Vol.2009-ARC-185 No.1
2009/10/26

複数の並列計算環境に対応したMapReduceのPythonによる実装

高田祐輔[†1]　Eric Heien[†1]
置田真生[†1]　萩原兼一[†1]

MapReduceの既存実装の多くは大規模なクラスタ環境を前提としており，マルチコア環境やグリッド環境下で容易に実行できるシステムは存在しない．しかし，大規模な環境だけでなく，小規模な環境における並列計算にもMapReduceは有用と考えられる．

本研究の目的は，クラスタ環境に限らず，マルチコア環境などユーザが所有している並列計算環境で容易にMapReduce計算を実行可能にすることである．そのために，Pythonの並列計算用モジュールであるPyMWを拡張し，ユーザが環境の違いを意識せずにMapReduceプログラミングできるシステムを実装する．また，実行時に環境に応じた通信方法へと自動的に変更するよう拡張することで，性能向上を目指す．

結果，マルチコア，クラスタ環境で動かす際はプログラム中の実行環境を設定する1行を修正することにより実行できた．これにより，プログラム中の環境依存部分を隠蔽し，ユーザが計算したい問題を解く部分のプログラミングに集中できる環境を構築できた．

MapReduce Implementation in Python for multiple Parallel Computing Environments

Yusuke Takata,[†1] Eric Heien,[†1] Masao Okita[†1]
and Kenichi Hagihara[†1]

Almost all MapReduce implementations require a cluster environment, so no flexible implementation exists for multicore or grid environments. However, we think that MapReduce is useful in not only large-scale but also small-scale computation environments.

We have developed an implementation of MapReduce for multiple parallel computation environments, such as multicore, cluster and grid environments. This implementation allows users to ignore the difference of environments while programming. To achieve this, we extend PyMW, which is a Python module for parallel computation. Moreover, we automatically apply environment specific communication to improve data transfer.

Our experiments show that users only have to modify one line in a program for execution in a different environment. Thus, we achieve a non-environment specific MapReduce implementation.

1. はじめに

インターネットの普及や記憶装置の大容量化に伴い，それらに付随する大量のデータを高速に処理する要求が高まっている．例えば，Google社は1日20ペタバイトを超えるデータを処理しており，その量は現在も増加している[1)]．

大規模処理の手法として，Google社はMapReduce[1),2)]を提唱している．これは大量のデータを大規模な並列計算機環境で処理するためのプログラミングモデルおよびその実装であり，ユーザはMapおよびReduceという2つの関数を定義したユーザプログラムを作成するだけで，大量のデータを効率よく処理できる．実際に，Google社ではWeb検索のためのインデックス作成などに使用している．さらに，Google社の実装以外にもHadoop[3)]など主にクラスタ環境を対象としたMapReduce実装が数多く提案されており，ウェブ検索分野などで活用されている．

ただし，マルチコア環境をはじめとする小規模な並列計算環境においても，Chuらの研究[4)]などで示されているように，MapReduceは並列処理のプログラム記述を容易にする点で有用と考えられる．しかし，MapReduceは大規模な並列環境での運用を前提として提唱されているため，クラスタ環境向けの実装が多く，クラスタを持たないユーザにとってMapReduceのプログラミングおよび実行は容易ではない．また，クラスタ環境向けの実装においても，耐障害性を高め，通信量を削減するためにHDFS[3)]など特別なファイルシステムを用いる実装もあり，これらはユーザにとって導入の敷居が高い．

一方，Python言語[5)]は幅広いプラットフォームで動作するインタプリタ言語である．Pythonの特徴は，可読性を重視して文法が設計されているため，習得が容易な点である．また標準で多数のモジュールが添付されており，多くの機能が使用可能なためプログラムの記述も容易である．

†1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

Python の拡張モジュールである PyMW[6] は，マスタワーカ型の並列計算を容易にプログラミングおよび実行する機能を提供する．PyMW はマルチコア，クラスタおよびグリッド環境に対応しているが，ユーザはプログラミング時にはプログラムの実際の実行環境を考慮する必要がなく，並列計算を行いたい部分のプログラミングに集中できる．

本研究の目的は，ユーザが扱える並列環境で容易に MapReduce を実行可能にすることである．ユーザが扱える環境として，マルチコア，クラスタおよびグリッド環境を想定している．したがって，複数の環境に対応する必要があるが，一般にコンパイルが必要な言語では，OS ごとにライブラリを用意することや再コンパイルが必要となる．また，並列環境が異なれば通信方法が異なる別のプログラムを作成する必要がある．しかし，ユーザにとっては使用する並列計算環境に依らず，計算したい問題を解く部分のプログラミングのみに集中でき，プログラムの実行も容易であることが望ましい．

そこで PyMW を拡張し，一つの並列プログラムを書けば複数の環境で動作可能な MapReduce 実装を提案する．また，PyMW への MapReduce 機能の追加と併せて，大量のデータを扱う MapReduce 計算の特性に合った通信方法に自動的に変更することで，性能向上を目指す．

以降では，まず 2 章で，関連研究について述べる．3 章で MapReduce，4 章で PyMW について説明する．5 章では，提案する PyMW の拡張の詳細を述べる．6 章では，提案手法の評価とその考察について述べる．7 章では，まとめと今後の展望について述べる．

2. 関 連 研 究

Google は MapReduce[1),2)] を提唱したが，その実装は公開されておらず，一般のユーザは入手できない．また，大量のデータを効率よく処理するために Google 独自のファイルシステム（GFS: Google File System）[7] 上で運用するのを前提として MapReduce を実装しているため，一般のユーザは Google 社の MapReduce を扱えない．GFS は耐障害性や，ファイルを異なる計算機から参照する際にネットワーク上で近い位置にあるファイルのコピーを参照することで通信データ量を削減する機能を持つ．

Hadoop は MapReduce プログラミングモデルを実装するオープンソースプラットフォームであり，一般のユーザが自由に使用できる．しかし，クラスタ環境での使用を前提としており，マルチコア環境およびグリッド環境には対応していない．本来，MapReduce は大規模な並列計算環境での運用を前提として提唱されているものだからである．

Amazon Elastic MapReduce[8] は，Amazon が提供するクラウドコンピューティング環境の上で Hadoop を用いた MapReduce を実現している．

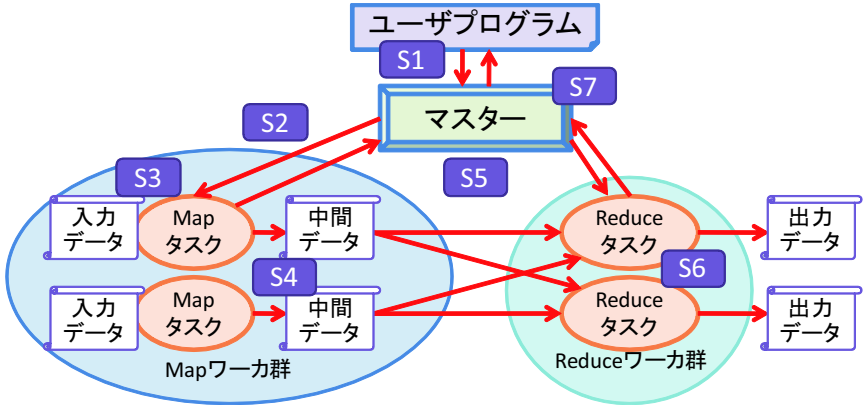

図 1 MapReduce の処理の流れ
Fig. 1 Processing stream of MapReduce

Phoenix[9),10)] はマルチコア環境向けの MapReduce であり，特に NUMA アーキテクチャを意識した実装を行っている．

山下らの研究ではクラスタ上でジョブスケジューラを使用できる MapReduce を実装している[11)]．これにより，共有計算機などでジョブスケジューラを使用しなければならない環境でも MapReduce を実行することが可能となる．

小川らの研究では，MapReduce の大量にデータを読み書きする特性を踏まえ，高速なストレージである SSD（Solid State Drive）のための MapReduce を提案している[12)]．

また，pyMPI[13)]，mpi4py[14)] および Parallel Python[15)] など Python には様々な並列計算のためのモジュールが提供されている．しかし，これらのモジュールはクラスタ環境での利用を前提としており，マルチコア環境での利用を想定していない．

3. MapReduce プログラミングモデル

MapReduce では，ユーザが定義した Map および Reduce という 2 つの関数を，キーと値の多数の組からなる大規模なデータに適用する．Map 関数にはデータ中の各項目に対して処理を行う処理を記述し，Reduce 関数には Map 関数の出力を併合する処理を記述する．Map 関数は一組のキーと値から多数のキーと値の組を中間データとして生成する．この後，

MapReduce 機能を提供するライブラリが中間データのうち，同じキーに対する値を一つのリストにまとめる．Reduce 関数は中間データの各キーに対する値を併合し，出力データを生成する．

図 1 に Google 社の MapReduce 実装の処理の流れを示す．Google 社の MapReduce 実装では，1 つのマシンをマスタとして選ぶ．マスタは他のマシンをワーカとし，各タスクの実行を統括する．入力ファイルは，最初から Map ワーカまたはネットワーク上で Map ワーカに近いワーカのディスクに置かれている．出力ファイルの場所は Reduce ワーカが指定する．MapReduce 計算は，以下のように処理を行う．

（S1） ユーザプログラムがマスタに Map および Reduce 関数と入力データの情報を送る．
（S2） マスタは入力データの位置を見て各ワーカに Map タスクを割り当てる．
（S3） Map ワーカは入力データを読み込み，計算を行う．
（S4） Map タスクの出力は Reduce タスクの個数に分割・ソートされ，中間データとして Map ワーカのローカルディスクに保存される．
（S5） Map ワーカは中間データの位置をマスタに通知し，マスタがその位置を Reduce ワーカに送る．
（S6） Reduce ワーカは各 Map ワーカと通信し，Map タスクの出力を読み込んで計算を行い，出力ファイルを作成する．
（S7） 全てのタスクの終了を確認したら，マスタはユーザプログラムの結果取得要求があるまで待つ．

4. PyMW

PyMW は複数の環境でマスタワーカ型の並列計算を使用することを目的として開発されている Python モジュールである．PyMW では，ユーザが逐次的にタスクを投入すると，投入されたタスクはワーカに自動的に割り当てられ処理される．このとき，PyMW はユーザプログラムとワーカの処理を非同期に実行することで，複数のタスクを並列処理する．これにより，ユーザが並列処理を意識せずとも，マスタワーカ型プログラムを記述することが可能である．

並列計算を実行するときにユーザが指定するのは，ワーカに計算させたい関数，入力データおよびワーカ数や計算を行う環境などの環境指定のためのいくつかのパラメータのみである．また，PyMW は指定された環境に応じて，通信方法などの環境に依存する処理を切り替え，並列計算を実行する．したがって，異なる環境で実行する場合も，プログラム中の環境を指定する部分を数行変更するだけで実行できる．各タスクが完了すると PyMW 内に結果が登録され，ユーザが必要とするときに結果を返すことができる．

5. PyMW による MapReduce

5.1 設　　計

本研究の目的を実現するためには，図 2 に示す F1 から F10 までの機能が必要である．

MapReduce として必要な機能は，F2 から F9 までの 8 項目である．F2 および F3 は計算する関数である．これは図 1 のユーザプログラム内で作成する．F4 は MapReduce タスクの実行と結果取得で用い，S1 および S7 が該当する．F5 は各 Map および Reduce タスクの入出力に関係し，S2，S5 および S7 で使用する．

F6 は Map タスクの出力を Reduce タスクに渡すために S6 で使用する．F7 は各タスクを並列実行するために必要であり，S3 および S6 で用いる．F8 は Map タスクの前処理であり，S2 で使用する．F9 は Reduce タスクの前処理であり，S4 で使用する．

一方，F1 は 1 つのユーザプログラムで複数の環境で MapReduce を実行するために必要である．F10 は提案手法の性能向上を目指すために必要な機能である．

図 2 に示すように F1，F5 および F7 は PyMW を使うことで解決できる項目である．複数環境への対応として，PyMW はマルチコア，クラスタおよびグリッド環境を想定している．

提案手法で実現する項目は，F4，F6，F8，F9 および F10 である．このうち，F4 は PyMW を既存の機能を拡張することで実現し，それ以外の機能は新しく作成する．

最後に，ユーザが担当しなければならない作業は F2 および F3 の作成である．また，F1 で選択する環境および F4 で用いる関数呼び出しはユーザプログラム中に記述する必要がある．

5.1.1 プログラムインタフェース

ユーザは Python を用いて，提案するモジュールを使用したユーザプログラムを書くことで，本手法を用いた計算を行う．ユーザプログラムには，計算したい Map および Reduce の 2 つの関数を定義し，並列計算のためのパラメータを記述する．並列計算のための主なパラメータは，環境名，ワーカ数，データの入出力方法，Map 関数および Reduce 関数が使用する Python モジュール名である．実行環境を変更する際，Map および Reduce 関数の修正は不要であり，並列計算のためのパラメータのみを修正する．

ユーザプログラムは大きく 4 つの部分に分かれる．Map 関数，Reduce 関数，環境の設定およびタスク実行・結果取得である．

このうち Map 関数および Reduce 関数は，F2 および F3 であり，ユーザが計算したい関数を定義する．それぞれの関数の入力はリスト型とする．リスト内の各要素は，一般的な MapReduce プログラミングモデルであるキーと値の組だけでなく，ユーザが作成する Map 関数および Reduce 関数の動作に合っていれば 1 要素のみのリストのように別の要素でもよい．

環境の設定は，図 2 の F1 の使用にあたる．マルチコア，クラスタなど使用する環境を指定し，PyMW の関数を呼び出してその環境向けのマスタを生成する．ユーザが計算の実行や結果の取得を行うときはマスタを介して行う．

タスク実行・結果取得は，図 2 の F4 の使用にあたる．タスク実行では，マスタに Map 関数，Reduce 関数，ワーカ数および入力データを引数として与えて MapReduce 計算を実行させる．MapReduce 計算を開始後は，ユーザプログラムは結果が必要になるまで MapReduce 計算と非同期に実行できる．計算結果は，必要となったときにマスタから取り出す．結果を取得しようとするときに計算が終わっていない場合，計算が終了するまで待つ．

5.1.2 MapReduce 機能

拡張した PyMW 中の MapReduce 機能について述べる．

図 2 の F4 は，PyMW の機能を MapReduce 用に拡張することで実現する．PyMW における並列計算では，個々のタスクの実行ごとにユーザが関数呼び出しを行うが，提案手法ではユーザが一度の呼び出しを行うことにより PyMW 内部で MapReduce 計算を実行するための関数を作成する．一方，結果取得では各 Reduce ワーカの出力をマスタが一つにまとめてユーザプログラムに返す．

F6 は，Map ワーカの出力を Reduce ワーカに渡す処理である．PyMW ではワーカ間の通信は行われないので，新しく作成する必要がある．提案手法ではマスタを介して通信を行っている．

F8 および F9 は，Map ワーカおよび Reduce ワーカそれぞれへの入力データをどのワーカに割り振るか決める処理である．提案手法では全てのワーカへの入力データ量が均等に近づくよう分割している．中間データの場合は分割する前にソート処理を Map ワーカで行っている．

最後に，F10 について説明する．PyMW では入力データはユーザプログラムからマスタへ送られ，さらにマスタからワーカに送られる．また，MapReduce 計算を行う場合，Map ワーカから Reduce ワーカへのデータの受け渡しはマスタを介して行われる．この方法ではマスタ・ワーカ間の通信量が大きくなるほど入出力のオーバーヘッドが増加する．そこで，

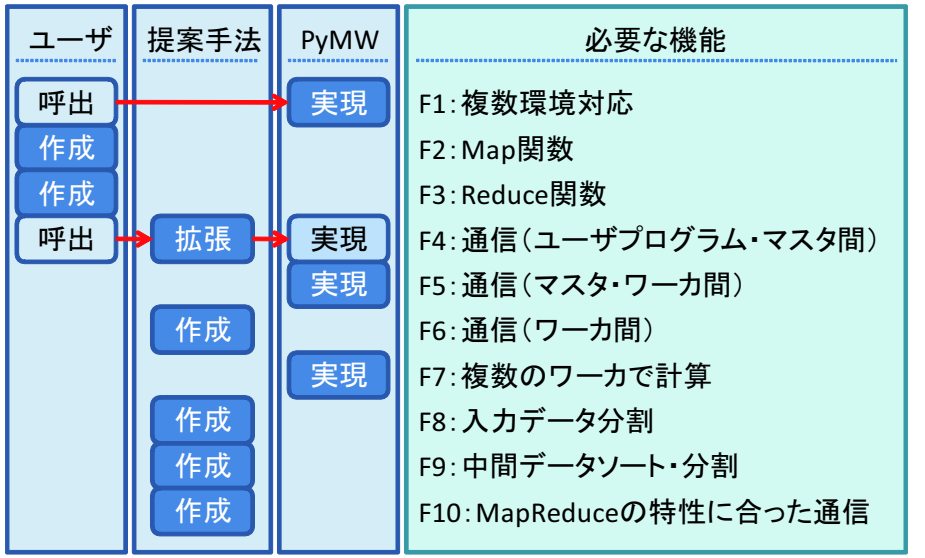

図 2 提案手法の設計方針
Fig. 2 Design requirement

提案手法では，マルチコア環境や NFS などの共有ファイルシステムが利用できる環境の場合，マスタ・ワーカ間で入出力ファイルのパスと処理する位置を記述したリストを送る方式も追加する．これによりマスタ・ワーカ間の通信量が削減できる．

6. 提案手法の評価

この章では，提案手法を次の 3 点について評価する．

(1) 異なる環境への移植性
(2) 導入・設定の容易さ
(3) MapReduce アプリケーションの性能

6.1 実験準備

性能評価をするにあたって，使用する実験環境と MapReduce プログラムについて説明する．

6.1.1 実験環境

表 1 にマルチコア実験環境，表 2 にクラスタ実験環境を示す．PyMW のマスタは，マル

表 1 マルチコア実験環境
Table 1 Multicore experiment environment

OS	Windows XP Professional 64bit
CPU	Xeon 2.66GHz × 4 コア
メインメモリ	32GB
Python	Python 2.6.1

表 2 クラスタ実験環境
Table 2 Cluster experiment environment

計算ノード	hp Proliant DL140G2 × 30
OS	CentOS 5.2
CPU	Xeon 2.8GHz × 2 個
メインメモリ	2GB
Python	Python 2.6.1
MPI	MPICH
通信	Gigabit Ethernet
Hadoop	Hadoop0.18.3

チコア環境ではワーカが動作するコアの一つで実行しているが，クラスタ環境ではワーカが働くコアとは別のコアで実行している．

6.1.2 単語カウント

本稿では，単語カウントプログラムを使用して MapReduce アプリケーションの性能評価を行う．単語カウントとは，テキストファイル中の全ての単語について出現回数を計算するプログラムである．単語カウントでは，計算量に対して通信量が大きくなる．この実験では 134 個計 100MB の英文テキストファイルを入力として使用した．

Map 関数では，各ワーカが担当するファイルから全ての単語について [単語，1] として key と value の組を作る．作った組のリストを Reduce 関数への入力とし，Reduce 関数では，各単語（key）ごとに 1（value）を足していき，[単語，出現回数] のリストを求める．

6.2 異なる環境への移植性

提案手法では，マルチコア，クラスタおよびグリッドと複数の環境で実行可能であるが，ユーザは環境ごとに異なったプログラムを書く必要はなく，一度プログラムを書けば環境設定部分を変更するだけでそれぞれの環境で実行できる．

実際に，異なる環境での実行時のユーザプログラムの修正は環境指定部分の 1 行のみで，Map 関数および Reduce 関数など他の部分は全て同一のプログラムで実行できた．このことから，環境を意識しない MapReduce プログラミングを実現できたといえる．

6.3 導入・設定の容易さ

本研究と既存の MapReduce 実装である Hadoop を，導入・設定の容易さについて比較する．

6.3.1 導　　入

既存の MapReduce 実装である Hadoop と提案手法について導入の容易さを比較し，表 3 に示した．

表 3 導入の比較
Table 3 Comparison of adaption.

	Hadoop	提案手法
実行環境	UNIX クラスタ	マルチコア，クラスタ，グリッド
インストール	クラスタ環境の設定が必要	インタフェースによっては設定が必要
使用言語	Java1.6 以上	Python2.5 以上
他言語のプログラム	標準入出力を介して可能	ファイル・標準入出力を介して可能
ファイルシステム	HDFS	特別なものは用いない

大きな違いとして，Hadoop は UNIX クラスタ環境での実行しか想定されてないが，本研究ではマルチコア，クラスタおよびグリッド環境それぞれでの実行が可能である．

また，Hadoop は HDFS を用いているが，本研究では特別なファイルシステムは用いていない．HDFS を用いると Hadoop で扱うためにユーザが手動で HDFS 上へファイルを移動しなくてはならず，また HDFS 上のファイルは自由に操作できないといった制限がある．提案手法では柔軟なファイル操作のために HDFS のようなファイルシステムを用いない．ただし，そのために性能は Hadoop よりも低下する．

ユーザプログラムの作成言語はそれぞれ Java と Python であるが，他の言語のプログラムも実行可能である．

6.3.2 設 定 項 目

Hadoop では設定ファイル hadoom-site.xml によって 152 項目が設定できる．提案手法ではユーザプログラムから明示的に設定できる部分が 10 項目前後である．PyMW 自体のソースコードからさらなる設定を行うこともできるが，基本的な MapReduce 計算を行うだけならこの 10 項目前後の設定で十分可能である．これらのことから，提案手法は Hadoop よりも少ない設定項目で MapReduce が実行可能である．

同一環境でのノード数の切り替えについて比較すると，Hadoop ではノード数を変更する際，次の手順が必要である．

(1) Hadoop デーモンの停止
(2) 使用するノードを指定する設定ファイル slaves の書き換え
(3) Hadoop デーモンの起動

これに対し，提案手法ではユーザプログラム内で使用するノード数を指定するだけでよい．

6.4 MapReduce アプリケーションの性能

図 3 および図 4 にマルチコアおよびクラスタ環境それぞれの実行時間を示す．また，図

表 4 提案手法による単語カウントの実行時間の内訳
Table 4 Detailed processing time

環境	ワーカ数	初期化	入力データ分割	Map	中間データソート・分割	Reduce	総実行時間
マルチコア	1	0s	1s	105s	4s	37s	148s
マルチコア	2	0s	1s	72s	4s	20s	98s
マルチコア	3	0s	2s	52s	3s	15s	73s
マルチコア	4	0s	1s	45s	4s	12s	63s
クラスタ	1	2s	3s	162s	19s	85s	271s
クラスタ	2	3s	5s	82s	19s	44s	153s
クラスタ	4	6s	4s	42s	17s	28s	97s
クラスタ	8	9s	4s	25s	17s	15s	70s
クラスタ	16	15s	4s	12s	17s	7s	55s

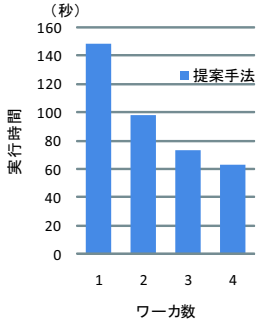

図 3 マルチコア環境の実行時間
Fig. 3 Processing time in multicore environment

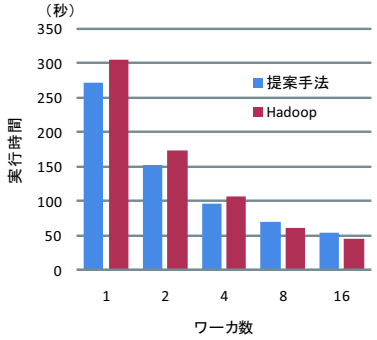

図 4 クラスタ環境の実行時間
Fig. 4 Processing time in cluster environment

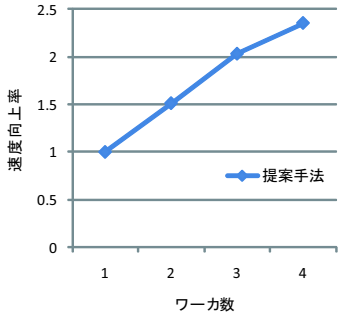

図 5 マルチコア環境の速度向上率
Fig. 5 Speedup in multicore environment

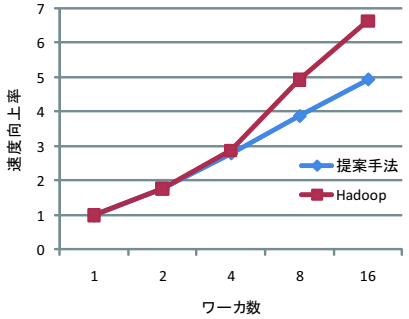

図 6 クラスタ環境の速度向上率
Fig. 6 Speedup in cluster environment

5 および図 6 にマルチコアおよびクラスタ環境それぞれの速度向上率を示す．

提案手法では，ワーカ数を増やすと同一の環境のワーカ数 1 に対しての実行時間はマルチコア環境で最大 57 %，クラスタ環境で最大 80 %の短縮が実現できている．ワーカ数に対する速度向上率を見ると，マルチコア環境ではワーカ数 4 のときで 2.3 倍，クラスタ環境ではワーカ数 16 のときで 4.9 倍となっている．また，実験した範囲内ではどちらの環境でもスケーラブルな性能向上となっている．

クラスタ環境において Hadoop と比較すると，ワーカ数 1 から 4 のときは提案手法のほうが実行時間が短いが，8 および 16 のときは Hadoop のほうが短い．また，提案手法の速度向上率はワーカ数 1 から 4 のときは Hadoop と同等だが，8 および 16 のときは Hadoop より小さい．この理由は，表 4 に示すように，提案手法のクラスタ環境での実行では，図 2 の F8 および F9 に一定量の時間がかかり，ボトルネックとなっていることが挙げられる．また，PyMW ではクラスタ環境の実行の際に，ワーカ数に応じて MPI の初期化時間が多くなるという理由もある．このため，これ以上ワーカ数を増やしても速度向上率は大きく増えないと予想できる．

F8 および F9 に一定量の時間を要するのは，分割処理時に全てのファイルを，行数を調べるために 1 度読み込むためである．今回の実験では入力データ，中間データの量は一定のため，ワーカ数に関わらず同じ処理時間を要する．

対して，Hadoop では HDFS を使用することにより，F8 および F9 を高速化している．

一方，提案手法のマルチコア環境では MPI を用いていないため，初期化時間はほぼ 0 秒である．ワーカ数 1 から 4 までの実験では，全体の処理時間のうち F8 および F9 が占める割合は 10 %以下である．実行時間の大部分を占めている Map および Reduce 時間はスケーラブルな性能向上が見られるため，さらにワーカ数を増やせば高速化できる可能性がある．ただし，マルチコア環境では Map 時間の速度向上率がクラスタ環境に比べて小さく，速度向上率が低い原因となっている．

以上の結果から，提案手法の性能については，マルチコア環境およびクラスタ環境共に理想的な速度向上は得られていない．

7. まとめと今後の課題

本研究では，ユーザが所有している環境で MapReduce プログラミングモデルを用いた計算を容易に実行可能とすることを目的として，PyMW を拡張して MapReduce を実装した．

本手法を用いて，単語カウントを行う MapReduce プログラムを作成し，マルチコア環境およびクラスタ環境でそれぞれ実行した．その結果，ユーザプログラムの環境指定部分を 1 行修正することで異なる環境での実行が可能であり，実行環境を意識しないプログラミングが実現できた．

各環境において，単語カウントプログラムを用いて性能を評価したところ，どちらの環境でもスケーラブルな性能向上が確認できた．ただし，速度向上率には改善の余地がある．

今後の課題として，ボトルネックとなっていた入力データ分割および中間データソート・分割処理を効率化することで，性能向上を目指す．また，本稿では実験していないグリッド環境での評価を考えている．

さらに，Google 社による実装と比べると提案手法の機能は部分的であるので，ユーザの利便性を向上させるために今回実装しなかった機能も実装する予定である．具体的には，タスク実行中にワーカに故障が発生した場合でも別ワーカで失われたタスクを実行する機能や，MapReduce 計算の最後に遅いワーカがタスクを実行している可能性を考えて別ワーカで同じタスクを並列実行する機能など，耐障害性を高めることを検討している．

謝辞 本研究の一部は科学研究費補助金（基盤研究 (A)2024002）の支援を受けた．

参 考 文 献

1) Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, *Communications of the ACM*, Vol.51, No.1, pp.107–113 (2008).
2) Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, Berkeley, CA, USA, USENIX Association, pp.10–10 (2004).
3) Hadoop: `http://hadoop.apache.org/`.
4) Chu, C.T., Kim, S.K., Lin, Y.A., Yu, Y., Bradski, G.R., Ng, A.Y. and Olukotun, K.: Map-Reduce for Machine Learning on Multicore, *NIPS*, MIT Press, pp.281–288 (2006).
5) Python: `http://www.python.org/`.
6) Heien, E.M., Kornafeld, A., Takata, Y. and Hagihara, K.: PyMW - a Python Module for Parallel Master Worker Computing, *The First International Conference on Parallel, Distributed and Grid Computing for Engineering* (2009).
7) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, *Proceedings of the nineteenth ACM symposium on Operating systems principles*, Vol.37, No.5, pp.29–43 (2003).
8) Amazon Elastic MapReduce: `http://aws.amazon.com/elasticmapreduce/`.
9) Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G. and Kozyrakis, C.: Evaluating MapReduce for Multi-core and Multiprocessor Systems, *HPCA '07: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, Washington, DC, USA, IEEE Computer Society, pp.13–24 (2007).
10) Yoo, R.M., Romano, A.J. and Kozyrakis, C.: Phoenix Rebirth: Scalable MapReduce on a NUMA System, *IISWC '09: Proceedings of the 2009 IEEE International Symposium on Workload Characterization*, Austin, TX, USA (2009).
11) 山下尊也，廣安知之，吉見真聡，三木光範，中尾昌広：PC クラスタ上のスケジューラを利用した MapReduce の実装，情報処理学会研究報告， Vol.2009-HPC-121, No.13 (2009).
12) 小川宏高，中田秀基，広渕崇宏，高野了成，工藤知宏：高速フラッシュメモリ向け MapReduce フレームワークの実現に向けて，情報処理学会研究報告， Vol.2009-HPC-121, No.42 (2009).
13) pyMPI: `http://pympi.sourceforge.net/`.
14) mpi4py: `http://mpi4py.scipy.org/`.
15) Parallel Python: `http://www.parallelpython.com/`.