Vol.2012-HPC-135 No.26
2012/8/2

ファイルアクセス履歴を用いたデータ集約的ワークフローアプリケーションの高速化

堀内 美希[1,a)]　田浦 健次朗[1,b)]

概要：ワークフローアプリケーションを分散環境で実行する際には，特にデータ集約的なアプリケーションであればあるほど，ファイル I/O にかかる時間は全体のボトルネックになる．ファイル I/O スループットを向上させるためには，適切なタスク割り当てを行いリモートファイルアクセスをなるべく行わないことなどが考えられ，そのようなタスクスケジューリング機構は I/O 性能向上のために重要である．今までに多数の研究がなされてきたものの，既存手法は各ジョブのファイルアクセスが判明していることを仮定している．本論文では，ワークフローアプリケーションの I/O 履歴を収集し，それを用いて各ジョブのファイルアクセス予測を行う．また，その予測を用いたワークフロー実行エンジンのスケジューリング機構の改善を行う手法を提案する．この提案手法を GXP Make に対して実装し，評価用に作成したベンチマークワークフローアプリケーションと，天文学分野で広く用いられているワークフローアプリケーション Montage により提案手法の評価を行った．その結果，ベンチマークアプリケーションでは全ファイルアクセスに対するローカルファイルへのアクセス比率を平均で 85%ほどに，Montage では 23%から 37%に向上させることができた．

キーワード：ワークフローアプリケーション，分散ファイルシステム，スケジューリング最適化

Acceleration of Data-Intensive Workflow Applications by Using File Access History

Miki Horiuchi[1,a)]　Kenjiro Taura[1,b)]

Abstract: I/O time is one of main bottlenecks in workflow applications, especially in data-intensive workflow applications. It is important to reduce remote data accesses to achieve high I/O throughput, which requires appropriate task allocations of a workflow, so a task scheduling takes an important role. It remains a difficult problem how to know I/O files of each job and optimize the scheduling of workflow applications to effectively harness distributed resources, even though many researches have attacked the problem. In this paper, we propose an improvement of scheduling mechanism by gathering file I/O history and expecting I/O files in the next job. We implemented our proposal method on GXP Make, a workflow driver, and evaluated the method by using a synthetic workflow application and Montage, a widely used scientific workflow application. The result shows that the ratio of local file accesses in the synthetic increased to about 85% on average and that in the Montage from 23% to 37%.

Keywords: workflow applications, distributed file systems, optimization of many-task computing scheduling

1. はじめに

分散コンピューティングで用いられるアプリケーションは，依存関係のある比較的小さなプログラムの組み合わせ

[1] 東京大学大学院情報理工学系研究科
The University of Tokyo
[a)] mikity@eidos.ic.i.u-tokyo.ac.jp
[b)] tau@eidos.ic.i.u-tokyo.ac.jp

によって表現されるワークフローアプリケーションの形態を取ることが多い．ワークフローアプリケーションの表現，実行方法には様々なものが存在するが，簡単にワークフローアプリケーションを記述し，実行するためのシステムとして GXP Make[13]，Pwrake[11]，Pegasus[6] などを挙げることができる．また，Hadoop[15] も，ある特定のワークフローアプリケーション実行のためのシステムということができる．これらのワークフロー実行エンジンにおいては，各ジョブ間の情報伝達のために基本的にファイルを用いるため，ファイルをノード間で共有するための仕組みが必要である．この際分散ファイルシステムを基盤システムとして用いると，各ジョブから行うファイルアクセスはローカルファイルシステムにアクセスするのと同じ仕組みで透過的に行うことができる．そのため，各ジョブを構築するプログラムを変更せずとも簡単にワークフローアプリケーションを構築することができる．

分散ファイルシステムを用いたノード間ファイルの共有では，透過的にリモートノード上のファイルアクセスが行えるという利点の一方，いくつか問題点も存在する．ネットワーク越しのファイルアクセスを透過的に行えるということは，逆に言うとワークフロー実行エンジンからはネットワーク越しのファイルアクセスとローカルストレージへのファイルアクセスを区別することが難しく，ファイルアクセスコストを考慮したジョブディスパッチが難しくなる．ファイルコンテンツの位置を問い合わせるための API が実装されている分散ファイルシステムも多数存在するが，その場合においても今からディスパッチするジョブが I/O を行うファイルを予測することは，ユーザからのアノテーションなしでは困難である．例えば GXP Make の場合，ディスパッチを行うジョブはコマンドとして渡され，このコマンドがどのようなファイルに I/O を行うか判断することは容易ではない．

そこで本論文では，ワークフローアプリケーションの I/O 履歴を収集し，それを用いて次に実行するワークフローアプリケーションの I/O ファイルを予測する手法を提案する．また，その予測を用いたワークフロー実行エンジンのジョブディスパッチ機構の改良を行う．これらの実装は，自身で実装を行った分散ファイルシステムとワークフロー実行エンジン GXP Make に対して行う．以後，本論文では 2 章で関連研究に関して述べ，3 章で提案手法の設計について述べる．その後，4 章で具体的な実装方法に関して述べ，5 章で実装したシステムを用いた評価を行う．最後に 6 章で本論文をまとめ，今後の課題について述べる．

2. 関連研究

ワークフローアプリケーションの実行時に，読み込みファイルの位置を考慮したジョブスケジューリングを行う研究は多数行われている [5][7]．例えば，[14] では，Workflow-Aware なストレージエンジンとして，筆者らのグループで開発を行っている分散ストレージシステム MosaStore[4] に改良を加え，ワークフローの依存関係グラフを考慮したジョブスケジューリングを試みている．同じグループでの試みとして [9] においては，分散ストレージシステムとワークフロー実行エンジンの間でクロスレイヤ通信を行い，両者の間での情報交換によるジョブスケジューリング最適化を提案している．

また，[12] では，ワークフローアプリケーションの DAG を Multi-Constraint Graph Partitioning[8] を用いてグラフ分割し，静的にジョブをスケジューリングすることによりデータ移動の少ないスケジューリングを行なっている．ここで用いられている Gfarm[10] にはファイルの位置を問い合わせるための API があり，Pwrake を用いて実装評価が行われている．

本論文では，静的にタスクをスケジューリングする手法を用いず，また Input/Output ファイルに関するアノテーションをユーザに求めることなく，データ移動の少ないジョブスケジューリングを行うことを目指す．ワークフローアプリケーションのファイル I/O 履歴から各ジョブの読み込みファイルを予測し，リモートファイルアクセスをなるべく避けるジョブディスパッチを行うための手法を提案する．

3. 設計

本章では，本論文で提案する，ファイルアクセス履歴に基づいたワークフローアプリケーションスケジューリング改良手法に関して述べる．まず，ワークフローアプリケーションの I/O ファイル予測を行うための手法の設計に関して述べ，その後に，その予測を用いてワークフローのスケジューリングを改善するための方針に関して述べる．

3.1 ファイルアクセス履歴からの I/O ファイル予測手法

ワークフローアプリケーションに限らず，アプリケーションの I/O ファイルをユーザからのアノテーションなしに決定することは容易ではない．しかし，ワークフローアプリケーションでは，

- Input データを変更する
- 実行時のパラメータを変更する

などの各ジョブのためのプログラムを使い回した実行を行うものが多い．本節では，その点に着目して，ファイルアクセス履歴から I/O ファイル予測を行う方法を提案する．

まず，どのようなファイルアクセス履歴を収集するかを述べる．集めるファイルアクセス履歴のフォーマットは以下のものとする（**表 1**）．

(1) 実行時コマンド
(2) PID
(3) 実行ノードのホスト名

表 1　ファイルアクセス履歴フォーマット
Table 1　File Access History Format.

cmd	pid	host	file	created	read log	write log
cat file1	8472	siteA	file1	False	(0-4096)	-
dd if=/dev/zero of=1.txt bs=1MB count=1	2398	siteB	1.txt	True	-	(0-1048576)

(4) ファイルパス
(5) ファイルが新しく作成されたかどうか (True or False)
(6) read 履歴
(7) write 履歴

この情報をワークフローアプリケーション実行時に各ファイルアクセスに関して収集する．今後の方針としてこの情報をリアルタイムに収集しつつ，この情報を用いたジョブスケジューリング改良を行うシステムを構築する予定である．しかし，今回はワークフローアプリケーションを実行したときのアクセス履歴を事前に収集しておき，その解析は事前に行っておいた上で，次のワークフロー実行時に役立てるシステムを目指す．

では次に，このファイルアクセス履歴を用いてこれから実行するワークフローアプリケーションが行うファイルアクセスを予測する手法に関して述べる．この予測の際には，コマンドの先頭の引数（例えば 'cat fileA' というコマンドであれば 'cat' という文字列）を用いる．これは将来的に拡張を考えており，他の引数の情報もうまく利用可能な特徴量をとる予定である．しかし，今回は研究の第一歩としてコマンドの先頭引数のみを特徴量とし，アクセスファイルの予測に用いることとする．アクセスファイルの予測を行う際には，コマンドとファイル名の関係性として，以下の関係性に着目する．

- 引数に出てくる文字列がファイル名そのものである
  - (その引数が) 第 x 番目の引数
  - オプション a の次の引数
- 引数に出てくる文字列 + α がファイル名である
  - 第 x 番目の引数
  - オプション a の次の引数
- 上記以外（ある決まったファイル名にアクセスしている，と捉える）

収集しておいたファイルアクセス履歴のそれぞれに関して，上記の関係性の分類を行い，それぞれの確率を求める．この分析を各コマンド（コマンド先頭の引数）に関して行う．例えば，'command FileA -f FileB' というコマンドが FileA, FileB にアクセス，'command FileC' というコマンドが FileC.txt にアクセスした履歴が得られているとすると，'command' アプリケーションは

- 第 2 引数のファイルに 1/2
- '-f' の後のファイルに 1/4
- 第 4 引数のファイルに 1/4
- 第 2 引数 + '.txt' のファイルに 1/2

の確率でアクセスするという分析を得る．この例での FileB のように，第 x 引数でありオプション a の次の引数であるといった場合には，それぞれの確率を半分にして扱う．提案手法では，これから実行するコマンド（コマンド先頭の引数）が既知のものであれば，この確率と，ファイルアクセスが行われたサイズの積により，アクセスファイル名とサイズの予測を行う．

3.2 ワークフロー実行エンジンにおけるジョブスケジューリング方針

本節では，ワークフロー実行エンジンがジョブをディスパッチする際に，どのような方針でディスパッチするノードを決定するかに関して述べる．あるジョブを実行するときに，どのノードにジョブをディスパッチすることが最適であるかは決して自明な問題ではない．ジョブの実行にかかる時間は CPU，ネットワーク，ストレージの性能に加え，実行中のジョブの数，各ジョブのネットワーク使用状況など，アプリケーション実行中に状況が変わるものの影響も受ける．しかし，それらの要因の中で，本論文ではデータ集約的アプリケーションの高速化のため，以下の要素に注目する．

- 読み込むファイル群（予測）の位置
- 各ファイルの読み込み量（予測）

これらの要素は 3.1 節において述べた I/O ファイル履歴を用いた次回ファイルアクセスの予測手法により予測を行う．つまり，ジョブスケジューリングの際に，次に実行するコマンドに関してこの手法を用いて読み込むファイル群と各ファイルの予測読み込み量を得る．次に，このファイル群の各ファイルの位置を得た上で，各ノードが持つファイルへのファイル予測読み込み量を算出する．最終的にその予測読み込み量が最も多いノードに，優先的にジョブをディスパッチするようにする．このようにしてジョブディスパッチを行うことにより，なるべくリモートファイルアクセスを削減することを目指す．

4. 実装

本章では，自身で実装を行った分散ファイルシステムと，GXP Make を用いた提案手法の実装に関して述べる．分散ファイルシステムは FUSE[1] ベース，内部で扱うブロックサイズは 1MB で実装を行った．システム内ではメタデータサーバ，データサーバ，クライアントの 3 つの役割が存在し，メタデータサーバはシステム内で単一である．

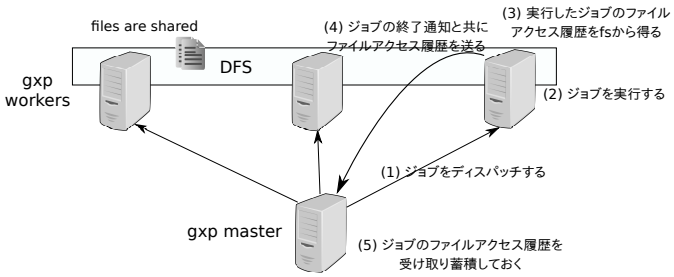

図 1 ファイルアクセス履歴収集機構の実装
Fig. 1 Mechanism for Gathering File Access History.

メタデータサーバはファイルのメタ情報を管理する．レプリケーションの機構は存在せず，ファイルコンテンツはクライアント自身がデータサーバでもあれば，そのファイルが作られたノード上に配置される（そうでない場合はランダムでファイルコンテンツを配置するデータサーバが選ばれる）．この分散ファイルシステムでは，リモートノードからのファイルアクセス時にはネットワークを経由したアクセスが行われ，ローカルノードにファイルコンテンツがある場合には，直接ファイルを該当ファイルコンテンツにアクセスを行う．

以降，4.1 節にて 3.1 節で述べたファイルアクセスの履歴を収集するための実装を述べ，4.2 節にて 3.2 節にまとめた内容の GXP Make のスケジューラへの実装に関して述べる．

4.1 ファイルアクセス履歴収集手法

本節では，ファイルアクセス履歴の収集を行うための実装に関して述べる．ワークフローアプリケーションの実行時に各ジョブの行ったファイルアクセス履歴を収集するために実装に変更を加えた箇所は，以下である．

分散ファイルシステム側

- FUSE の API を用い，open, read, write, close 時に，その要求プロセスの PID を取得する（そこから/proc/pid/cmdline を参照することによりそのプロセスの実行コマンドも得る）機構を実装
- GXP のプロセスとの通信用パイプを用意し，そのパイプを通じて，ある PID が行ったファイルアクセス履歴を返答する機構を実装

GXP Make 側

- ディスパッチしたジョブを実行した後に，実行したジョブの PID によりファイルアクセス履歴を分散ファイルシステムから取得する機構を実装
- 得たファイルアクセス履歴をジョブの終了通知とまとめてマスターノードに送信する機構を実装

これらの実装を行い，ワークフローアプリケーション実行の際に各ジョブが行ったファイルアクセスの履歴を，GXP のマスターノードが得ることができる（**図 1**）．

Algorithm 1 ファイルの位置を考慮したディスパッチノード決定アルゴリズム

for all *job* in ディスパッチ可能なジョブ群 **do**
  bestnode = *job* のディスパッチに最適なノード (3.2 節で述べた方針で選出)
  if *bestnode* のリソースが *job* の必要とする分空いている **then**
    job を *bestnode* にディスパッチ
  else
    job を *leftjob* 群に追加
  end if
end for
leftjob 群のジョブを従来のディスパッチ機構によりリソースの空いているノードにディスパッチ

表 2 実験用マシン基本スペック
Table 2 Basic Spec of Machines for Experiments

CPU	Intel Xeon E5410 4cores (w/ HT 8cores)
メモリ	32GB
OS	Linux 2.6.32 (64bit)
ネットワーク帯域	10Gbps

4.2 ジョブディスパッチノード決定方法

次に，GXP Make へ行った，ジョブディスパッチノードを決定する機構の実装に関して述べる．GXP Make では，Makefile の依存関係の管理は GNU Make に一任しており，依存関係が解決し実行可能となったジョブのみを GXP 側で受け取り，分散実行するという仕組みになっている．従来の GXP Make では，ノードに id として数値が関連付けられており，その数値の順にリソースが空いているものをチェックし空いていればディスパッチを行う，という仕組みを取っていた．本論文では，3.2 で述べた指標に従って，できるだけローカルノードへのファイルアクセスが多くなることを目指したスケジューリングを行う．この際には，分散ファイルシステムでメタデータサーバが管理している ’ファイルコンテンツを持っているノード’ の情報が必要となるため，メタデータサーバにその情報を問い合わせるための API を実装する．ある実行可能なジョブ群をディスパッチするノードを決めるアルゴリズムを **Algotirhm 1** に示す．このアルゴリズムに従い，ジョブを実行するノードを決定する．

5. 評価

本章では，3 章で設計に関して，4 章で実装に関して述べた提案手法の評価を行う．評価は主に 2 つの内容に分かれており，5.2 節で自身で評価用に作成したワークフローアプリケーションを用いた提案手法の評価を行う．その後，5.3 節においてリアルワークフローアプリケーション，Montage を用いた評価を行う．

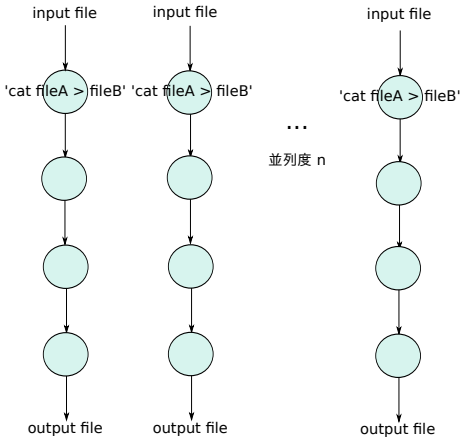

図 2 作成ワークフローアプリケーション（パイプライン型）
Fig. 2 Synthetic Workflow Application (Pipeline).

5.1 実験環境

評価実験の内容に入る前に，実験の際に使用した環境に関して述べる．本評価は InTrigger[2] の hongo クラスタ内の 6 つのノードを用いて評価を行った．使用したマシンの基本性能を**表 2** に示す．4 章で述べた実装を行った GXP Make と分散ファイルシステムを用いて実験を行う．使用する 6 ノードの内 1 つを分散ファイルシステムのメタデータサーバとし，残り 5 ノード中 1 つを GXP のマスターノードとして 5 ノードでジョブを実行する．この 5 つのノードは分散ファイルシステムのデータサーバかつクライアントとなりデータを共有した状態でワークフローアプリケーションを実行する．また，GXP Make の configure で各ノードで並列に実行するジョブの数を，コア数（物理コアの 4 コア）に設定する．これにより，以降の実験では 20 並列までジョブが並列実行される．

5.2 ベンチマークアプリケーション

本節では提案手法の評価を，擬似的に作成したベンチマーク用ワークフローアプリケーションを用いて行う．

5.2.1 アプリケーション詳細とデータセット

ここで使用する擬似ワークフローアプリケーションは，**図 2** に示す流れのワークフローアプリケーションである．各ジョブは 'cat fileA > fileB' (ファイル名はジョブによって異なる) を実行する，データの読み書きのみを行うもので，この流れを 4 回繰り返して最後の出力ファイルを得る．各ジョブの I/O ファイルサイズは全て 256MB とし，はじめの input ファイルはあるノードに全て配置する．また，並列度 n を変化させて実験を行い，各パイプラインのフローは他のものに依存することはない．よってこのワークフローのジョブは並列度 n で Embarrassingly Parallel に実行される．実験はカーネルのページキャッシュをクリアした状態で各 5 回行い，それぞれの平均を算出する．

5.2.2 ベンチマークアプリケーションによる評価結果

ベンチマークアプリケーションによる評価実験中の，ローカルファイルアクセスの比率の比較を**図 3** に，各ジョブの平均実行時間の比較を**図 4** に示す．全ファイルアクセス中のローカルファイルアクセスの比率は提案手法により大幅に向上しており，20 並列まででは基本的に 80%を越えるローカルファイルアクセスとなるスケジューリングができている．それに伴いジョブの平均実行時間も多くの並列度で短くなっている．しかし，20 並列を越えると大幅にローカルファイルアクセスの比率が下がっている．これは，設定によりジョブを 20 並列以下で実行するようにしているため，ジョブが飽和した状態では 1 つジョブが終了するとリソースが空いてジョブがディスパッチされ，ジョブディスパッチ時に input ファイルが存在するノードにディスパッチすることが難しいためである．この解決のためにはジョブ Queue を用いることが考えられるが，これは今後の課題とする．

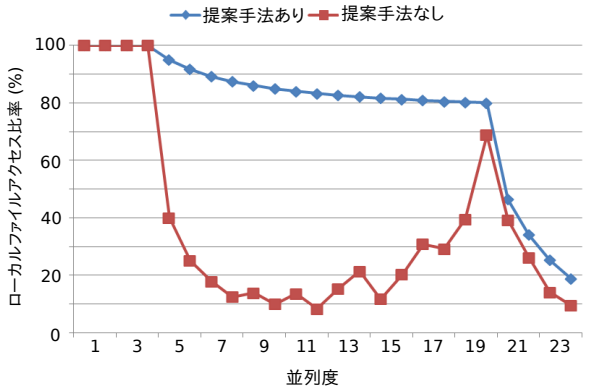

図 3 ベンチマークワークフロー実行時のローカルアクセス比率
Fig. 3 Ratio of Local File Accesses in Synthetic Workflow Application.

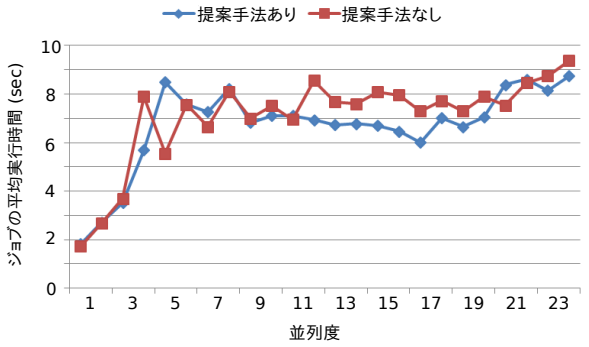

図 4 ベンチマークワークフロー実行時の各ジョブの平均実行時間
Fig. 4 Average Execution Time of Each Job in Synthetic Workflow Application.

5.3 Montage ワークフローアプリケーションによる評価

本節では，広く知られているリアルワークフローアプリケーションである，Montage[3] を用いた提案手法の評価を行う．

表 3　Montage データセット
Table 3　Datasets of Montage.

	data（小）	data（大）
Input ファイルサイズ	2.1MB	1.7MB or 2.1MB
Input ファイルの個数	6	609
合計データサイズ	12.6MB	1270MB
ジョブの個数	19	1542

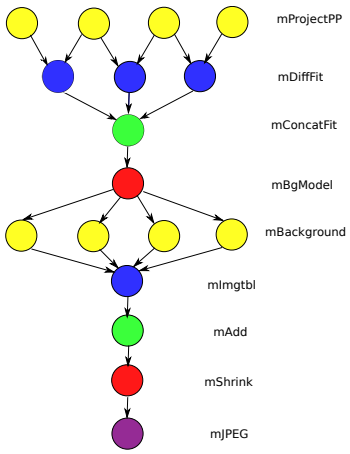

図 5　Montage ワークフローアプリケーションの DAG サンプル
Fig. 5　Sample DAG of Montage Workflow Application.

表 4　ファイルアクセス履歴による Input ファイル予測の精度
Table 4　Accuracy of Expecting Input Files by File Access History.

アプリケーション	予測したファイルへのアクセス率 (%)	アクセスファイル中の予測達成率 (%)
mProjectPP	99.84 (1230/1232)	100 (1230/1230)
mDiffFit	41.14 (5632/13691)	75.56 (5632/7454)
mConcatFit	33.33 (2/6)	0.22 (2/914)
mBgModel	100 (2/2)	100 (2/2)
mBackground	99.89 (1846/1848)	100 (1846/1846)
mImgtbl	33.33 (2/6)	33.33 (2/6)
mAdd	33.90 (20/59)	2.78 (20/720)
mShrink	50 (4/8)	100 (4/4)
mJPEG	100 (1/1)	100 (1/1)

5.3.1　Montage ワークフローとデータセット

Montage は，天文学の分野で有名なワークフローアプリケーションであり，複数の天文画像を組み合わせて最終的に 1 つのモザイクイメージを作成するためのプログラムたちの集合により構成される．Montage の依存関係グラフサンプルを**図 5** に示す．これから実行する Montage のジョブはこのような 9 種類の C 言語で書かれたプログラム（mProjectPP 等）により実行される．

今回の実験では大小 2 通りのデータセットを用いて実験を行う．データセットと，各データセットでのワークフローアプリケーション詳細を**表 3** に示す．

5.3.2　履歴を用いた I/O ファイル予測の正確さ評価

まず，提案手法による I/O ファイルの予測の正確さに関しての評価を行う．この評価では Montage アプリケーションをデータセット大小 2 通りで実行し，これらのファイルアクセス履歴を収集する．そして，データセット小における実行のファイルアクセス履歴より，次にデータセット大を実行した場合にアクセスが予測されるファイルを，提案手法を用いて予測する．この予測と，実際のデータセット大での実行時ファイルアクセス履歴を比較し，評価を行う．本評価は，予測したファイル内実際にアクセスされたファイルの割合と，アクセスされたファイルの内予測が行えていたものの割合，という 2 つの指標により評価を行った．評価の結果を**表 4** に示す．mProjectPP, mBgModel, mBackgound, mShrink, mJPEG などのコマンドにより実行されるジョブは, 提案手法によりほぼ正しく I/O ファイル予測が行えている．しかし一方で，mConcatFit, mImgtbl, mAdd などのコマンドにより実行されるジョブはまだまだファイルアクセス予測が難しいという状況にあり，今後依存関係の解析などによる予測精度の向上を目指していく予定である．

5.3.3　Montage 実行時のジョブスケジューリング手法評価

次に，データセット小でのファイルアクセス履歴を収集した後に，その履歴を用いてデータセット大で Montage を実行した際の提案手法の効果を測定する．5.3.2 項で用いたデータセットと同じものを用いて実験を行うため，I/O ファイル予測は 5.3.2 項で示した精度で行われている．実験は，ワークフロー実行前に存在しているデータはあるノードが全て保持している状態で行う．ワークフローアプリケーションの実行は 4 回行い，結果は全実行の平均を用いている．

データセット大の実行時ファイルアクセス中の，ローカルファイルアクセスの比率を**図 6** に示す．アプリケーションはじめの mProjectPP では，input ファイルがあるノードに集中しているため，ローカルファイルアクセスを優先したスケジューリングが行いにくい．他のジョブでは，特にジョブ数も多く，ファイルの読み込みも多く行う mDiffFit アプリケーションで，提案手法によるローカルファイルアクセスの割合の向上が観測された．ワークフローアプリケーション全体でのローカルファイルアクセス比率も従来手法では 23%ほどであったところ，提案手法では 37%ほどまで向上している．また，各ジョブの実行時間の合計をアプリケーション毎に示した結果が**図 7** である．こちらも特に大きく時間のかかる mDiffFit でローカルファイルへのアクセスが行えたため，2057 秒から 1852 秒と，大きく実行時間を削減できている．しかし，このワークフロー全体に要した時間は両者とも 180 秒弱であり，全体の実行時間はほとんど変化が見られなかった．これはローカルファイルアクセスの優先によりジョブの実行時間は削減できているものの，終了していないジョブを待機し並列度

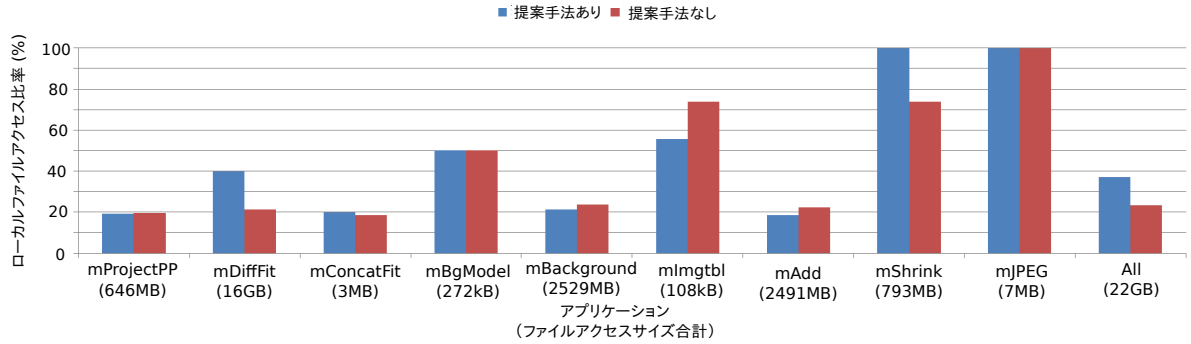

図 6 Montage 実行時のローカルファイルアクセス比率
Fig. 6 Ratio of Local File Accesses in Montage.

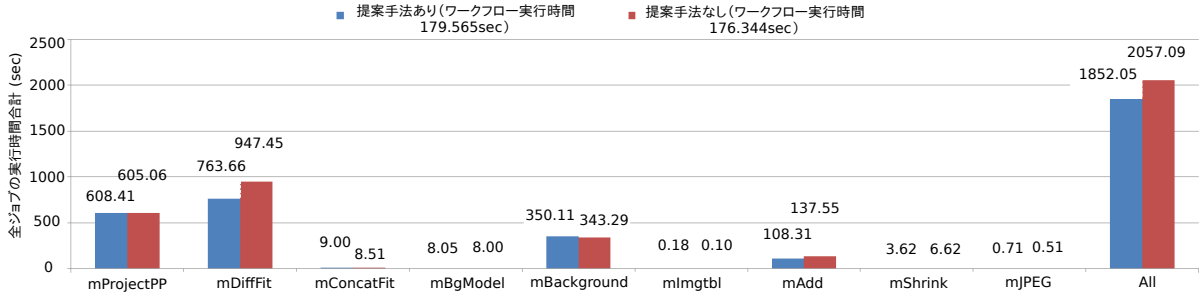

図 7 Montage 実行時のジョブ実行時間合計
Fig. 7 Sum of Job Execution Time in Montage.

の下がる部分があるのではないかと考えている．

6. まとめ

本論文では，ワークフローアプリケーション中の I/O 履歴を収集，活用し，なるべくネットワークを介したファイルアクセスを行わないタスクスケジューリングを行うための手法を提案した．また提案手法を GXP Make と，自身で FUSE を用いて実装した分散ファイルシステムに対して実装し，評価を行った．評価実験は，ベンチマーク用に作成した擬似ワークフローアプリケーションと，天文学分野のリアルワークフローアプリケーション，Montage を用いて行った．提案手法により，ベンチマークワークフローアプリケーションでは約 85%へ，Montage では 23%から 37%への，ローカルファイルアクセス比率の向上が観測できた．

今後の課題としては，I/O ファイル予測の精度向上として，コマンドの先頭引数だけではなく他の引数の情報も用いた特徴量の取り方の検討，多数のリアルワークフローへの適用と改善を行う予定である．実装面ではリアルタイムにワークフローのファイルアクセス履歴を収集しつつ，ファイルアクセス予測を動的に行うシステムを構築する．また，スケジューリングに関しては，実行可能なジョブだけでなくて先に実行することになるジョブ（コマンド）をある程度知った上で，ファイル I/O 予測を行い，先で同時に使用しそうなファイルを同じノードに位置させるなど，ワークフロー全体を見通したスケジューリング方針も，今後検討を行っていく予定である．

参考文献

[1] Fuse :. http://fuse.sourceforge.net/.
[2] Intrigger platform :. http://www.intrigger.jp/.
[3] Montage : An astronomical image mosaic engine. http://montage.ipac.caltech.edu/.
[4] Mosastore :. http://mosastore.net.
[5] Ann Chervenak, Ewa Deelman, Miron Livny, Mei-Hui Su, Rob Schuler, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Data placement for scientific applications in distributed environments. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 267–274, Washington, DC, USA, 2007. IEEE Computer Society.
[6] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237, July 2005.
[7] Frederic Desprez and Antoine Vernois. Simultaneous scheduling of replication and computation for data-intensive applications on the grid. *Journal of Grid Computing*, 4:19–31, 2006. 10.1007/s10723-005-9016-2.
[8] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '98, pages 1–13, Washington, DC, USA, 1998. IEEE Computer Society.

[9] Elizeu S. Neto, Samer A. Kiswany, Nazareno Andrade, Sathish Gopalakrishnan, and Matei Ripeanu. enabling cross-layer optimizations in storage systems with custom metadata. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 213–216, New York, NY, USA, 2008. ACM.

[10] Kohei Hiraga Osamu Tatebe and Noriyuki Soda. Gfarm grid file system. *New Generation Computing*, 28, 2010.

[11] Masahiro Tanaka and Osamu Tatebe. Pwrake: a parallel and distributed flexible workflow management tool for wide-area data intensive computing. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 356–359, New York, NY, USA, 2010. ACM.

[12] Masahiro Tanaka and Osamu Tatebe. Workflow scheduling to minimize data movement using multi-constraint graph partitioning. In *12th International Symposium on Clusters, Cloud, and Grid Computing (CCGrid'12)*, pages 65–72, May 2012.

[13] Kenjiro Taura, Takuya Matsuzaki, Makoto Miwa, Yoshikazu Kamoshida, Daisaku Yokoyama, Nan Dun, Takeshi Shibata, Choi Sung Jun, and Jun ichi Tsujii. Design and implementation of gxp make - a workflow system based on make. In *eScience*, pages 214–221. IEEE Computer Society, 2010.

[14] Emalayan Vairavanathan, Samer Al-Kiswany, Lauro Costa, Zhao Zhang, Daniel Katz, Michael Wilde, and Matei Ripeanu. A workflow-aware storage system: An opportunity study. In *12th International Symposium on Clusters, Cloud, and Grid Computing (CCGrid'12)*, May 2012.

[15] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., June 2009.