並列分散ワークフローシステム Pwrake による大規模データ処理

田中 昌宏[*1 *2]，建部 修見[*1 *2]

Large-scale data processing with Pwrake, a parallel and distributed workflow system

Masahiro TANAKA[*1 *2] and Osamu TATEBE[*1 *2]

Abstract

High-performance parallel processing with a distributed computer cluster is inevitable for handling large-scale science data. We are developing Pwrake, a parallel and distributed workflow system, to enable parallel and distributed processing without special programming technique. Pwrake is based on Rake, a Ruby version of make command. Rake has the feature that workflow definition can be written in programming script, and it brings the power of writing complex scientific workflows. Pwrake provides Rake with functions for parallel and distributed execution and the support for Gfarm filesystem. The workflow of Montage, astronomical image processing software, is written in Rake and its performance is measured using Pwrake. The result shows that Gfarm provides a scalable performance, and that the use of local storage improves performance by 14%. In addition, the result using clusters at two sites also shows a scalable performance improvement.

Keywords: Scientific Workflow, Distributed System, Astronomy Data

概　要

大規模な科学データ処理のため，計算機クラスターによる高性能な並列処理が必要とされている．特別な並列プログラミングを必要とせずにこれを実現するため，私たちはワークフローシステム Pwrake を開発している．Pwrake は Rake というビルドツールをベースにしており，これによりプログラミング言語を活用した高度な科学ワークフロー定義が可能となる．Rake に並列分散機能および Gfarm ファイルシステムのサポート機能を拡張したものが Pwrake である．Pwrake の性能評価のため，天文画像処理ソフトウェア Montage のワークフローを Rake で記述し，Pwrake を用いて実行時間を測定した．Gfarm で実行した結果はスケーラブルな性能向上を示し，ローカルストレージの利用を高めることで性能が 14% 向上した．さらに 2 拠点のクラスタを用いた測定においてもスケーラブルな性能向上を達成した．

1　はじめに

天文学，素粒子物理学，生命科学などの多くの科学分野において，扱うデータ量は年々増加している．こうした膨大なデータを処理するには，並列分散処理が不可欠になってきている．MPI などのフレームワークを用いてプログラムを開発する場合，独特のプログラミング手法を習得する必要があり，敷居が高い．一方，シングルコア用のプログラムを並列に実行することができれば，並列プログラムを書くことなく並列処理が実現できる．その場合，複数のプロセスを並列に実行するために，処理内容や依存関係を記述した「ワークフロー」を記述し，それに基づいてクラスターやグリッド上で並列分散処理を行う．ワークフローの処理系として，グリッド向けとして TeraGrid [16] で用いられる Pegasus [2] や Swift [18] などがあり，

*1 筑波大学計算科学研究センター (Center for Computational Sciences, University of Tsukuba)
*2 独立行政法人科学技術振興機構 /CREST (JST/CREST)

クラスタ向けとして GXP Make[15] などがある．ワークフローシステムの課題として，(1) ワークフローの記述性（記述性の高い言語により，わかりやすくしかも柔軟性が高い記述ができること），および，(2) スケーラブルな性能（計算機の数が増えても計算機 1 つあたりの処理能力が維持されること，すなわち，計算機の数だけトータルで速くなること）が挙げられる．

ワークフローを表現するとき，計算科学ではグラフが用いられる．このとき，タスク（およびファイル）をグラフの「頂点 (vertex)」，それらの依存関係をグラフの「辺 (edge)」として表す．ワークフローのグラフは，依存関係による方向があり，依存関係は循環しないことから，DAG (Directed Acyclic Graph, 有向非循環グラフ) と呼ばれる．Pegasus などのワークフロー処理系では，ワークフローの DAG を XML で記述している．しかし，タスク数が多いワークフローを手作業で DAG として記述することは事実上不可能である．ワークフロー記述言語として GXP では Makefile を採用し，Swift では独自のワークフロー定義言語を開発している．これらの言語は，ワークフローの定義という目的に特化しており，記述力に限界がある．複雑な科学ワークフローを定義するには，さらに記述力の高い言語が求められる．

一方，膨大なデータ量を処理するワークフローの場合，スケーラブルな並列 I/O 性能が鍵となる．優れた並列 I/O 性能を持つファイルシステムとして，Lustre[4], PVFS[9], Gfarm[14] などの分散ファイルシステムがある．特に Gfarm には，計算ノードのストレージを用いることにより，ネットワークを経由しないローカルストレージの I/O を利用でき，高い並列 I/O 性能を達成できるという特徴がある．この特徴を生かすには，入出力ファイルが格納されている計算ノードでタスクを実行する必要があり，その実現にはワークフローシステムの支援が不可欠である．ローカルストレージの活用により性能を上げるという方策は，MapReduce[1] フレームワークにも見られる．MapReduce フレームワークでは，専用の特殊なファイルシステムが用いられることが多く，例えば，Hadoop[3] では，HDFS というファイルシステムにデータを格納する．しかし，専用のファイルシステムは，一般のプログラムからは利用できないという問題がある．その点，Gfarm には，Gfarm ファイルシステムをマウントするための gfarm2fs というコマンドが用意されており，一般のプログラムから Gfarm ファイルの読み書きが可能である．しかし，Gfarm の並列 I/O 性能の特徴を生かす機能を持ったワークフローシステムはこれまでになかった．

以上のことから，著者らは，スケーラブルな並列分散ワークフローを実現するためのシステム Pwrake[13, 10] を開発している．Pwrake は，(1) Rake によるワークフローの記述性，および，(2) Gfarm によるスケーラブルな並列 I/O 性能，を活用することを考えて設計したシステムである．本稿では，Pwrake による大規模科学データ処理の概要について述べる．

2 ワークフローの記述言語としての Rake

並列分散ワークフローシステム Pwrake は，ワークフロー記述言語として Rake の仕様をそのまま利用している．本節では，Rake のワークフロー記述力の高さについて述べる．

Rake は，オブジェクト指向スクリプト Ruby 言語によって記述された，make と同様なビルドツールである．Rake の特徴は，タスクを記述する言語についても Ruby を利用していることである．Rake で定義されたタスク定義のためのメソッドを用いて，あたかもタスク定義言語のように記述することができる．このようにホスト言語（この場合 Ruby）を使って定義された特定用途向けの言語を，内部 DSL (Internal Domain Specific Language) と呼んでいる．Ruby は内部 DSL を定義しやすい言語として知られている．Rake タスクを記述するデフォルトのファイル名は Rakefile である．Rakefile は Ruby スクリプトとして実行されるため，Makefile ではできないような記述が可能になる．このことが，ビルドツールとしてだけではなく，ワークフローの記述に威力を発揮する．

2.1 Rakele 記述例

Rakefile は Ruby の文法に従っており，Ruby スクリプトとしてパースされる．Rakefile を書くためには，Ruby の文法と Rake の仕様の両方を知る必要があるが，Ruby の文法をすでに知っていれば学習コストも低い．次のコードは，プログラムをビルドする Rakefile の例である．

```
SRCS = FileList["*.c"]
OBJS = SRCS.ext("o")

task :default => "prog"

file "prog" => OBJS do |t|
  sh "cc -o prog #{t.prerequisites.join(' ')} "
end
```

```
rule "*.o" => "*.c" do |t|
  sh "cc -o #{t.name} #{t.prerequisites[0]}"
end
```

最初の行で拡張子が .c のファイルリストを得て，SRCS という配列に格納し，次の行でその拡張子を .o に変換して OBJS に格納している．その次以降の task，file，rule で始まる部分が Rake のタスク定義であるが，これらは Ruby の文法に照らせばメソッドコールの記述である．メソッドの引数中にある => の記号は，Ruby のキーワード引数の文法であるが，Rake ではこれを依存関係として解釈する．上の例のように，task メソッドの引数に :default => "prog" が与えられると，:default[*3] というターゲットが "prog" というターゲットに依存する，という意味になる．ターゲットが :default の場合は特にターゲットを省略した際の最終ターゲットとなる．次の file メソッドで，"prog" をターゲットとするタスクを定義している．task の代わりに file を用いると，ターゲット名はファイル名とみなされ，タイムスタンプを比較して入力ファイルが新しい場合にのみタスクが実行される．引数に書かれた "prog" => OBJS のうち，OBJS は冒頭で設定したファイル名の配列である．このように複数のファイルに依存する場合は配列で与える．依存関係の引数の後ろに続く do と end で囲まれた部分は，Ruby のコードブロックである．この部分がタスクのアクションであり，依存関係に従って実行される．ブロック中の sh は Rake で定義されたメソッドであり，引数の文字列をコマンドラインとして実行する．最後の rule はルールの記述である．上の例では "*.o" にマッチするファイルが "*.c" にマッチするファイルに依存する，という意味になる．ここでもファイルのタイムスタンプからアクションの実行が決められる．

2.2　スクリプトを用いた依存関係の定義

Rake では，rule によって make と同様な拡張子ルールを定義できる．これによって，異なる入力ファイルのセットに対しても，ワークフローを書き換えることなく実行することが可能である．さらに Rake は，ファイルの拡張子ルールだけでは定義できないワークフローも記述することが可能である．例えば，B00 が A00 と A01 に依存して，B01 が A01 と A02 に依存する，というようにファイルの番号によって依存関係が決まる場合を考える．Makefile の場合は，次のように各定義を書き下す必要がある．

```
B00: A00 A01
       prog A00 A01 > B00
B01: A01 A02
       prog A01 A02 > B01
B02: A02 A03
       prog A02 A03 > B02
...
```

一方，Rake では，次のようにタスク定義をループの中に書けば，タスク定義を繰り返すことができる．

```
for i in "00".."10"
  file "B#{i}" => ["A#{i}","A#{i.succ}"] do |t|
    sh "prog #{t.prerequisites.join(' ')} > #{t.name}"
  end
end
```

Rakefile は Ruby スクリプトであるから，このようなプログラミングが可能である．

2.3　依存関係がファイルに書かれているケース

科学ワークフローでは，ファイル名からは定義できないような依存関係も考えられる．例えば，ある領域を位置をずらしながら撮像した画像ファイルがあり，領域が重なった部分について処理を行う，というケースを考える．この場合，重なり

[*3]: default のようにコロンで始まるリテラルは , Ruby の Symbol であるが，文字列と思って差し支えない．

部分を持つファイルペアのリストが，次のようにファイルに書かれているとする．

```
$ cat depend_list
dif_1_2.fits image1.fits image2.fits
dif_1_3.fits image1.fits image3.fits
dif_2_3.fits image2.fits image3.fits
...
```

この内容に基づくワークフローを定義することを考える．Make の場合は，依存関係を 1 つずつ Makefile に記述する必要がある．そのためには，awk などを使用してテキスト処理を行い，Makefile を出力するスクリプトを作成する必要がある．一方 Rake の場合は，次の例のように Ruby でスクリプトを記述することにより，完結した記述ができる．

```
open("depend_list") do |f|
  f.readlines.each do |line|
    name, file1, file2 = line.split
    file name => [file1,file2] do |t|
      sh "prog #{t.prerequisites.join(' ')} #{t.name}"
    ends
  end
end
```

2.4 動的ワークフロー

動的ワークフローとは，タスクや依存関係がワークフロー開始時に不確定であり，それらは途中のタスクを実行した結果に基づいて決定されるようなワークフローである．動的ワークフローは，プログラムのビルドにはないケースであるが，科学ワークフローにはしばしば必要になる．例えば，前節の depend list ファイルを出力するタスクがワークフロー中に含まれている場合，そのタスクを実行して depend list ファイルを出力した後で，その内容に基づいてタスク定義が行われなければならない．このような動的ワークフローは，make の場合は，Makefile を動的に生成することによって可能になる．つまり，make の実行中に Makefile を生成し，子プロセスの make としてこの Makefile を実行する．ところが大規模なワークフローの場合，この方法では巨大な Makefile を出力しなければならないという問題がある．またこの場合でも awk といった別のツールが必要であり，Makefile だけでは定義できない．

一方，Rake の場合は，タスクのアクションの中にタスク定義を書く，というようなネスト記述によって実現可能である．簡単な例を次に示す．

```
task :A do
  b = task :B do
    puts "B"
  end
  b.invoke
end

task :default => :A
```

ここで，task B は task A のアクションの中にあり，task A のアクションはパース時には実行されないから，最初は task B は定義されていない．ワークフローが開始された後，task A のアクション部を実行している最中に，task B が定義される．ただしそれだけでは task B は実行されない．というのは，依存関係上，task B は最終ターゲットから必要とされていないためである．task B を実行するには明示的に task B を実行する必要がある．そのため上の例では，task B を定義する `task` メソッドが返したタスクオブジェクトを変数 `b` に格納し，そのオブジェクトに対して `invoke` メソッドを発行している．Rake ではこのようにして動的ワークフローを定義できる．

3　並列分散ワークフローシステム Pwrake

3.1　概要

以上で述べたように Rake は科学ワークフロー記述に必要な特徴を備えている．しかし Rake には並列分散実行機能がない．そこで著者らは，Rake に並列分散機能を拡張した，Pwrake（Parallel Workflow extension for RAKE）[13, 10] を開発している．Pwrake はこれまで 1 度設計変更しており，設計前については別論文 [13] にて発表した．このバージョンでは，Rake から仕様拡張を行った（並列分散実行を定義する `pw_multitask`，および遠隔実行を指示する `rsh` というメソッドを導入した）．しかしこの仕様は Rake と非互換という問題がある．そこで現在のバージョンでは Rake と互換のある仕様を採用している．それにより Rake 用に記述したワークフローは Pwrake で並列分散実行でき，Pwrake で動作すれば Rake でも実行可能である．

現バージョンの Pwrake の実装については，別論文にて発表予定であるため，ここではその概要について述べる．このバージョンでは，`task` や `file` などで定義したすべての Rake タスクについて，並列実行可能なタスクについては並列に実行する，という仕様にした．並列実行可能なタスクを取得するため，キューが空になったときにワークフローの依存関係のツリーを探索する，という実装を行った．並列分散実行の仕組みは，前バージョン [13] とほぼ同じである．使用するホスト名とコア数をファイルに書いておき，`pwrake` コマンドの引数に `NODEFILE=` ファイル名を指定して実行すると，Pwrake はコアごとにワーカースレッドを起動し，ワーカースレッドから SSH によってリモートマシンに接続する．Ruby 1.9 のスレッドはロックにより複数スレッドが同時に走らないという仕様のためマルチコアを利用できないが，Pwrake では，外部プロセスを並列に実行することによって並列性を実現している．

Pwrake の処理履歴をログファイルに出力するには，オプションで `LOG=` ファイル名を指定する．ログファイルには，処理の開始時刻，終了時刻，経過時間，入出力ファイル，実行されたコマンドライン，実行したホスト名が書き出される．このログファイルを見れば，どのファイルに対してどのノードでどういう処理が行われたかがわかる．

3.2　Gfarm サポート

高い並列 I/O 性能を実現するため，Pwrake は広域分散ファイルシステム Gfarm の利用をサポートする．Gfarm には，Gfarm ファイルシステムをマウントするための gfarm2fs というコマンドがある．これにより，通常のプログラムから Gfarm ファイルを直接読むことができる．しかしこのコマンドはシングルスレッドで動作し，1 つのマウントポイントに並列にアクセスしても性能は向上しない．そこで，並列アクセスの場合はマウントポイントをコア数分用意する必要がある．Pwrake はこの手間のかかる処理を自動的に行うことができる．そのためユーザは各ノードでマウントポイントを作成する必要はない．ユーザは，まず Pwrake を実行するノードで gfarm2fs を用いてマウントし，Gfarm にワーキングディレクトリを作成し，そこに Rakefile を置く．それから `pwrake` コマンドに `FS=gfarm` というオプションを付けて実行すると，自動的にすべてのリモートノードに接続し，Gfarm ファイルシステムをコアの数だけマウントし，ワーキングディレクトリに移動してタスクを実行する．

さらに，Pwrake には，効率的なタスク配置のための仕組みが実装されている．1 節で述べたように，ローカルストレージの活用のためには，入力ファイルが格納されている計算ノードでタスクを実行する必要がある．そこで Pwrake は，Gfarm に付属する gfwhere コマンドを用いて，入力ファイルが格納されているノードの情報を得て，そのノードでタスクを実行する，ということを自動的に行う．この仕組みによって，ファイル I/O の比重が高いワークフローにおいて，高い並列 I/O 性能を得ることができる．

4　天文画像合成ソフトウェア Montage

I/O の比重が高い天文データ処理ワークフローとして，Montage [7] というソフトウェアの事例について述べる．Montage は，図 1 のように複数の画像を 1 つの画像に合成（モザイキング）を行う汎用ソフトウェアである．Montage の処理は，いくつかのプログラムを組み合わせたワークフロー処理である．図 2 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 によって入力画像を出力画像の座標系へ投影する．その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ばらつくため，そのまま画像を接合すると空の明るさに段差が生じる．この段差を補正して明るさの変化をスムーズにするための処理が明るさ補正である．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 によって各々の差の画像をその明るさの 1 次成分でフィットする．その結果に基づき，mBgModel により画像全体でスムーズにつながるようにそれぞれの画像に

ついての補正パラメータを計算する．mBackground 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．最後に mAdd プログラムによって 1 枚の画像に統合し，画像合成処理が完了する．

この Montage のワークフローは Rakefile として 200 行程度で記述できる（コメント，空行，その他を含む）．この Rakefile は，異なるデータセットに対してもそのまま適用でき，ワークフローの途中で別の Rakefile が動的に生成されることはない．この Rakefile は GitHub リポジトリ [10] で公開している．

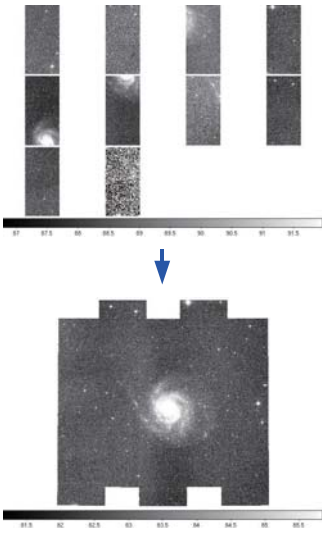

図 1: Montage による天文画像合成

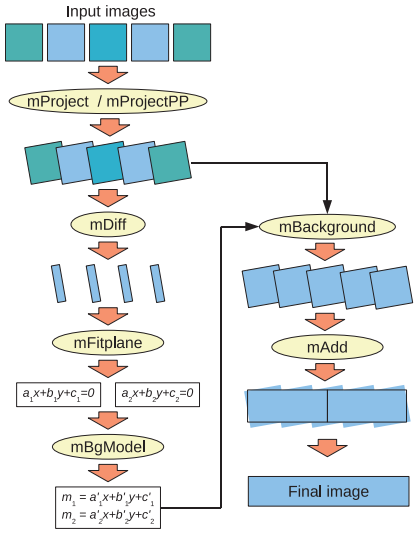

図 2: Montage ワークフロー

5 ワークフローの性能測定

本節では，論文 [13] に掲載した，Pwrake による Montage ワークフローの性能測定について紹介し，若干説明を補足する．本測定で使用した Pwrake は，3.1 節で述べた変更前のバージョンであるが，性能は現バージョンでも同等である．

5.1 測定条件

性能測定に使用した計算機クラスタは，筑波大および産総研に設置されている 2 つのクラスタである．1 拠点での測定には，筑波大のクラスタ 8 ノードを使用した．CPU は AMD Opteron 2218 (2.6 GHz), 4 cores/node, メモリーは 4 GB を搭載する．2 拠点での測定には，前述の筑波大のクラスタに加えて，産総研のクラスタを使用した．産総研のクラスタでは，CPU は Intel Xeon CPU (2.80 GHz), 2 cores/node, メモリーは 1 GB 搭載する．測定対象のファイルシステムは，NFS および Gfarm である．NFS の場合は，それぞれのクラスタ内の別ノードのストレージを 1 つ用いる．Gfarm のストレージは，すべての計算ノードに配置し，メタデータサーバは筑波大に設置した．

ワークフローの入力データとして，2MASS [11] の画像データを用いた．画像フォーマットは FITS である．画像ファイルサイズは 1 枚約 2 MB である．ターゲットとして 6.67 度四方の領域を設定し，ここに 1580 ファイルが含まれる．入力データサイズは 3.3 GB である．

以上のような条件で，Montage ワークフローを Pwrake で実行した際の経過時間を，使用ノード数を変えて測定した．

5.2 1 拠点クラスタでの測定結果

1 拠点クラスタの NFS 上でワークフローを実行した場合の経過時間を，図 3 の #1 に示す．この結果は，コア数が 16, 32 と増えるにもかかわらず実行時間が増加していることを示している．NFS ストレージが 1 つであるために I/O がボトルネックとなり，さらに多数のクライアントから同時にアクセスされるために性能が低下していることがわかる．

1 拠点での Gfarm の測定では，後述する 3 種類の条件で測定を行った．それらの結果を図 3 の #2-#4 に示す．これらはどれもコア数の増加にしたがって実行時間が減少しており，Gfarm を用いることによりスケーラブルな性能を実現できることを示している．

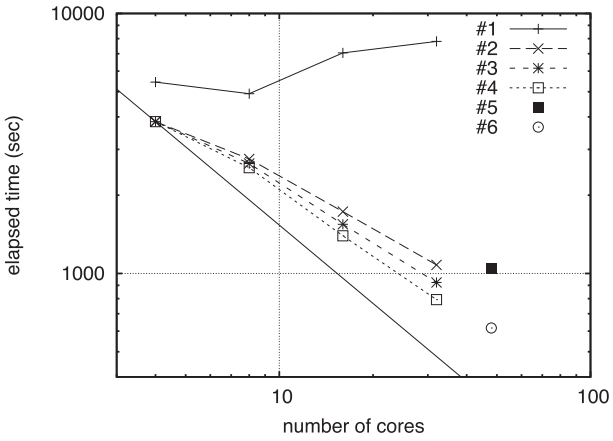

図 3: Montage ワークフローの実行時間．詳細は本文を参照．

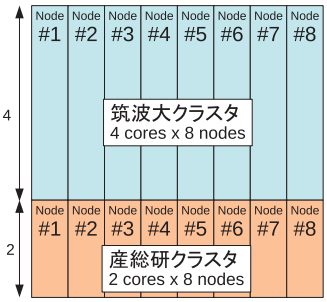

図 4: ノード割り当てのための領域分割．ターゲット領域を図の 16 領域に分割し，入力ファイルを各領域ごとにグルーピングし，ワークフローの初期条件として各ノードに配置．

#2 と #3 の測定条件の違いは，それぞれタスク配置機能の off と on のケースである．前述のように，Pwrake には，入力ファイルが格納されたノードにタスクを配置する機能が備わっている．32 コアの場合の経過時間は，この機能を off にした場合 (#2) は 1,079 秒，on にした場合 (#3) は 923 秒である．実行時間が約 14% 減少し，ファイル格納ノードへタスクを配置する機能には効果があることがわかった．

#3 と #4 の測定条件の違いは，最初の入力ファイルをどのノードに配置するかの違いである．#3 のケースでは最初にすべての入力ファイルを 1 ノードに配置した．この場合，最初の入力ファイルを読むときにそのノードのストレージにアクセスが集中する．一方，#4 のケースでは，入力ファイルを全ノードに均等に分散して配置した（複製は作られていない）．この場合，最初のタスクについても各ノードのストレージに分散してアクセスされる．ただし，中間ファイルは各ノードに分散されるため，どちらのケースでも分散アクセスとなる．32 コアの場合の #4 の経過時間は 741 秒であり，#3 に比べて約 20% 減少し，アクセス分散の効果があることがわかった．

5.3　2 拠点クラスタでの測定結果

2 拠点のクラスタを用いた測定では，筑波大の 8 ノード（ノードあたり 4 コア）と，産総研の 8 ノード（ノードあたり 2 コア）の合計 48 ノードを用いた．

入力ファイルの初期配置として，両方クラスターが入力ファイルセットの複製を持ち，各クラスター内で #4 と同様に分散させ，Pwrake のタスク配置機能を on にしてワークフローを実行した場合の結果を，図 3 の #5 に示す．#5 の実行時間は 1,046 秒であり，1 拠点 32 コアの場合より増加している．この原因を調査すると，mDiff や mAdd など，複数のファイルを入力とするタスクにおいて，入力ファイルのいずれかが別の拠点へアクセスするとき，スループットが低く性能が低下するためであることがわかった．

そこで，入力ファイルの初期配置として，図 4 のように，ターゲット画像の領域を各ノードのコア数に応じた面積の領域に分割し，各領域に含まれる入力ファイルごとにグループ化し，グループごとに各ノードに配置した．この条件で測定を行った結果が，図 3 の #6 である．#6 の実行時間は 617 秒であり，1 拠点 32 コアの場合より性能向上となった．これは，入力ファイルのグループ化により，mDiff・mAdd タスクの複数の入力ファイルが同じ拠点にある機会が高まり，スループットが向上したためである．

このように，タスクやファイルの配置を工夫すれば，複数の拠点のクラスタを使用したワークフローについてスケーラブルな性能が得られることがわかる．しかし，このような配置を手作業で行うには大きな手間がかかり現実的ではない．そこで，その後の研究において，著者らはグラフ分割アルゴリズムを用いて適切なタスク配置を行う手法を開発した [19, 20]．これによって自動的に適切なタスク配置を行うことを可能にしている．

6　すばるデータ解析 SDFRED 1

Montage 以外の天文データ処理ワークフローの例として，すばる望遠鏡主焦点カメラ SuprimeCam のデータ処理ソフトウェア SDFRED 1 [17, 8] をワークフローで記述し，Pwrake リポジトリ [10] の demo ディレクトリにて公開している．このワー

クフローは，SDFRED 1 のマニュアル [12] に記述されている以下の手順を Rakefile で記述したものである．

- step2: bias 引きおよび overscan の切り取り
- step3: flat 作り
- step4: 感度補正（flat fielding）
- step5: 歪補正（distortion correction）および微分大気差補正
- step6: PSF 測定
- step7: PSF 合わせ
- step8: sky の差し引き
- step9: AG probe の影を自動でマスク
- step10: 画像を目で見て，悪い部分をマスク
- step11: 組み合わせ規則作り（matching）
- step12: 組み合わせ（mosaicing）

オリジナルの SDFRED 1 は，C-shell，AWK などのスクリプトおよび C 言語プログラムなどから構成されており，特に複数のスクリプト言語の組み合わせによる記述が複雑である．このスクリプトの大部分を Rakefile で記述することにより，Ruby による一貫したわかりやすい記述にすることができた．

SDFRED 1 は，処理の途中結果をユーザが目視で確認し，調整しながら処理を進めるというように作られている．Pwrake はこのようなインタラクティブな処理も可能である．デモワークフローでは，各ステップのターゲットを step1, step2, ... と設定している．そこで，

```
$ pwrake step3
```

などとターゲットを引数に指定することにより，make と同様に任意のステップまでの実行が可能である．中間ファイルを残してあれば，前回終了した時点から処理を再開することも可能である．さらに Pwrake によって複数の入力画像に対する並列処理が可能となった．

7 まとめ

大規模科学データ処理に向けた高性能な並列分散処理のため，ワークフローシステム Pwrake を開発した．ワークフロー定義言語として Rake を採用することにより，複雑な依存関係や動的ワークフローなど，科学ワークフローの高度な記述が可能になった．Rake をベースに，並列分散実行機能および Gfarm ファイルシステムを活用する機能を実装したものが Pwrake である．天文画像処理ソフトウェア Montage のワークフローを Rake で記述し，Pwrake による並列実行性能の測定した結果，Gfarm ではスケーラブルな性能向上を示し，ローカルストレージの利用を高めることで性能が 14% 向上した．さらに 2 拠点のクラスタを用いた測定においてもスケーラブルな性能向上を達成した．その他の天文ワークフローとして，すばる望遠鏡主焦点カメラ SuprimeCam のデータ処理ソフトウェア SDFRED 1 をワークフローで記述し，並列分散処理に成功した．

Pwrake により，様々なワークフローの並列分散実行が可能となる．天文学だけではなく，バイオインフォマティクス分野でも Pwrake が使用されている [5, 6]．Pwrake は導入が簡易であり，大規模データではなくても，マルチコアを簡易に利用するためのツールとしても利用することができる．今後は，スケジューリングなどの機能拡張，および，さらに大規模なクラスタへの適用のための改良などを行う予定である．

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) Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, Vol. 51, pp. 107–113, January 2008.
2) 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. *Scientic Programming Journal*, Vol. 13, No. 3, pp. 219–237, 2005.
3) Hadoop. http://hadoop.apache.org/.
4) Lustre. http://www.lustre.org/.
5) Hiroyuki Mishima. Pwrake for bioinfomatics workflows using GATK and Dindel. https://github.com/misshie/Workflows.
6) Hiroyuki Mishima, Kensaku Sasaki, Masahiro Tanaka, Osamu Tatebe, and Koh-ichiro Yoshiura. Agile parallel bioinformatics workflow management using pwrake. *BMC Research Notes*, Vol. 4, No. 1, p.331, 2011.
7) Montage. http://montage.ipac.caltech.edu/.
8) M. Ouchi, K. Shimasaku, S. Okamura, H. Furusawa, N. Kashikawa, K. Ota, M. Doi, M. Hamabe, M. Kimura, Y. Komiyama, M. Miyazaki, S. Miyazaki, F. Nakata, M. Sekiguchi, M. Yagi, and N. Yasuda. Subaru Deep Survey. V. A Census of Lyman Break Galaxies at z˜=4 and 5 in the Subaru Deep Fields: Photometric Properties. *Astrophysical Journal*, Vol. 611, pp. 660–684, August 2004.
9) PVFS. http://www.pvfs.org/.
10) Pwrake. http://github.com/masa16/pwrake.
11) M. F. Skrutskie, R. M. Cutri, R. Stiening, M. D. Weinberg, S. Schneider, J. M. Carpenter, C. Beichman, R. Capps, T. Chester, J. Elias, J. Huchra, J. Liebert, C. Lonsdale, D. G. Monet, S. Price, P. Seitzer, T. Jarrett, J. D. Kirkpatrick, J. E. Gizis, E. Howard, T. Evans, J. Fowler, L. Fullmer, R. Hurt, R. Light, E. L. Kopan, K. A. Marsh, H. L. McCallon, R. Tam, S. Van Dyk, and S. Wheelock. The Two Micron All Sky Survey (2MASS). *Astronomical Journal*, Vol. 131, pp. 1163–1183, February 2006.
12) Subaru Data Reduction. http://www.naoj.org/Observing/DataReduction/mtk/subaru red/SPCAM/.
13) 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, pp. 356–359, New York, NY, USA, 2010. ACM.
14) Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda. Gfarm Grid File System. *New Generation Computing*, Vol. 28, No. 3, pp. 257–275, 2004.
15) 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. *eScience, IEEE International Conference on*, Vol. 0, pp. 214–221, 2010.
16) TeraGrid. http://www.teragrid.org/.
17) M. Yagi, N. Kashikawa, M. Sekiguchi, M. Doi, N. Yasuda, K. Shimasaku, and S. Okamura. Luminosity Functions of 10 Nearby Clusters of Galaxies. I. Data. *Astronomical Journal*, Vol. 123, pp. 66–86, January 2002.
18) Yong Zhao, Mihael Hategan, Ben Clifford, Ian Foster, Gregor von Laszewski, Veronika Nefedova, Ioan Raicu, Tiberiu Stef-Praun, and Michael Wilde. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. *Services, IEEE Congress on*, Vol. 0, pp. 199–206, 2007.
19) 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, May 2010.
20) 田中昌宏，建部修見．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．情報処理学会研究報告 2011-HPC-130 (SWoPP2011), July 2011.