DEIM Forum 2013 P2-4

Hadoopによる時系列画像からの時空間データマイニング

-植生指標の時空間モデリングを例として-

西前　光† 本田　理恵†

† 高知大学 { 大学院総合人間自然科学研究科理学専攻, 教育研究部自然科学系理学部門 }
〒 780-8082 高知県高知市曙町 2-5-1
E-mail: †{nishimae,honda}@is.kochi-u.ac.jp

あらまし 衛星による地球観測等の分野では大量の時系列画像が蓄積されており，ここから効率的にデータ抽出を行い機械学習等の手法を適用することによって有用な時空間変動パターンを発見する事が期待されている。我々はこうした時空間データマイニングのプラットフォームとして Hadoop を検討している。本論文では，全地球の植生指標 (GIMMS) データをテストベッドとし， 典型的な問題として，参照点の時系列データ抽出とこれに対する最尤法によるロジスティック関数のモデリングの 2 つを MapReduce で実装し，iMac50 台で構築した Hadoop システム上でそのスケール効果を確認したので，その結果について報告する。

キーワード 分散データマイニング，時空間モデリング，Hadoop MapReduce，時系列データ 植生指標 ロジスティック関数，最尤法

Distributed spatio-temporal data mining from time-series imagery by Hadoop

-application to spatio-temporal modeling of vegetation index-

Kou NISHIMAE† and Rie HONDA†

† Faculty of Information Science and Engineering, Kochi University
2-5-1 Akebonocho, Kochi-shi, Kochi, 780-8082 Japan
E-mail: †{nishimae,honda}@is.kochi-u.ac.jp

Key words distributed data-mining, spatio-temporal modeling, Hadoop, MapReduce, time-series data, vegetation index, logistic function, maximum-likelihood method

1. はじめに

近年諸分野でテラバイト級の大量のデータが蓄積されるようになり，こうしたデータから新しい知識を発見しようとするデータマイニングの研究が進んでいる。一方，クラウドコンピューティングの分野では，コモディティなコンピュータによる分散計算環境の実現が注目されており，そのミドルウェアとして，Apache Hadoop(以下 Hadoop) [1]，Gfarm [2] 等が開発されている。これらのミドルウェアの利用により，大学の計算機実習システムのような環境においても大量データからの知識発見を行うシステムを実装する事が可能になってきている [3]。

中でも，Hadoop は分散システムでの実行環境である MapReduce と分散ファイルシステムである HDFS (Hadoop Distributed File System) をその中に含み，さらに機械学習用ライブラリとして Mahout [4] を持つ事から，大規模分散データマイニングのプラットフォームとして有望と考えられる。ただし，MapReduce では，全てのデータを <key,value> の組として扱い，Map，Reduce の 2 ステップで処理する事によって効率的な並列処理を行うフレームワークをとっているため，従来はテキストや表などの単純なデータ形式を対象に検討される事が多かった。

大量の画像集合を扱う衛星画像処理や天文画像処理の分野での分散処理の導入例としては，白崎ら (2012) によるバーチャル天文台システムを想定してデータ分散検索・解析に Hadoop を導入する検討 [5] があるが，本格的なデータマイニングの検討までは進んでいない。

衛星画像の中でも，気象画像や植生指標等の地球環境のデータは，時系列画像としてアーカイブされることが多く，ここから時空間変動に関する知識を発見するには，問題に応じて，時間，空間断面を大量の画像集合から収集するという煩雑な前処理が必要となる。また，取り出したデータからのパターン発見には，逐次反復計算等の長時間の処理を要する事が多いため，これらのプロセスの効率化にも分散処理化が大きな効果を持つと考えられる。またこのような時系列画像処理の分散化は，監視画像や映像アーカイブなどへの応用性も高いと考えられる。

本研究では，時系列画像からの時空間データマイニングに対する分散処理システムを構築するために，地球環境データをテストベッドとして Hadoop を用いた分散処理環境を構築することを検討する。この過程で，データ選択やモデリングという時系列画像からの時空間データマイニングに特有な基本プロセスを実装して，その効果についての評価を行うものとする。

以降では，2 で Hadoop について簡単にレビューし，3 ではテストケースとして扱う時空間マイニングの問題とデータセットについて述べ，4 で構築するシステム，5 で実験条件とその結果，考察を述べ，6 で結論を述べる。

2. Hadoop

本章では Hadoop [6]，HDFS，MapReduce の特徴とそのデータマイニングに対する有用性についてまとめる。

Apache Hadoop Project は，信頼性の高いスケーラブルな分散コンピューティングの為の Open Source Software で，Hadoop common を中心とし，分散処理システムである MapReduce，分散ファイルシステムである Hadoop Distributed File System(以下 HDFS)，機械学習ライブラリである Mahout 等の様々なプロジェクトから構成されている。

HDFS は Google 社の The Google File System [7] に触発されて開発されたソフトウェアである。ストリーミング型のデータアクセスによって，非常に大きなファイルを保存するために設計されている。ファイルはあるブロックサイズ (通常 64MB) で分配され，それぞれ独立した単位として保存される。またこの時，耐障害性と可用性を上げる為，複数のマシン (通常 3 スレーブ) に分散して保存 (レプリケーション) される。

MapReduce は Google から発表されたプログラミングモデルのオープンソース実装である [8]。MapReduce における分散処理は，図 1 のように処理を Map，Shuffle，Reduce の 3 つの段階に分けて実行される。Map では入力データを分配し，それぞれの Map 関数を実行し，中間データを <key,value> のペアとして出力する。Shuffle では中間データの <key,value> を取得し，それぞれ同じ key ごとに value を集約，ソートを行い Reduce へ送る。Reduce では key と集約された value を取得し，key ごとに Reduce 関数を実行する。

Hadoop では，こうした機能によって，データのアーカイブと処理の両方の分散処理化を容易に実装できるようになっているため，大量データからの知識発見を扱うデータマイニングのプラットフォームとして適していると考えられる。

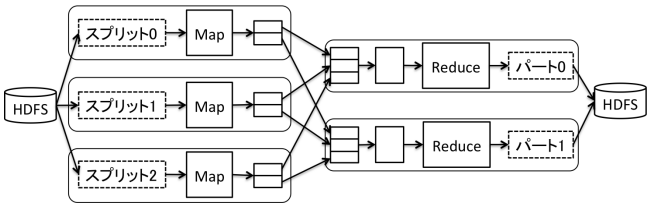

図 1　MapReduce の処理方法

3. 検討する問題とデータセット

時系列画像からの時空間データマイニングのケーススタディとしては，植生指標を格納した時系列画像から，ある参照点の時系列を抽出し，長期年間変動をロジスティック関数によって求めるという問題を扱う [9] [10]。

植生指標 (VI: Vegetation Index) は地球観測衛星で得られた可視のマルチバンド画像の特定の波長の観測値から計算される植物の量や活動度の指標である。一般的には VI は夏に増加し，冬に減少するという季節変動を示す。温帯での VI の季節変動の例を図 2 に示す。このようなパターンは，冬から夏，夏から冬の 2 つのロジスティック関数を年毎に接続した関数で表現できるものとすると，そのモデル F は以下の式で表される。

$$F(t_k|\theta_{i,j}) = \begin{cases} f_{i,1}(t_k|\theta_{i,j}), tb_{i-1} \leqq t_k < tt_i \\ f_{i,2}(t_k|\theta_{i,j}), tt_i \leqq t_k < tb_{i-1}, \end{cases} \tag{1}$$

where

$$f_{i,j}(t_k|\theta_{i,j}) = \frac{c_{i,j}}{1+\exp(a_{i,j}+b_{i,j}t_k)} + d_{i,j}, \tag{2}$$

$$\theta_{i,j} = (a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j}), \tag{3}$$

$$\boldsymbol{\theta} = \{\theta_{i,j}|i = 1, 2, \cdots, T, j = 1, 2\}. \tag{4}$$

ここで，t_k は時間，i は年数，j は季節のインデックス (1 が冬季から夏季，2 が夏季から冬季)，tb_i は $f_{i-1,2}$ と $f_{i,1}$ の交点，tt_i は $f_{i,1}$ と $f_{i,2}$ の交点を表す。$\theta_{i,j}$ は i 年の j 区間のロジスティック関数のモデルパラメータであり，T は時系列に含まれる年数を示す。

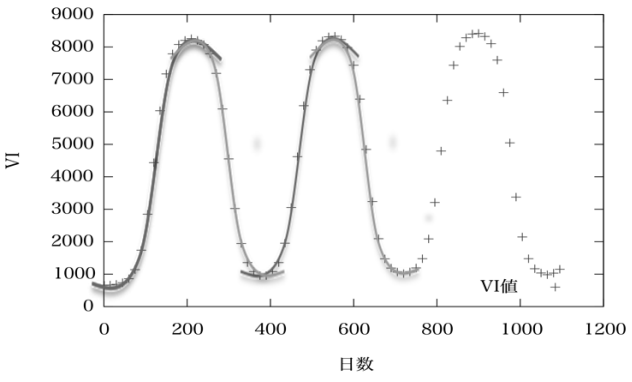

図 2　VI の変動パターンの例 (3 年間)

実際の観測データ $D = \{vi(t_k)|k = 1, 2, \cdots, n\}$ はノイズの

影響でモデルの周りに正規分布 $N(\mu, \sigma^2)$ で現れるとすると，尤度は次式で与えられる。

$$l(\boldsymbol{\theta}) = \log P(D|\boldsymbol{\theta}), \tag{5}$$

where

$$P(D|\boldsymbol{\theta}) = \prod_{i=1}^{T}\prod_{j=i}^{2}\prod_{k=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} \exp \frac{-\{vi(t_k) - F(t_k|\theta_{i,j})\}^2}{2\sigma^2} \tag{6}$$

この式に対し，最尤法によって観測値 D から尤度を最大化するモデルパラメータを求める。

$$\hat{\theta_{i,j}} = \arg_{\theta_{i,j}} \max l(\boldsymbol{\theta}) \tag{7}$$

モデルパラメータ $\theta_{i,j}$ は，式 7 をニュートン法によって反復計算することで求める事ができる [6]。

実験には実データとして GIMMS (Global Inventory Modeling and Mapping Studies) [11] を用いる。GIMMS は 1981 年 8 月から 2006 年 12 月までの 25 年 5ヶ月にわたって NOAA (National Oceanic Atmosphere Administration) 衛星の搭載センサである AVHRR (Advanced Very High Resolution Radiometer) のデータから作成された植生指標 NDVI (Normalized Difference Vegetation Index) を時系列画像としてアーカイブしたものである。GIMMS では半年毎の NDVI が 1 枚の画像として蓄積されているので分析にあたってはこれらの時系列画像から時間方向に画像を横断して必要なデータを収集することが必要になる。

4. システム構成

4.1 システムの概要

システムの概念図を図 3 に示す。まず時系列画像を Hadoop で構成したシステムの HDFS に格納することによってデータアクセスの分散化を行う。格納されたデータから，(1) 時間横断で特定の地点の時系列を抽出する。さらに (2) 抽出された時系列を最尤法によってモデルにフィッテングし，モデルパラメータを取得するものとする。このデータ抽出，モデルパラメータ決定という時空間データマイニングに特徴的な 2 つのプロセスの分散処理化，ならびにその効果の評価を行う。これらの過程の MapReduce での実装については次章で述べる。

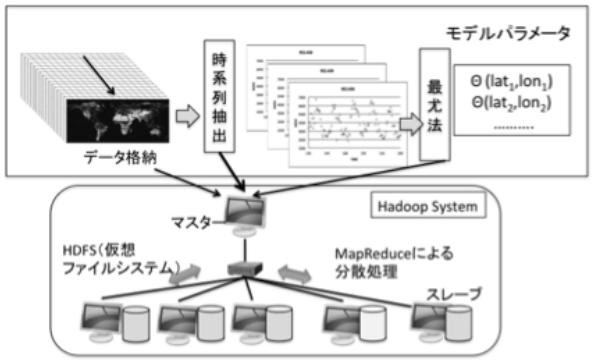

図 3 システムの概要

4.2 MapReduce によるコアプロセスの分散化

ここでは時系列抽出，モデルへのフィッティングのそれぞれに対して，MapReduce での実装方法を説明する。

4.2.1 時系列画像の時系列抽出

図 4 に，本研究で想定する時系列抽出の処理過程を示す。時刻 t の画像 における座標 $(x,\ y)$ の画像の階調値 (NDVI の値) を $vi_t(x, y)$ とおく。最初の Map のフェーズでは，key を空間座標 (x,y)，value を (データ ID, 時刻 t, 植生指標値 $vi_t(x, y)$) としてスプリットファイルを作成し出力する。Shuffle では座標を用いてスプリットファイルをまとめ，Reduce フェーズでは，座標ごとにデータ ID(時刻順) を用いて整列し，key を座標 $(x,\ y)$， value を植生指標値の時系列ベクトル $\boldsymbol{VI}(x, y) = \{vi_t(x, y)|t = 1, 2, \ldots\}$ として出力する。擬似コードを図 5 に示す。

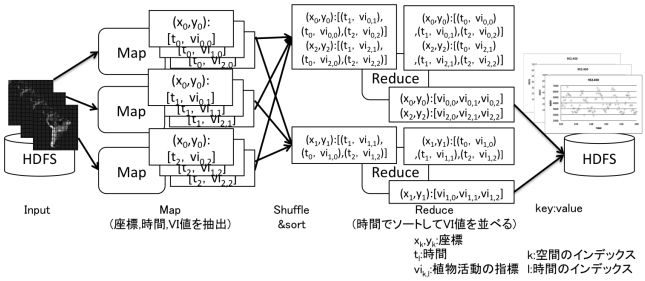

図 4 時系列抽出の MapReduce での実装概要

```
1: class MAPPER
2: method MAP(null, Image img)
3: t ← EXTRACT_TIME(img)
4: for all point p ∈ image do
5:     vi(p) ← GET_INTENSITY(image, p)
6:     EMIT(p, (t, vi(p)))
7: end for
8: class REDUCER
9: method REDUCE( p, D=[(t1, vi1(p)), (t2, vi2(p)), ···])
10: VI ← newLIST
11: for all (tk, vik(p)) ∈ D do
12:     VI(tk) ← vik(p))
13: end for
14: EMIT(p, VI)
```

図 5 時系列抽出の MapReduce 実装の擬似コード。ここでは簡単のために座標を $p = (x, y)$ としている。

4.2.2 最尤法によるモデリング

図 6 に最尤法によるモデリングの MapReduce による実装を示す。Map フェーズでは座標ごとのデータを読込み，key を座標 (x,y)，value を植生指標の時系列 $\boldsymbol{VI}$ として出力する (データ抽出と連続して行う場合はこのプロセスは不要)。Reduce フェーズでは，座標ごとに最尤法によりロジスティック関数へのフィッテングを行い，key を座標 (x,y)，value をロジスティック関数のパラメータとして出力する。なお，パラメータ数は年数 T に比例して増加し，パラメータ数の増加に応じて反復数

も増加するため，計算量は $O(T^2)$ となって好ましくない。このため，Reduce フェーズでは，25 年 (600 点) の時系列データを前後 2 年オーバーラップさせた 5 年ごとのデータに計算を区切って，5 年のデータに対し半年ごとにパラメータの推定を行った [10]。なお，擬似コードを図 7 に示しておく。

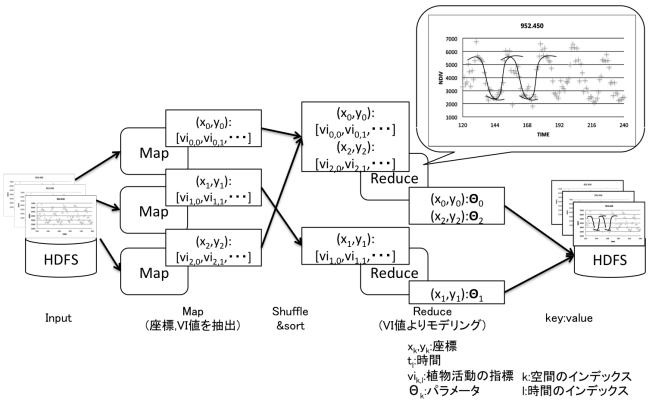

図 6　最尤法によるモデリングの MapReduce での実装概要

```
1: class MAP
2: method MAP(null, doc c)
3: for all line ∈ c do
4:     [p, VI] ←SPLIT(line)
5:     EMIT(p, VI)
6: end for
7: class REDUCE
8: method REDUCE(point p, VI)
9: θ ←ML_FITTING (VI)
10: EMIT(p, θ)
```

図 7　時系列抽出の MapReduce 実装の擬似コード

5. 実　　験

5.1 実験に用いたシステム

実験には高知大学情報科学教室の教育システムの iMac 51 台 (マスター 1 台，スレーブ 50 台) を利用した。マスター，スレーブの性能を表 1 に示す。マスター，スレーブマシンとも Mac OS X 10.6.8 をネットブートで起動する設定となっており，ユーザ管理は LDAP で行っている。このため，Hadoop アカウントを LDAP で作成し，Hadoop システムデータを NFS に格納することで，管理，設定を簡素化した。一方，各計算機の未使用のローカルストレージ (1 台あたり約 455GB) を HDFS 領域として利用する事により，ファイルアクセス・計算の効率的な分散化と 23TB の大容量を確保した。

実験に用いたシステムのネットワーク構成を図 8 に示す。各計算機は，2 台の L2 スイッチにスター型に接続されており，スイッチ間は 10Gbps で接続されている。

表 1　実験に用いる計算機 (マスター，スレーブ共通) の性能

諸元	値
台数	51 台
プロセッサ	Intel Core 2 Duo (3.06GHz)
コア数	2
メモリ	4GB
HDD	500GB
HDFS 用 HDD 容量	455GB(全システムで 23TB)
OS	Mac OS X 10.6.8
Hadoop version	1.0.3
Java version	1.6.0
ネットワーク	1000baseT

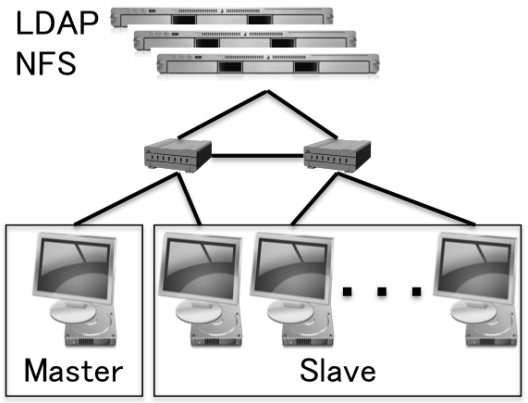

図 8　実験に用いたシステムのネットワーク構成

5.2 実 験 条 件

実験では (1) 時系列抽出と (2) モデリングの 2 種類のケースについて，スレーブの台数やコア数を変化させて速度向上比への影響を調べた。各ケースの実験条件を表 2 にまとめる。時系列抽出では 1152 × 1152 画素の 600 枚の画像すべての点を抽出するものとした。最尤法によるモデリングの実験では実データを全て計算するのは現実的でないため，1200 点のデータに対する実験をおこなった。このデータ数 1200 は，分配後のデータ数が等しくなるように各条件のスレーブ数の公倍数になるよう設定している (5.3 参照)。計算の終了条件については，計算機数，コア数でどの程度速度向上するかに焦点をおくため，同じデータを用いて終了条件を反復回数（固定）として実施した。スレーブの台数は 1, 10, 20, 30, 40, 50，各スレーブのコア数は 1 または 2 で変化させてそれぞれの問題の計算時間を測定した。また，HDFS のレプリケーション数は 3 とし，ブロックファイルサイズは 32MB とした。その他の map，reduce などのチューニングパラメータはデフォルト値をとっている。

表 2　実 験 条 件

条件	ケース	
	(1) 時系列抽出	(2) モデリング
データ	画像 1152*1152pixel (600 枚)	時系列データ (600 次元)
サンプル数	1327104	1200(注1)
スレーブ数	{1,10,20,30,40,50}	
コア数	{1,2}	

速度効率の効果の評価値としては，下記の式によって求められる計算速度向上比を利用した。

$$\text{計算速度向上比} = \frac{\text{各条件での実行時間}}{\text{1 台 1 コアでの実行時間}} \quad (8)$$

上式の正規化の基準値である 1 台 1 コアの計算時間を表 3 に示す。1 点あたりの実行時間はモデリングでは時系列抽出に比べると，4 桁大きい事がわかる。一方，(1)，(2) の問題では処理するデータ数が 10^5 倍も異なることにも注意する必要がある（表 2)。特に (2) のモデリングの問題ではデータ数 1200 点に対し，最大スレーブ数 100(50 台 2 コア) を使用すると 1 スレーブあたりのデータ数が 12 となり，Shuffle 時に Partitioner によってスレーブに分配されたデータ数にばらつきが生じることで処理時間に大きな違いが出る可能性がある。よって，次節では予備実験としてデフォルトの HashPartitioner によって各スレーブに分配されるデータ数を調べ，効果的なスケーリング効果の測定実験に備えるものとする。

表 3　1 台 1 コアの計算時間

条件	ケース	
	(1) 時系列抽出	(2) モデリング
全体の実行時間	5927sec	76608sec
1 点あたりの実行時間	4.47msec	65sec

5.3 予備実験

ここではデフォルトの HashPartitioner によるデータのスレーブへの分配数のばらつきを調べるため，時系列抽出，モデリングの両ケースについて，スレーブ数を 10(10 台各 1 コア) と 100(50 台各 2 コア）に対するデータ分配の予備実験を行った。各ケースで得られたスレーブへ分配されたデータ数のばらつきを表 4 に示す。この結果，デフォルトの HashPartitioner では，1 スレーブあたりの平均データ数が小さくなるにつれて分配数のばらつきが大きくなり，スレーブ数 100，平均データ数 12 というケースでは，スレーブへの分配数に標準偏差で平均値の 18%ものばらつきが生じてしまうことがわかる。

表 4　HashPartitioner を用いた場合の 1 スレーブあたりの分配点数。標準偏差は平均分配値で正規化。

case	スレーブ数	分配点数平均値	標準偏差（平均値で正規化）
(1) 抽出	10	132710	8.87×10^{-5}
	100	13271	1.97×10^{-5}
(2) モデリング	10	120	0.034
	100	12	0.183

図 9，図 10 に，特にデータ数の少ないモデリングでの 10 スレーブ，100 スレーブに対するデータ分配数のヒストグラムをそれぞれ示す。この結果から 10 スレーブのケース（スレーブあたりの平均分配数 120）では一様分布，100 スレーブのケース（スレーブあたりの平均分配数 12）では正規分布に近い分布となり，スレーブあたりのデータ数が少ない場合，HashPartitioner 関数を利用するとスレーブの増加による速度向上を正確に評価する事ができない可能性があることがわかる。よって以降の実験では，全てのスレーブに強制的にデータを等分配する Partitioner を作成してスケーリング効果の測定を行う事にした。また，比較の位置づけで HashPartioner を用いた場合の評価も行う事とした。

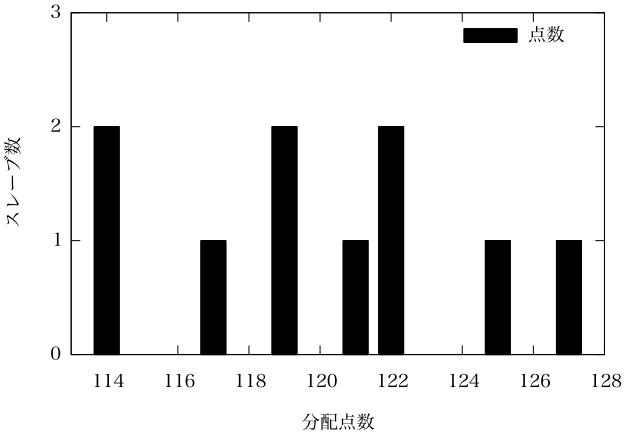

図 9　モデリングの計算時 10 スレーブの Reduce 時の各スレーブへの入力データ点数

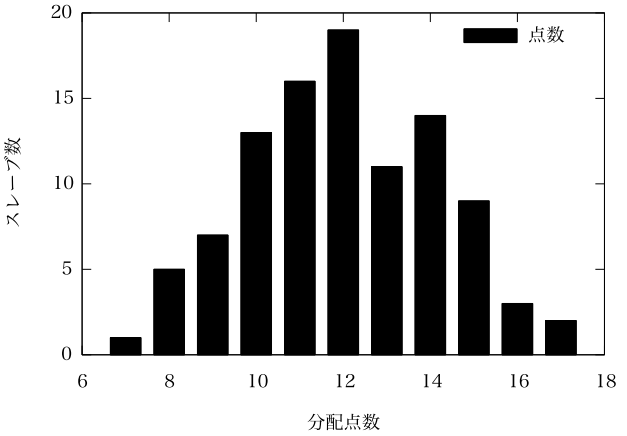

図 10　モデリングの計算時 100 スレーブの Reduce 時の各スレーブへの入力データ点数

5.4 実験結果

図 11 に時系列抽出のスケーリング効果に関する実験の結果として，コア数に対する計算速度向上比を示す。各計算機 1 コアのとき 20 台まではほぼコア数に比例して実質的な計算速度向上が達成されているが，30 台あたりより徐々に減少していく事がわかる。50 台各 1 コアでの計算速度向上比は 33 倍となった。また，各計算機のコア数を 2 に変化させてもあまり効果がない事がわかる。これら 2 つの結果より， HDD へのアクセスかネットワークの遅延がボトルネックとなっている事が考えられる。また，HashPartitioner を用いた場合はの結果と比較すると，本来，スレーブへの平均データ分配数が大きいこのケースでは等分配との違いは現れないはずであるが，特に 2 コアの場合は差異が生じていることがわかる。

(注1)：同じデータを複写。反復回数 10 万回

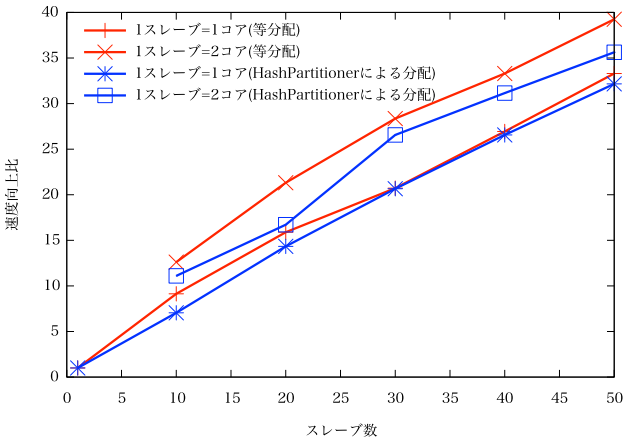

図 11　時系列抽出の計算速度向上比

図 12 に最尤法によるモデリングの速度向上比を示す。この結果からは，等分配の場合はほぼコア数に比例して計算速度向上比が増加している事がわかる。50 台各 2 コア (100 スレーブ）での計算速度向上比は 90.9 倍となった。またコア数を 2 倍にする事によって速度向上比もほぼ 2 倍となった。また，HashPartioner を使用した場合は，予備実験で予測した通り，等分配の場合と比べて速度向上比が低下した。Hash によるデータ入力点数の差が全体の計算時間に大きく影響を与えている事がわかる。また，図 12 の 40 台 2 コアのケースで大きく効率速度が低下しているが，これは分配数の偏りが影響していると考えられる。よって Partitoner に関しては問題に応じて注意深い調整が必要であることが確認され，以降のスケーリング効果の議論については等分配の実験結果に限って行うものとする。

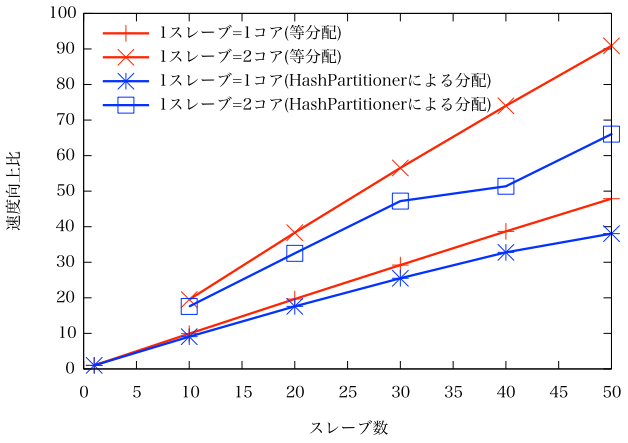

図 12　最尤法によるモデリングの計算速度向上比

5.5　考　　察

以上の結果より，時系列抽出では計算機を 40 台以上に増加させても効率が上がらなかった一方で，最尤法によるモデリングでは，台数（コア数）に比例して速度向上する事がわかった。よって，計算時間が長く，データアクセス頻度の低い問題では，Hadoop による分散処理による高速化の効率が高く，逆に 1 つの処理の単位時間が短く，データアクセス頻度の高い問題では計算機の台数やコア数を増加させても期待した効果が得られない可能性があるといえる。しかし，時系列抽出においても 30 倍程度の高速化が実現できたことと，MapReduce の利用によってデータの抽出過程が簡素化され，レプリケーションによってデータ保全の頑健性が確保できたことには着目すべきであり，Hadoop は大量の時系列画像を用いる時空間データマイニングには有用と期待される。

またこのテストケースに限ると，最尤法によるモデリングの実行時間が時系列画像の時系列抽出の実行時間に比べて 1 桁長いため，2 つのプロセスを接続した現実的なシステムにおける実質的な効率は，モデリングの過程の効率で支配されるため，分散処理による効果が十分期待できると考えられる。ただし，さらに規模の大きなシステムや，スレーブの一部がリモート環境に有るような場合についてはさらなる検討が必要である。

今後は，上記の遠隔サイトの問題に加え，統合的な時空間データマイニングシステムとして構築するために，mahout を用いた高次の知識抽出や，可視化，容易に試行錯誤を行う事ができるユーザインターフェースなどについて検討していく事が必要である。

6.　ま　と　め

時系列画像の時空間データマイニングの分散化を目的として，地球観測から得られた時系列画像からの時系列データの抽出とロジスティック関数による最尤法でのモデリングを Hadoop, MapReduce によって実装した。各 2 コアの iMac51 台のシステムで実験したところ，時系列抽出では計算機を 30 台付近で速度向上比が低下し，またコア数を増やしてもその効果が見られなかったのに対し，モデリングでは，計算機，コア数に比例して計算速度が向上した。

全体的なパフォーマンスとしては，最尤法によるモデリングの実行時間が時系列画像の時系列抽出の実行時間に比べて長いため十分な高速化を実現できる。またデータ抽出過程についても MapReduce の利用によって簡便に扱う事が可能となり，実験に用いたシステムでは 30 倍程度の高速化が実現できた。よって，Hadoop の利用は大量の時系列画像を用いる時空間データマイニングには有望であると考えられる。

文　　献

[1] Apache Hadoop Project, http://hadoop.apache.org
[2] Gfarm File System, http://datafarm.apgrid.org/index.ja.html
[3] 山邊 大樹, 奥村 勝, “教育用 PC システムと分散処理の共存に関する考察”，情報処理学会研究報告
[4] Apache Mahout Project, http://mahout.apache.org
[5] 白崎 裕治, 小宮 悠, 大石 雅寿, 水本 好彦, 石原 康秀, 堤 純平, 檜山 貴博, 中本 啓之, 坂本 道人, “JVO 開発における大規模天文データ処理： 全天対応天文データ分散検索・解析機構の試験構築”，宇宙航空研究開発機構研究開発報告 JAXA-RR-11-007,

pp57-66, 2012
[6] Tom White, 玉川 竜司, 兼田 聖士, “Hadoop”, O ’Reilly Japan, 2011.
[7] Sanjay Ghemawat, Howard Gobiof and Shun-Tak Leung, “The Google File System”, ACM SIGOPS Operating Systems Review, Vol.37, No.5, pp29-43, 2003.
[8] Jeffery Dean and Sanjay Ghemawat, “MapReduce Simplified Data Processing on Large Clusters”, OSDI, 2004.
[9] Rie Honda, “Temporal modeling and missing data estimation for Modis vegetation data”, The 2nd NASA data mining workshop , 2006
[10] 吉岡 和浩, “地球観測衛星による植生指標データからの長期年変動の抽出”, 高知大学卒業論文, 2008.
[11] GLCF Global Inventory Modeling and Mapping Studies, http://glcf.umiacs.umd.edu/data/gimms/