gfarm[3)] などの分散ファイルシステムが利用されている．加えて，高性能な並列処理が可能なワークフローシステムが必要である．高いワークフローの性能のためには，ワークフローを構成するタスクをどの計算ノードに割り当てるか，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，グリッド向けとして TeraGrid[4)] で用いられる Pegasus[5)] などがあり，クラスタ向けとして GXP Make[6)] などがある．ワークフローは，構成するタスクの依存関係に基づき，DAG (Directed Acyclic Graph, 有向非循環グラフ) により表現されることが多い．

Gfarm ファイルシステムには，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，という特徴がある．この特徴により，タスクを計算ノードに割り当てる際に，ファイルが保存されているノードへ割り当て，ファイルアクセス局所性を高めることにより，高い並列 I/O 性能を達成することが可能である．これを実現するには，ワークフローシステムの機能として，適切にタスクを計算ノードに割り当てる必要がある．一方で，ワークフローの記述する人はサイエンスユーザであり，システムに関する知識を持たないことが多く，そのような場合でも，適切なタスク配置を行うことが求められる．

本研究では，ノード間のデータ移動を最小化するタスク配置を自動的に決定する手法として，ワークフローのタスクの依存関係のグラフに対して “Multi-constraint Graph Partitioning”[7)]（多制約グラフ分割）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，実際の処理性能について測定し評価を行う．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 節でデータ移動最小化のための手法について考察する．4 節で提案手法について述べ，5 節で実装について述べる．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．

†1 筑波大学
University of Tsukuba
†2 独立行政法人科学技術振興機構 (JST)/CREST

2. 関 連 研 究

ワークフローをクラスタリングする研究として，Pegasus の例がある[8]．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，本研究のようにデータ移動の最小化を目的とするものではない．データの空間情報に基づいてグループ化する手法については，天文データ解析のワークフロー関して，入力画像の座標に基づきワークフローのクラスター化を行う研究[9] がある．そこ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与える必要があり，タスク配置の自動化を行うものではない．本研究では，ワークフローのグラフ情報のみ基づき，自動的にワークフローのデータ移動を最小化するようなタスク配置を行う手法が目的である．

3. データ移動最小化

3.1 アクセス局所性による性能向上

Gfarm のような分散ファイルシステムを用いる場合，I/O 性能の向上のための手段として，ファイルアクセスの局所性の利用が考えられている．Gfarm ファイルシステムでは，ファイルシステムノードにおいて FUSE 経由でファイルを書き込む場合，ファイルの保存先として，可能であれば書き込み元と同一のノードが選択される．Gfarm では，この仕組みによって，書き込みに関する局所性を実現している．一方，ファイルの読み込みに関して局所性を高めるためには，ファイルが保存されているノードにタスクを割り当てることが必要となる．そこで，ワークフローシステム Pwrake[10),11)] では，Gfarm ファイルシステムで実行する場合，タスクを実行するノードを選択するアルゴリズムとして，入力ファイルが保存されているノードを優先的に選択する，という仕組みが実装されている．これによって局所性を高めることにより，I/O 性能を向上させている．しかしこのタスク配置方式は，次に述べる Montage のような複雑なワークフローの場合は，十分ではないことがわかっている．

3.2 Montage ワークフロー

本研究で扱うワークフローは Montage[12)] という天文画像の合成（モザイキング）を行うソフトウェアである．天文学における撮像観測では，カメラが 1 度に撮影できる空の大きさは固定であるため，広い領域を観測する場合には複数回の撮像を行う．そのため，複数ショットの画像を合成し，1 枚の画像にする処理が必要になる．Montage は処理ごとにプログラムが分かれている．そこで，画像 1 枚を処理するプログラムについては，並列実行が可能である．Montage ワークフローの DAG を**図 1** に示す．

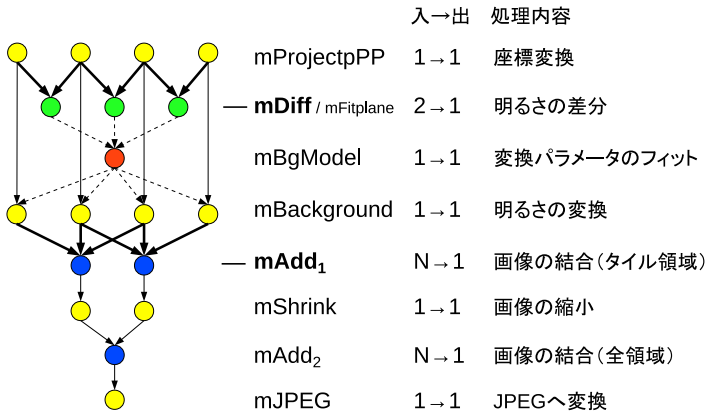

図 1 Monrage ワークフローの DAG．点線の矢印はパラメータファイルを経由する依存，その他は画像ファイルを経由する依存を示す．

ワークフローにおける I/O 性能という観点で見ると，入出力に関わるデータ量が問題になる．Montage ワークフローの場合，データサイズが大きいのは，画像ファイルである．Montage のタスクについて，画像ファイルへのアクセスパターンについて分類すると，(1) 入力ファイル 1 つのケース（mProjectPP, mBackground, mShrink），(2) 入力ファイルが 2 つ以上のケース（mDiff, mAdd）の 2 つのパターンに分類できる．データフローの観点で重要なのが，(2) のように入力ファイルが複数のパターンである．(2) の場合，入力ファイルがすべて同じノードに保存されていれば，そのノードでタスクを実行すれば，局所性が保たれる．ところが複数の入力ファイルが別々のノードに保存されている場合，タスクをどのノードに割り当てても，離れたノードへのアクセスとなり，データ移動が発生する．

そこで，(2) のパターンの場合に，できるだけ複数の入力ファイルが同じノードに保存されていることが，データ移動最小化のために必要である．ところが，Montage ワークフローを Gfarm ファイルシステムで実行した場合，mDiff の入力ファイルは mProjectPP の出力ファイルであるから，通常は mProjectPP が実行されたノードに保存される．このことは，mDiff において局所性を高めるためには，mProjectPP に遡ってタスク配置を考慮しなければならないということである．さらに mAdd_1（2 回の mAdd のうちの前の方）も同様に mBackground のタスク配置に依存しており，そのうえ mBackground は mProjectPP のタスク配置に依存している．このように，データの局所性を高めるには，ワークフロー全体

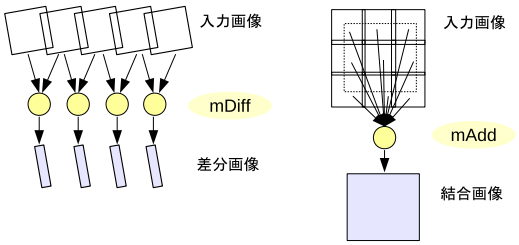

図 2 mDiff および mAdd の処理

の依存関係に基づいてタスク配置を決定する必要がある．

3.3 座標に基づくグルーピング

科学データには位置の情報を伴う場合がある．Montage で扱う画像ファイルも天球座標が伴う．このような位置情報を伴うデータの処理において，複数のデータに同時にアクセスする場合，それらの座標はお互いに近いことが多い．例えば，Montage ワークフローでは，mDiff プログラムでは，**図 2** に示すように，2 つの画像の重なった領域の明るさの差分を計算する．その 2 つの入力ファイルは，隣り合う 2 つの画像である．mAdd の処理は，指定された座標範囲の画像を結合して 1 枚の画像にする処理である．その N 個の入力ファイルは，指定された座標範囲に重なる画像ファイルである．このように，Montage ワークフローは，位置情報に基づいて構成されている．

このことから，ファイルアクセスの局所性を高めるために，ファイルの位置情報によってグルーピングする手法が考えられる．しかし，この手法は汎用的ではないことが問題である．ファイルから座標データを取り出す方法は，ファイルの種類ごとに異なる．位置情報の種類についても，1 次元から 3 次元，あるいは球面座標などさまざまである．位置情報はアプリケーション側の問題であるから，タスク配置についてもワークフローシステムを利用するユーザーの側が設計を行う必要が出てくる．

3.4 ワークフローのグラフに基づくグルーピング

ワークフロー実行時のデータ移動を最小化するタスク配置を自動的に行うために，著者らは，グラフ分割を用いる手法[13] を提案した．

グラフ分割問題は，グラフの頂点を複数のグループに分ける際に，異なるグループに属する頂点を結ぶ辺（エッジ）ができるだけ少なく（エッジカットコストが小さく）なるように，頂点をグループに均等に配分する問題である．この問題は NP 完全であることが知られている．グラフ分割問題のアルゴリズムについては，多くの研究が行われており，実装として METIS[14], SCOTCH[15] などがある ．

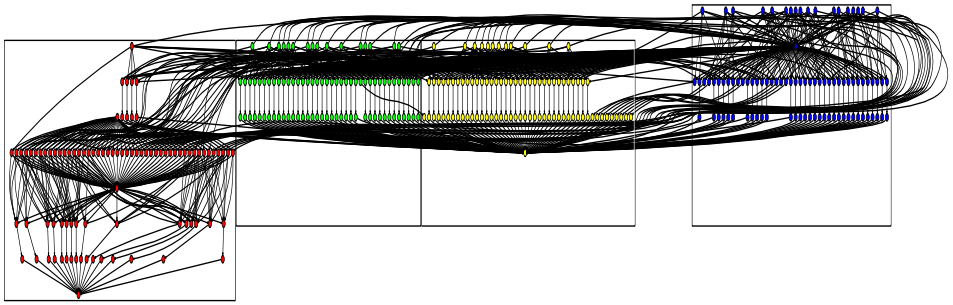

図 3 ワークフローのグラフに対して単純にグラフ分割を適用した結果．一番右のグループは前半のタスク，一番左のグループは後半のタスクに偏ってしまう．

エッジカットを，ノードまたはサイトをまたぐデータの移動と捉えれば，エッジカット最小化という目的は，本研究の目的であるデータ移動最小化に相当する．そこで，データ移動が最小となるようなタスク配置を求める手法として，ワークフローのグラフに対してグラフ分割を適用する手法を考えた．しかし，この手法には以下のような問題がある．単純にワークフローのグラフを分割した結果について，**図 3** に示す．その結果，一番右のグループはワークフロー前半のタスク，一番左のグループは後半のタスクに偏っていることがわかる．これは，並列に実行できないタスクまで含めた全タスクについて等分配したためである．グラフ分割アルゴリズムには，タスクの依存関係，すなわち，エッジの「方向」が全く反映されていない．

この解決策として，ある段階の全出力ファイルに依存するタスクのような，グラフの辺が 1 つの頂点に集まる「集約タスク」を除いたグラフについて，グラフ分割を行う手法を提案した．この手法により，依存関係があるタスクの間での配分が回避され，並列実行なタスクの中でバランスする結果となった．しかし，この手法には，並列実行なタスクの中でバランスする条件が含まれていないため，他のワークフローについてもタスクの適切な配分が行われるとは限らない．そのため,「エッジカット最小」というグラフ分割の条件と同時に,「並列実行可能なタスクについて各グループに等分配する」という条件を含んだアルゴリズムが必要となる．

4. 提 案 手 法

4.1 多制約グラフ分割

本研究では，データ移動を最小化するタスク配置のため，「多制約グラフ分割 (multi-constraint graph partitioning)」[7] というアルゴリズムを利用した手法を提案する．このアルゴリズムの実装は METIS に含まれている．グラフ分割において，グラフ頂点や辺に対してウェイトを設定することがある．グラフ頂点にウェイトを設定する場合，各グループに属する頂点のウェイトの合計が，グループ間でできるだけ等しくバランスする，という条件が与えられる．多制約グラフ分割では，グラフ頂点のウェイトが，スカラーではなくベクトルとなる．そして，各次元のウェイトが，それぞれバランスすることが条件となる．

多制約グラフ分割の用途として提案されているものとして，論文[7] では，ネットワーク上の CPU パワーとメモリの両方に等分配するようにグループ化する問題，および，マルチプロセッサのグルーピングの際に複数の処理段階を考慮する問題が挙げられている．

4.2 タスクの段数

多制約グラフ分割をワークフローのグラフへ適用するために，まず，ワークフローを構成するタスクを，並列実行可能な処理段階にグループ化する．タスクがどの処理段階に属するかは，最初のタスクから依存関係を辿って何段目に来るか，というように段数を数えることによって決める．**図 4** のグラフにこの段数を示している．最初の入力ファイルが依存するタスクを第 1 段とし，そのタスクの次のタスクを第 2 段とする．もしあるタスクが複数のタスクに依存し，それらの段数が異なる場合は，それらのうちで最も大きい段数を基準として，その次の段数とする．このようにして各タスクについて段数を決めると，同じ段数のタスクについてグループ化ができる．同じ段のグループに含まれるタスクは，依存関係がなく，並列に実行可能である．

4.3 ウェイトベクトル

前述のように定めたワークフローの各段について，タスク数が均等になるようにグラフ分割を行うことが，この提案の目的である．多制約グラフ分割は，ウェイトベクトルの各次元ごとに等分配するようにグループ化するアルゴリズムである．そこで，本提案では，ワークフローの各段をウェイトベクトルの次元に対応させる．図 4 にウェイトベクトルの例を示す．

ウェイトベクトルの設定方法は，その段に含まれるタスクの数によって異なる．まず，段に含まれるタスクの数が，分割しようとするグループの数より大きいか等しい場合について述べる．この場合は，ウェイトベクトルの 1 つの次元に 1 を設定し，他の次元には 0 を設定する．1 を設定する次元については，ワークフローの 1 段目がウェイトベクトルの 1 次元目，2 段目が 2 次元目，というように対応させる．つまり，1 段目のタスクについては，ウェイトベクトルは $(1, 0, 0, ...)$ とし，2 段目のタスクについては，$(0, 1, 0, ...)$ とする．

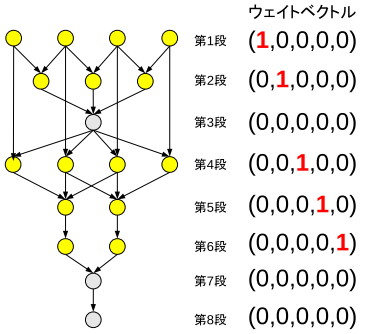

図 4 タスクの段数とウェイトベクトル

次に，段に含まれるタスクの数が，分割しようとするグループの数より小さい場合について述べる．図 4 において，3 段目のタスク数は 1 であり，このケースにあてはまる．この場合，各グループにタスクを 1 つずつ配分するしか解がないため，バランスの計算が不要になる．そこで，このような段については，ウェイトベクトルの対応する次元を設けず，すべての次元に 0 をセットしたウェイトベクトルを与える．このような処置は，計算量を減らすという意味の他に，実装に用いた METIS の問題もある．METIS では，このような場合にウェイト 1 を与えるとエラーとなり，グラフ分割の解が得られない．ウェイトベクトルの次元を省略したため，以降の段について，対応する次元を 1 つ前に詰める．こうして，ウェイトベクトルの次元数は，ワークフローの段の数から，次元を省略した段の数を引いた数になる．また，METIS には，最大次元数が 15 という制限があることからも，次元数を減らすほうがよい．

5. 実　　装

提案手法の評価のため，本研究ではワークフロー実行ツール Pwrake[10),11)] を使用した．Pwrake は，Ruby 版の make 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ツールである．Rake は言語内 DSL (Domain Specific Language) と呼ばれる特徴で示されるように，ワークフローの記述言語は Ruby 言語と等価である．これにより

表 1 2MASS 画像データセットの諸元

画像ファイル 1 枚		データセット	
ファイルサイズ	2.1 MB or 1.7 MB	ファイル数	609
画素数	512 × (1024 or 830)	全ファイルサイズ	1270 MB
画像領域	8.5′ × (17.1′ or 13.8′)	合成画像領域	250′ × 250′

表 2 測定環境

Intrigger	神戸拠点
CPU	Xeon E5410 (2.33GHz)
主記憶容量 (GB)	16
使用ノード数	8
全使用コア数	32

Rake では，Ruby の機能を使った複雑な科学ワークフローの記述が可能である．Pwrake で拡張した部分は，SSH による遠隔実行，スレッドプールによる並列実行の他，Gfarm ファイルシステムのマウント機能，タスク単位のローカリティ機能などである．

Pwrake による並列実行の仕組みを簡単に述べる．まず，タスクの記述ファイルを Ruby スクリプトとして実行し，Rake::Task クラス（およびそのサブクラス）のインスタンスという形で保持する．そのインスタンスがタスクの依存関係などの情報を含んでいる．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．Pwrake では，タスクキューから待ちタスクが無くなると，タスクの依存関係をトレースして，次に実行可能なタスクの一群を取得する．Pwrake にはその際に実行するフックが用意されている．このフックを置き換えることにより，新しいタスク配置のアルゴリズムを導入する．

グラフ分割の実装として，多制約グラフ分割のアルゴリズムを含む METIS ver.5.0pre2 を用いた．METIS の API を Ruby から利用するため，Ruby の拡張ライブラリを実装した．METIS の多制約グラフ分割の API には，METIS_mCPartGraphRecursive および METIS_mCPartGraphKway があり，それぞれアルゴリズムが異なる．しかしこれらの API では，グループを不均等に分割するためのパーティションのウェイトを設定できない．この機能は，今後非一様なマシン環境へ応用する際に必要となることが考えられる．そこで，ParMETIS に含まれる METIS_mCPartGraphRecursive2 を METIS ver.5.0pre2 に合わせて書き換えた．本研究の性能評価ではこの METIS_mCPartGraphRecursive2 を用いる．

6. 性 能 評 価

本節では，提案手法を Montage ワークフローに適用し，その有効性について評価する．

6.1 入力データ

Montage ワークフローに対する入力データとして，本性能評価では，2MASS[16] の画像データを用いた．画像ファイルおよびデータセットの諸元について**表 1** に示す．

6.2 提案手法によるタスク配置と座標との関係

今回の評価では，ワークフローを実行するノード数を 8 ノードとし，提案手法により各ノードにタスクを配置する．そこで，グラフ分割により 8 つのグループに分割する．

Montage ワークフローは，mAdd_1 の段階で，$n \times m$ のタイル領域ごとに画像を結合する．このタイルの数は，ワークフローのパラメータとして指定できる．今回の評価では，このタイルを 4×4 および 8×4 のケースについて評価した．計算ノード数は 8 であるから，各ノードに割り当てられるタイル領域の数は，それぞれ，2 および 4 となる．

提案手法によるタスク配置の結果と，画像の座標との対応について，**図 5** に示す．グラフ分割によるグループ化の結果を，色と番号で示している．黒い格子状の枠が，mAdd_1 のタイル領域を示す．図 5 上では，この 2 つのタイル領域ごとに，グループ化されていることがわかる．このような配置であれば，mDiff および mAdd_1 の双方について，ノードをまたぐデータ移動の機会が小さくなることが理解できる．提案手法はこのように，空間情報を与えなくても，ワークフローのグラフのみから，適切なタスク配置が達成できる．

図 5 下では，上図に比べて，グループの境界がいびつになっており，タイルの境界と一致している場所としていない場所が存在する．METIS のグラフ分割アルゴリズムにおいて，ここで解の探索を終えたわけである．このような配置であれば，隣合う 2 つのファイルを読む mDiff については効率のよい配置であるように見える．しかしタイル領域内のファイルを読む mAdd_1 については，効率的な配置かどうかは明らかではない．これは次に行う実際の測定により評価する．

比較のため，Pwrake に実装されている，従来のタスク配置アルゴリズムの場合に，タスク配置と座標との関係について**図 6** に示した．前提条件として，最初の入力ファイルが，1 ノードに偏っていると仮定した．この場合，どの入力ファイルについても，アクセスの近さは変わらないため，キューから取り出したタスクから順番に計算ノードに割り振られる．この順番は座標とは無関係であるため，図 6 のように座標に無関係なタスク配置となるのである．

6.3 測 定 環 境

測定に使用した環境は，InTrigger プラットフォーム[17] の神戸拠点である．用いた計算

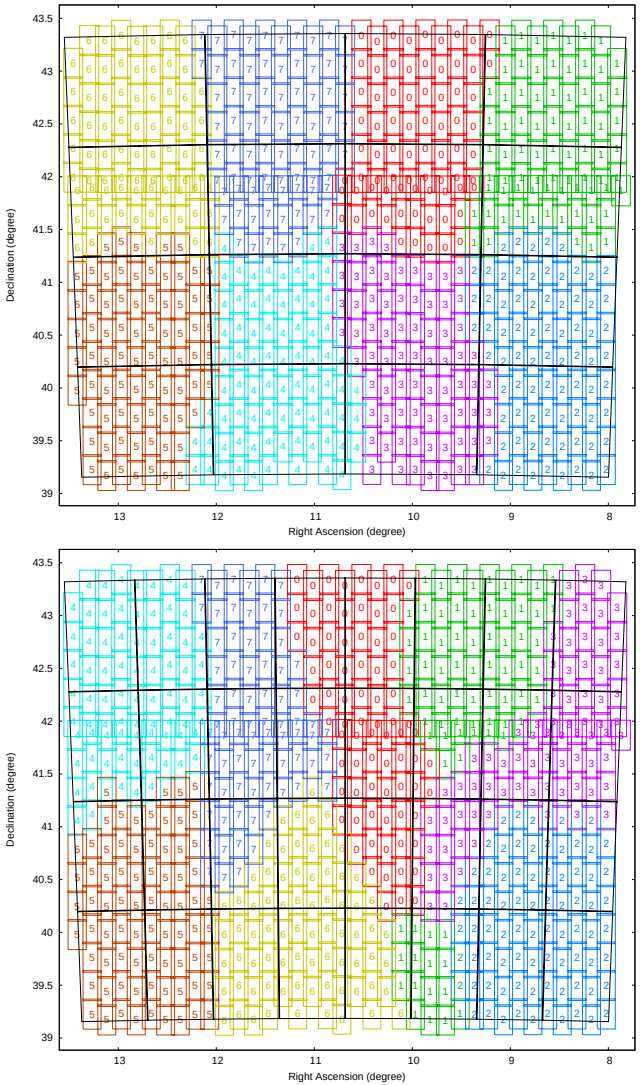

図 5 提案手法によるタスク配置と座標との対応．$mAdd_1$ の段階で，上図が 4×4，下図が 8×4 ののタイル領域ごとに結合．色と番号で示されたグループが，計算ノードへの割り当てに対応する．

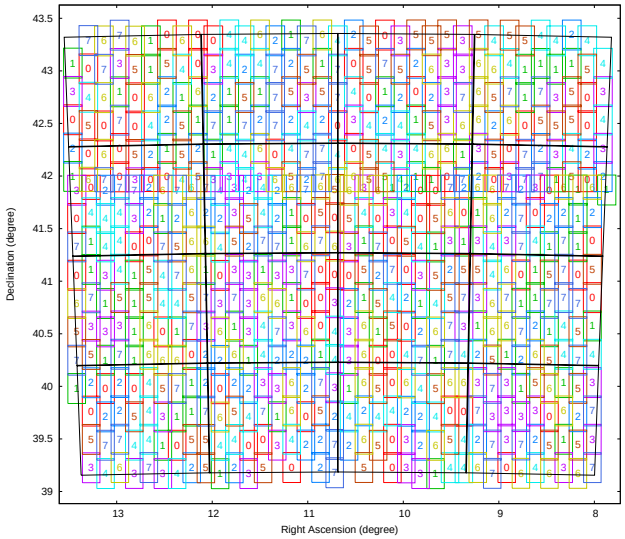

図 6 グラフ分割を用いない場合の計算ノードと座標の関係：従来のタスクごとに局所性を判断するアルゴリズムでは，初期入力ファイルが 1 ノードに偏っている場合，最初のタスクを実行するノードは，座標と無関係な処理順に割り当てられる．

ノードの仕様を**表 2** に示す．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神戸拠点に設置した．

入力ファイルは前述の表 1 のデータセットを，(1) すべて 1 ノードに配置した場合，(2) 提案手法により割り当てられたノードへ gfrep により複製を配置した場合，の 2 種類について調べた．タスク配置アルゴリズムについては，(1) の 1 ノードに配置したケースについて，(A) 無作為：ファイルの局所性について考慮しない配置，(B) 従来手法：タスク単位で，入力ファイルが保存されているノードに配置，(C) 提案手法：多制約グラフ分割を用いたタスク配置，について測定し，(2) の複製したケースについても，(D) 従来手法，(E) 提案手法，について測定を行った．測定は 3 回行い，平均値で示す．

6.3.1 実 行 時 間

それぞれのケースについて，ワークフローの経過時間を測定した結果を**図 7** に示す．こ

こでグラフ分割に要した時間は約 0.03 秒であり，ワークフロー全体に比べて十分小さい．

$mAdd_1$ のタイルが 4×4 のケースについて見ると，(A) 無作為から (C) 提案手法へは，31 % ほど実行時間が短縮された．(B) 従来手法から (C) 提案手法へは，22 % の時間短縮となった．このように，提案手法はワークフローの実行時間を短縮するために有効な手段であることが確認できた．初期データの配置が 1 ノードに偏る場合 (C) と，複製を行った (E) とを比較すると，実行時間の差はわずかである．これは，Montage における最初のタスクである mProjectPP が比較的計算量が多いタスクであり，ファイルを読み込む頻度が比較的低く，アクセス集中が分散されるため，実行時間への影響が小さいのかもしれない．

タイル領域が 8×4 のケースでは，提案手法である (C) と (E) の場合，実行時間は 4×4 のときより若干増加するものの，ほぼ同等であることがわかる．8×4 の分割は，図 5 下からは，最良なタスク配置ではないように見えるが，ワークフローの実行時間からは，このタスク配置によって性能向上を果たしたことが確認できる．

6.3.2 ノード間のアクセス

ワークフロー実行中にアクセスしたデータサイズは，タイル領域が 4×4 のケースでは約 72 GB，8×4 のケースでは 約 73 GB である．このデータ量に対して，異なるノードへのアクセスしたデータ量の割合を**図 8** に示す．4×4 のケースでは，(A) 無作為が 88 %，(B) 従来手法が 47 % であるのに対し，(C) 提案手法では 14 % となり，ノード間のデータ移動が大きく削減された．この傾向は，8×4 のケースでも同様である．提案手法は，ワークフロー実行中のデータ転送量を削減する上で有効であることを示すことができた．

今回の測定では，同一クラスタ内の測定であり，実行時間にして 31 % の削減であったが，この速度はネットワークの速度に依存し，特に距離が離れたクラスタを用いてワークフローを実行する場合，ネットワークのスループットが遅くなれば，データ転送量が遅くなり提案手法のようなデータ移動を最小化する手法はより有効となる．

7. まとめと今後の課題

大量のデータを分散計算機を用いて並列処理するワークフローでは，ノード間のデータを最小にするタスク配置により，性能向上が期待できる．本稿では，最適なタスク配置を求める手法として，ワークフローのグラフに対して「多制約グラフ分割」を適用する手法を提案した．本提案手法を Montage ワークフローに適用して評価した結果，ノード間のデータ移動がデータアクセス全体の 14% に削減され，これにより実行時間が 31% 短縮された．

今後の課題として以下のようなものがある．今回評価したワークフローは，ファイルサイ

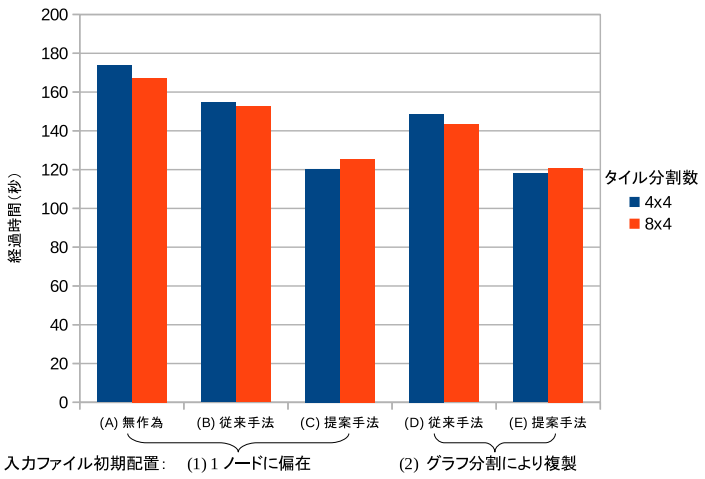

図 7 ワークフローの実行時間

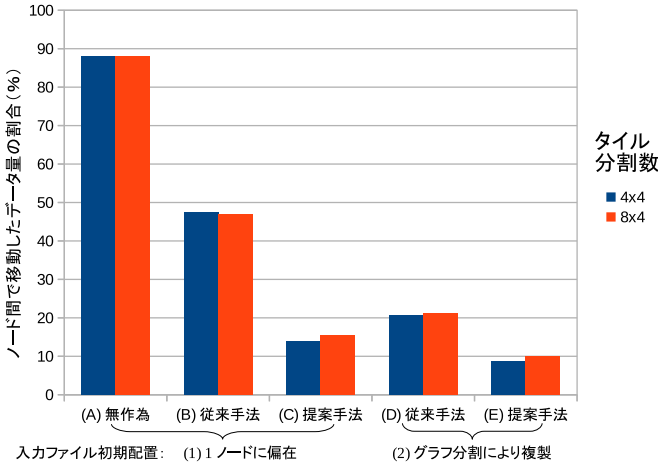

図 8 ワークフロー実行中のノード間のデータアクセスの割合

ズや処理時間が各処理段階で比較的均等であった．処理の大きさに差があるタスクを並列に実行する場合は，グラフの頂点のウェイトとしてタスクの処理時間，辺のウェイトとしてファイルサイズを与えることが考えられる．この場合は，処理時間や生成されるファイルサイズについてどう予測するかという課題がある．また，今回は，1 拠点クラスタ内のタスク配置を考えた．複数のクラスタを用いる場合は，ノード間のスループットを考慮し，クラスタについてのグループ化した上で，さらに，ノードについてのグループ化が必要になる．こうした多段のグループ化についての実装や評価も今後の課題として挙げられる．

謝辞 本研究は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」および，文科省次世代 IT 基盤構築のための研究開発「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) の支援により行った．

参 考 文 献

1) Lustre: http://www.lustre.org/.
2) PVFS: http://www.pvfs.org/.
3) Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol.28, No.3, pp.257–275 (online), DOI:10.1007/s00354-009-0089-5 (2004).
4) TeraGrid: http://www.teragrid.org/.
5) Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C. and Katz., D.S.: Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems, *Scientific Programming Journal*, Vol.13, No.3, pp.219–237 (2005).
6) Taura, K., Matsuzaki, T., Miwa, M., Kamoshida, Y., Yokoyama, D., Dun, N., Shibata, T., Jun, C.S. and Tsujii, J.: Design and Implementation of GXP Make – A Workflow System Based on Make, *eScience, IEEE International Conference on*, Vol. 0, pp. 214–221 (online), DOI:http://doi.ieeecomputersociety.org/10.1109/eScience.2010.43 (2010).
7) Karypis, G. and Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning, *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '98, Washington, DC, USA, IEEE Computer Society, pp. 1–13 (online), available from ⟨http://portal.acm.org/citation.cfm?id=509058.509086⟩ (1998).
8) 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).
9) 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 (online), DOI:http://doi.acm.org/10.1145/1141277.1141456 (2006).
10) Tanaka, M. and Tatebe, O.: Pwrake: A Parallel and Distributed Flexible Workflow Management Tool for Wide-area Data Intensive Computing, *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, New York, NY, USA, ACM, pp.356–359 (online), DOI:http://doi.acm.org/10.1145/1851476.1851529 (2010).
11) Pwrake: http://github.com/masa16/pwrake.
12) Montage: http://montage.ipac.caltech.edu/.
13) 田中昌宏，建部修見：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，先進的計算基盤システムシンポジウム SACSIS2010 論文集，pp.63–70 (2010).
14) METIS: http://www.cs.umn.edu/~metis.
15) SCOTCH: http://www.labri.fr/perso/pelegrin/scotch/.
16) Skrutskie, M.F., Cutri, R.M., Stiening, R., Weinberg, M.D., Schneider, S., Carpenter, J.M., Beichman, C., Capps, R., Chester, T., Elias, J., Huchra, J., Liebert, J., Lonsdale, C., Monet, D.G., Price, S., Seitzer, P., Jarrett, T., Kirkpatrick, J.D., Gizis, J.E., Howard, E., Evans, T., Fowler, J., Fullmer, L., Hurt, R., Light, R., Kopan, E.L., Marsh, K.A., McCallon, H.L., Tam, R., Van Dyk, S. and Wheelock, S.: The Two Micron All Sky Survey (2MASS), *Astronomical Journal*, Vol.131, pp. 1163–1183 (online), DOI:10.1086/498708 (2006).
17) 斎藤秀雄，鴨志田良和，澤井省吾， 弘中健， 高橋慧，関谷岳史， 頓楠，柴田剛志，横山大作，田浦健次朗：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).