計算機アーキテクチャ 137－20
ハイパフォーマンス 80－20
コンピューティング
（2000．3．3）

PCクラスタを用いた決定木生成

久保田 和人† 仲瀬 明彦†
酒井 浩† 小柳 滋†

{kazuto, nakase, h-sakai, oyanagi}@isl.rdc.toshiba.co.jp

新情報処理開発機構 並列応用東芝研究室†

概要

数百ギガから数テラバイトクラスのデータに対するデータマイニングを実用時間で行えるシステムの構築を検討している．その知見を得るために，データマイニングの代表的な手法である決定木について，PCクラスタをターゲットとした並列アルゴリズムを実装し，予備評価を行った．本手法の特徴は，データを重複なく各プロセッサに分散配置すること，中間データをディスク上に置くことで大規模なデータを扱えること，処理手順を動的に決めることでディスクアクセスの軽減が図れることである．ベンチマークデータを用いた実験により，16台のプロセッサで11～24倍程度の高速化が実現された．また，オンメモリでは処理できない大規模なデータに対して動的に処理手順を変更する手法が有効であるという結果が得られた．

Parallelization of Decision Tree Algorithm on PC Cluster

KAZUTO KUBOTA†, AKIHIKO NAKASE†, HIROSHI SAKAI†
and SHIGERU OYANAGI†

Parallel Application Toshiba Laboratory
Real World Computing Partnership†

Abstract

We are planning to develop a practical data-mining system to the hundreds of Giga or Tera byte class data. In order to obtain the knowledge for the construction of the system, the decision tree which is the typical technique in data-mining is parallelized and implemented on a PC cluster. Our method has the following features. Input data are distributed to each processor element without overlap. Intermediate data are stored on disks, so that large size data can be executed. Disk access is decreased by dynamic execution order changing. Experimental results show that from 11 to 24 times acceleration is achieved by 16 processors, and dynamic execution order changing technique is effective for large scale data which cannot be solved on memory execution.

1. まえがき

流通業やサービス業をはじめとする多くの分野で，データマイニングと呼ばれる，計算機を用いて多量のデータから規則や特徴を抽出する手法が利用されている．例えば，金融機関では取り引きの減った顧客を分析することでその原因を探っている．スーパーマーケットやデパート等の小売業では，顧客の買物の内容や履歴の分析結果を営業戦略に役立てている．近年，データマイニングで扱われるデータサイズは増加の一途を辿っている．特にインターネット上のオンラインショッピングで生じるログファイルは膨大な量となっており，このような大規模なデータを精度良く高速に扱えるデータマイニング手法の確立が望まれている．

本稿では，データマイニングの重要な手法であるを決定木を取り上げ，処理の高速化，扱える問題の大規模化を図るために，並列アルゴリズムを構築し実装した結果について報告する．プラットフォームは，拡張性およびコストパフォーマンスが高いという点からPCクラスタとした．

アルゴリズムのベースとしては，Agrawalらが提案している並列決定木生成プログラムであるSPRINT[1]を用いた．この手法の処理手順に変更を加えることで高速化を試みている．決定木における枝の分岐を決定する評価基準としては，Quinlanらによって開発された決定木プログラムC4.5[2]で用いられている情報エントロピーを利用した．本手法は以下の3つの特徴を持つ．

- 入力データを重複なく各プロセッサに分散配置する．
- メモリに乗り切らない大規模なデータを扱える．
- 動的に処理手順を変更する．

本手法をベンチマークデータを用いて評価した．1千万レコード9属性，約400MBのデータを用いてPE(Processing Element)台数を1台から16台まで変化させた時の台数効果を調べた．また，1PE時において100万レコード9属性，約40MBのデータを用い，動的処理手順変更の効果を調べた．

本稿は，以下の構成をとる．第2節では，今回我々が構

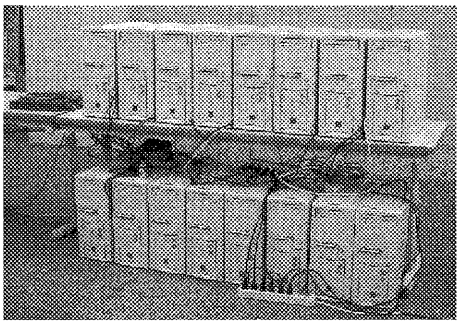


図1　PCクラスタ

表1　PCクラスタのノードの仕様

CPU	Dual Pentium-III 500MHz
Chip set	Intel 440GX
memory	PC100 SDRAM 256MB (up to 2GB)
HDD	IDE 28GB(7200rpm)
NIC	100 BASE-TX Ethernet

築したPCクラスタについて説明する．第3節では，決定木およびその構築のための基本アルゴリズムについて述べる．第4節では，今回ベースとしたSPRINTアルゴリズムについて説明する．第5節では，SPRINTをベースに我々が開発した決定木生成アルゴリズムおよび具体的な実装方法について説明する．第6節では実験結果を示し，第7節では今後の課題について述べる．第8節では関連研究について述べ，第9節ではまとめを行う．

2. データマイニングPCクラスタの構成

コストパフォーマンスおよび大規模なデータの取り扱いを考慮して，ノードPCの仕様を決定した．ノードPCの仕様を表1に示す．現在16台構成である．全体の写真を図1に示す．チップセットが440GXであるマザーボード，Tyan社製のS1837UANGを選択したので，メインメモリは各ノード当たり2GBまで拡張することが可能である．各PCは，24portのSwitching HUB(Cisco社製Catalyst3500)で接続されている．OSは，LINUX RedHat6.1(2.2.12kernel)を用いた．

3. 決定木生成基本アルゴリズム

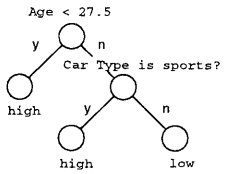
3.1 決定木とは

決定木は，データマイニングで用いられるclassificationアルゴリズムの一手法である．他の手法として，ニューラルネットを用いたものや，遺伝的アルゴリズムを用いたものがあるが，決定木は，これらの手法と比べて高速であり，同程度以上の質の解が得られることが知られている[2)3)]．

決定木では，トレーニングセットと呼ばれるデータ集合が与えられる(図2(a))．1つのレコードは，複数の属性`Age`，`Car Type`と1つのクラス`Risk`を持つ．属性はそのレコードを分類するための情報であり，クラスは分類先の情報である．属性は，カテゴリ値と呼ばれる離散値を取る場合(図2(a)の`Car Type`属性)と連続値をとる場合(図2(a)の`Age`属性)がある．決定木は，トレーニングセット中の各レコードに対して属性からクラスを決定するための木である．図2(b)は，図2(a)のトレーニングセットから生成された木である．

Age	Car Type	Risk
23	family	high
17	sports	high
43	sports	high
68	family	low
32	truck	low
20	family	high

(a)



(b)

図2　トレーニングセットと決定木(文献[1)]より)

```
FormTree( data )      /* node generation */
{
    EvalAtt( data );    /* evaluation */
    DivData( data );    /* data division */

    for each sub data i
        FormTree( subdata[i] );
}
```

図3　決定木生成アルゴリズム

ある．

決定木は分岐ノードと葉から成る．各分岐ノードには，レコードの属性を判定する条件式が与えられる．葉にはクラスが与えられる．あるレコードをルートノードから葉に対して適用することで，そのレコードのクラスが決定される．決定木は，テストセットと呼ばれるクラスを持たないデータのクラスを予測するのに用いられる．

3.2 基本アルゴリズム

決定木生成手法の基本的なアルゴリズムを図3に示す．これは，分割統治法に基づくものである．ノード生成(`FormTree`)では，トレーニングセットを分割した際の評価値が計算され(`EvalAtt`)，その中の最大の評価値をとる分割によってトレーニングセットが分割される(`DivData`)．分割されたデータは引数となり，再帰的に`FormTree`が呼ばれる．分割の種類は，カテゴリー値をとる場合は，すべてのカテゴリー値に分割する方法とグループに分割する方法がある．連続値をとる場合は，ある閾値を設けて2分割が行われる．評価値の計算方法は，情報エントロピーを基準とした方法[2)]や，gini index[4)]，カイ自乗検定を利用した方法[5)]などが考案されている．

4. SPRINT

SPRINTはAgrawalらによって開発された決定木生成アルゴリズムである．並列化することをを前提として設計されており，オンメモリでは処理できない大規模なデータを扱えるという特徴がある．IBMの並列計算機SP2上での性能が文献[1)]に示されている．ここでは，その逐次アルゴリズムおよび並列アルゴリズムを順に説明する．なお，詳細については，文献[1)]を参照されたい．

4.1 逐次アルゴリズム

アルゴリズムの概要を以下に示す．

- **step 1.** ファイルからデータ読み込む．各レコードは分解され，属性毎に属性リストが生成される．各リストの要素は，図4に示すような，属性の値(`var`)，クラス(`class`)，レコード番号(`id`)といった構造をとる．
- **step 2.** 連続値属性を持つ属性リストは属性の値によってソートされる．

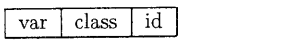


図4　属性リストの構造 (文献[1] より)

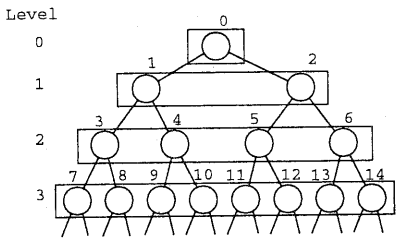


図5　SPRINTと処理の手順

- **step 3.** 全ての同じ深さ1にあるノードに対して，**step 3-1.** から **step 3-5.** の処理を行う．
  - **step 3-1.** ノードの分割方法を全て列挙し，それぞれの分割を行った場合の評価値を計算する．
  - **step 3-2.** 各ノード毎に **step 3-1.** で計算された評価値の中から最大のものを選ぶ．
  - **step 3-3.** **step 3-2.** で選択された分割方法でデータを分割した時に，各レコードはどのノードへ割り当てられるのかを計算し，変換テーブル (元論文[1] では probe structure と記述されている) を作成する．
  - **step 3-4.** 変換テーブルを用いて，全ての属性リストのノード番号を変更する．
  - **step 3-5.** 新たに作成されたノードのレコードが全て同じクラスを持つならば，そのノードを葉としクラスを与え，レコードを属性リストから削除する．属性リストが空ならば **step 5** へ．
- **step 4.** 深さ1を1増やし，**step 3** へ戻る．
- **step 5.** 終了．

SPRINT のアルゴリズム的特徴は，

- 同じ深さのノードは同時に計算される．
- 前処理で連続値を持つ属性を1度だけソートする．

というものである．SPRINT を用いた各ノードの処理の様子を図5に示す．この図で長方形で囲まれたノードは同時に処理される．C4.5 等の決定木生成システムではレコード単位でデータを管理していたため，各ノードで連続値の分割評価値計算が現れる毎にソートを行う必要があった．SPRINT では，データを属性リストで管理しているのでその必要がない．

4.2 並列版 SPRINT

入力データは PE 個に分割され，各 PE に配置される (図6)．逐次アルゴリズムと同様，各属性は属性リストと呼ばれる構造に変換される．連続値を持つ属性は全ての PE に渡ってソートされる．

処理手順は逐次版と同様に，評価値の計算，変換テーブルの作成，データ分割処理となる．評価値計算においては，連続値をとる属性に関しては，あらかじめ各 PE が保持するレコードのクラスの分布を調べておくことで，PE 独立に計算できる．離散値の場合は，各属性毎のクラスの分布を PE 毎に求め，全体で集計することになるので，同様に PE 独立な計算が可能となる．変換テーブルの作成は，各 PE がそれぞれテーブルの一部を作成し，全体で合成することでテーブルが完成する．データ分割は，変換テーブルを元に各 PE で独立に行うことができる．

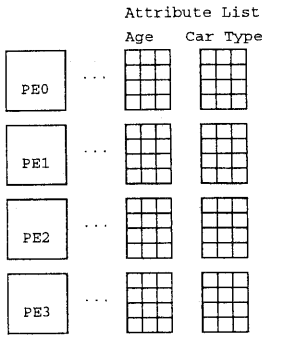


図6　並列 SPRINT におけるデータ配置

5. 並列版決定木の実装

今回我々が実装した並列版決定木について説明する．SPRINT をベースとしたアルゴリズムに，動的処理手順変更という方式を導入している．まず，逐次版 SPRINT アルゴリズムへの動的処理手順変更の導入について述べ，続いて並列版への導入について説明する．また，処理の高速化において問題となるオンメモリ処理とディスクアクセス処理について言及し，並列版決定木のデータ構造と実装について説明する．

5.1 動的処理手順変更処理

図5において，深さが同じノードは同時に処理されることを述べたが，これは，そのノード群すべてに含まれるデータを扱えるだけの記憶領域が必要になることを意味する．図5の例では，仮にルートノードの処理がオンメモリで行えないとすると，他の深さにおける処理も同様にオンメモリで行えないことになる．もし，記憶領域がメインメモリで足りない場合は，中間データをディスクに書き出す処理を，全ての深さのレベルで行わなければならない．

決定木においては，分割後のノードは依存関係がないため独立に処理が行える．したがって，各ノードの処理に必要なメモリサイズを考慮して，処理の手順を決定することで，ディスクアクセス処理を削減することができる．

逐次版の SPRINT アルゴリズムに動的処理手順変更処理を導入するため，前節で示した SPRINT アルゴリズムの二つの step を以下のように変更する．

- **step 3-4.** 変換テーブルを用いて，全ての属性リストのノード番号を変更する．**各ノードに含まれるレコード数から，その処理に必要なメモリサイズを計算する．この大きさがプラットフォームのメモリサイズを下回る場合は，ノードキューと呼ばれるキューにレコードを退避し，データ構造から削除する．**
- **step 5.** **ノードキューからレコードを取り出し，順次オンメモリで決定木を作成する．**

図7は，処理手順変更後のアルゴリズムによる木の生成手順である．この図において，白抜きの長方形で囲まれたノードの処理はディスクアクセスを伴う処理であり，斜線の長方形のノード処理はオンメモリで行える処理である．深さ2のノードにおいて，ノード3とノード6はノードキューに退避され，ノード4と5はディスクアクセスを伴う処理が

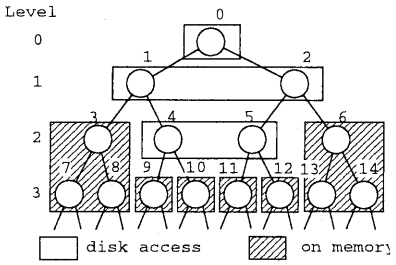


図7　SPRINTと処理手順の変更

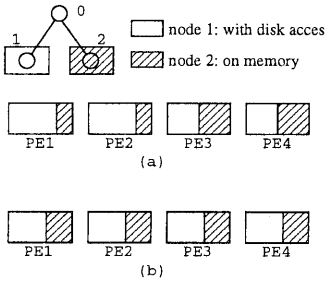


図8　ノードデータの分布

行われる．深さ3のノードにおいて，9~12のノードはオンメモリで処理できる大きさになるので，すべてのノードはキューに退避され，後から順次処理される．オリジナルのアルゴリズムでは，全てのノード処理がディスクアクセスを伴うことになるが，変更後のアルゴリズムではディスクアクセスを伴う処理を5ノードに減らすことができる．

5.2　並列版SPRINTへの動的処理手順変更処理の組み込み

SPRINTの並列版に対しても，同様に上記の処理手順を組み込むことができる．あるノードの処理に必要なデータサイズが全PEのメモリサイズ以下になった場合に，その処理はノードキューに登録できる．逐次処理と違って並列処理で問題となるのは，並列処理の場合，各ノードのデータが全てのPE上に不均一に分散されている可能性があるということである．図8(a)に例を示す．ルートノードの処理後に，二つのノードが生成された状況である．ノード1は，全PEのメモリを用いてもオンメモリでは処理できないが，ノード2はできるものする．これを，そのままのPEへのデータの割り当てで別々に処理したとすると，それぞれのノードの処理ではデータが均等にPE上に分散していないため台数効果が期待できなくなる．これを避けるためには，各ノードのデータをPE上に均等に分散させる必要がある(図8(b))．しかしながら，この状況を実現するためにはデータの移動が必要となる．どちらを選ぶかは両者のオーバーヘッドを比較する必要がある．

ノードキューに溜ったノードの処理手順の決め方に関しても，いくつかの方法が考えられる．処理の順番に依存関係はないため，どの順番で処理してもよく，また，複数のノードを組み合わせて同時に処理してもよい．これらのノード(群)を解くPE数は，オンメモリ処理になる範囲で自由に設定できる．一方，それぞれのノードのデータは各PEに分散しているため，トータルの性能をあげるためには再配置を行わなければならないかも知れない．ここでもデータ転送オーバーヘッドを考慮する必要がある．

表2　(データ構造のメモリ上への保持方法)

属性リスト	全部持つ，1属性分，1属性の一部分
変換表	全部持つ，一部持つ

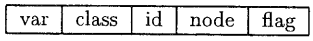


図9　属性リストの構造

5.3　オンメモリ処理とディスクアクセス処理

提案アルゴリズムでは，属性リストと変換テーブルが大規模な記憶容量を必要とするデータ構造である．これらのデータ構造がメモリに入り切らない場合は，一部をディスクに書き出す必要がある．メモリ上に何を残すかという戦略は，様々な方法がある(表2)．どの方法が優れているかを定性的に判断することは難しく，プラットフォームとの兼ね合いもある．ここでは，パラメータを変化させることでこれらの条件が設定できるようにプログラムを実装する．

5.4　データ構造とインプリメント

データの一部がディスクに持たれる可能性があることを考慮し，属性リストの構造を図4から図9のように変更する．追加されたnodeは，現在そのレコードが属しているノード番号を表す．idがあれば変換テーブルからノード番号を引くことができるが，変換テーブルがディスク上にある可能性もあるので追加した．flagは，nodeフィールド書き換え処理のときに使われる．実プログラムでは，リストを構成する構造体は16byteである．

変換テーブルを用いた属性リスト更新処理は，高速化およびデータの大規模を実現する上で並列決定木生成の核となる部分である．変換テーブルの保持方法は様々なバリエーションが考えられるが，ここでは全PEで重複して持たれる場合について説明する．決定木の分割方法が決まると，各PEの該当する属性リストが持つidに関して変換テーブルが更新される．図10(a)は4PEが重複して変換テーブルを持った状態を示しており，黒い部分が自分のPE処理で更新された部分である．PE0では，変換テーブルのid=1,4の部分が更新され，PE1では，id=0,7の部分が更新されている．PE間では重なりはない．この変更部分を全てのPE間で反映させ合うことによって変換テーブルが完成する(図10(b))．

全PEで重複して変換テーブルを持った場合，全レコード分の変換テーブルを1PEで保持しなければならなくなるため，扱えるデータ数にスケーラビリティがなくなる．変換テーブルをPE毎に分割すれば，1PEで扱うべきデータ量は$1/PE$となるが，データ分割処理は変換テーブル全体を参照する必要があるので，各PEに分散されたテーブルを順次シフトしながら処理を行うなどの必要が出て来る．図10(c)では，各PEが持つ変換テーブルの部分を全PEに巡回させることで変換テーブルの更新処理を実現した例である．初期状態(step 0)では，PE1,2,3,4がそれぞれ，id=0,1, id=2,3, id=4,5, id=6,7の変換テーブルを持っている．各PEの持つテーブルを巡回させることでテーブルの更新を行っている．データ分割処理では，再びテーブルを巡回させることで各PEの持つ属性リストのノード番号を更新することになる．

一方，データ分割処理において，変換テーブルは各PE上に分散させておき，必要な情報を必要なときに聞きに行くというアプローチも考えられる．この方法では，処理の手

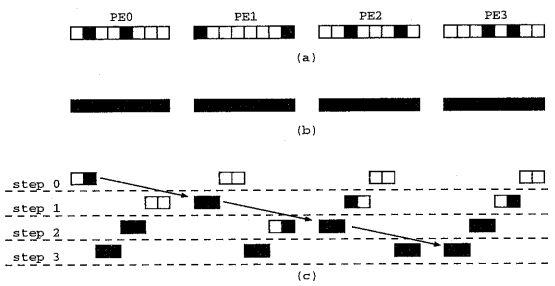


図 10　変換テーブル更新処理

表 3　処理時間 (sec)

PE 数	木生成 (a)	変換テーブル集計 (b)	(c):(a)-(b)	全体 (d)
1	14600	0	14600	18569
2	7120	88	7032	9430
4	3738	168	3570	5120
8	2139	248	1891	3041
16	1301	329	971	1958
16(on memory)	611	329	282	1271

間は $1/PE$ となる．細粒度の高速な通信機能，できればリモートリード，ライトのような機能があれば，この方法も実用的な性能を示す可能性がある．

なお，現在のインプリメントでは，変換テーブルを全 PE で重複して持つ方法を採用しており，各 PE の変換テーブルの情報を全 PE に反映させる処理は，更新されていない変換テーブルの部分をゼロにしておき，リダクション加算を行うことで実現している．

6. 性能評価

現在，並列版決定木の基本部分の実装が終わり，動的処理手順の組み込みを行っている段階である．ここでは，並列版決定木の現段階での台数効果について調べ，動的処理手順変更処理については，1PE で処理手順を変更した結果からその有効性を検証する．

6.1 台数効果

10M レコード 9 属性，大きさ約 400MB のベンチマークデータを用いた．使用したプラットフォームは，第 2 節で述べた 16 台構成の PC クラスタである．今回はシングル CPU のみ使用している．通信ライブラリには MPI(mpich 1.2.0) を利用した．コンパイラは gcc を用い，オプション `-O3 -Wall` でコンパイルした．

メモリの使用方法は，1 属性リストのみメモリにおけるものとした．なお，今回のデータサイズでは，16PE に分割した場合 9 属性全てをメモリ上に持つことが出来るので，16PE に関しては属性リスト全てをメモリ上に持った場合の結果も計測した．入力データは，あらかじめ分割して各 PE のローカルディスクに配置して各 PE が読み込むようにした．プログラム全体，木を作る部分，変換テーブル集計処理時間を測定した．結果を表 3および図 11に示す．なお，木は深さ 10 で処理を打ち切っている．

木の作成については，ディスクアクセスを伴う処理の場合 16PE で約 11 倍の高速化となっている．図 12に示すように変換テーブル集計処理の割合が PE 台数が増えるにしたがって増加している．この処理を除いた場合 (c) について

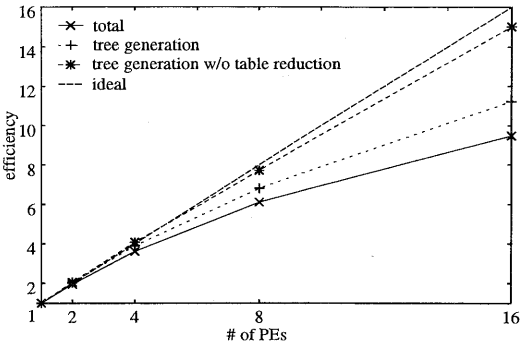


図 11　台数効果

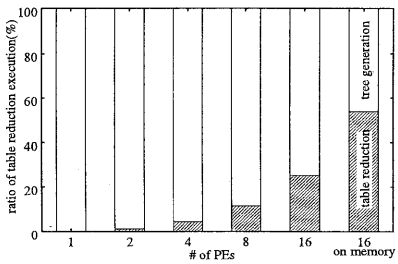


図 12　変換テーブル集計処理の割合

考えると，8 台で 7.7 倍，16 台では 15 倍程度の高速化が達成されており，メインの計算部分について本プログラムはほぼスケールアップしているといえる．全体の処理時間 (d) で考えると，16 台の場合 9.5 倍程度の速度向上である．16 台をオンメモリで処理した場合，処理時間がディスクアクセスを伴う場合と比べて大幅に短縮され，1 台と比較した場合の性能は，木の生成で 24 倍，処理全体でも 14.6 倍となっている．変換テーブル集計処理を除いた場合は 52 倍の性能向上を示している．

6.2 処理手順変更による効果

データの保持方法を途中で変更した場合に，処理がどの程度の高速化されるのかを 1PE の場合について計測した．データは，1M レコード 9 属性で約 40MB のデータを用いた．データ保持方法の変更方法は以下の 2 通りである．

- 方法 1:1 属性リストと変換テーブルをメモリ上に半分づつしか持てない状態から両者をメモリ上に持てる状態へと変化させた場合．
- 方法 2:1 属性リストしかメモリ上へ持てない状態から，全属性リストと変換テーブルをメモリ上に持てる状態へと変化させた場合．

このデータから生成される木の各ノードの持つデータサイズを図 13 に示す．各深さにおいて大きい順にソートしてあり，一番大きいサイズは斜線の長方形で示してある．途中からトータルのデータ数が減っているのは，葉に割り当てられたデータがあるためである．今回は，ある深さにおいて全ノードのデータ数が 500K 以下になった場合 (図 13では深さ 2) に処理を切替えるものとした．計測にあたっては，深さ 2 までと 3 以降の処理を別々のプログラムで行い，両者の合計時間を全体の時間とした．したがって，データの選別，移動等の切替え時に必要なオーバーヘッドは含まれて

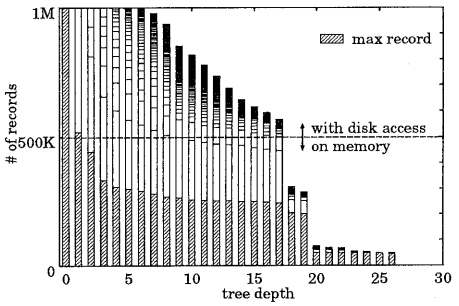


図 13　木が分割される様子

表 4　処理手順変更による処理時間の変化 (sec)

	変更前	変更後
方法 1	2413	1057
方法 2	1574	412

いない．深さ 15 のところで処理を打ち切っている．結果を表 4に示す．方法 1 では 2.3 倍，方法 2 では 3.8 倍程度の高速化が図られた．なお，本来ならば方法 2 の変更前は方法 1 の変更後より処理時間が小さいことが予測されるが，実際は逆になっている．これは，キャッシュ等の影響によるものと考えている．

7. 今後の課題

現在は，基本的なアルゴリズムの実装および予備評価が終った段階である．将来の高速化に向けて処理結果をブレークダウンし，アルゴリズムおよびプログラムの詳細を把握する必要がある．現在の PC クラスタは，ネットワークに 100Base-TX を仕様しており，通信ライブラリは mpich1.2.0 を利用している．PE 台数の増加に伴い，リダクション処理の影響が出て来ているので，例えば myrinet 等を使うなどの通信機能の高性能化により処理時間がどの程度改善されるかを検討する必要がある．

高速化のもう一つのポイントとしては，処理のオーバーラップが挙げられる．本プログラムに導入可能なオーバーラップ処理としては，ファイルの入出力と計算のオーバーラップおよび通信と計算のオーバーラップがある．これらのテクニックを利用することで，本手法はさらに高速化できる可能性がある．

我々が構築した PC クラスタは，コストパフォーマンスを考慮して Dual CPU のものを採用した．この 2CPU を利用する方法として，1PE に 2 プロセスを走らせる並列化手法が考えられるが，今回の我々のプログラムはメモリを多く消費するため効果が上がらないことが予想される．処理の内容から，スレッドプログラミング等を利用した並列化が効果があるのではないかと考えている．

また，逐次処理で効果が確認できた処理手順変更手法を並列版の決定木生成に導入して，その効果を確認する必要がある．

8. 関連研究

逐次版の決定木アルゴリズムとして Cart[4]，および C4.5 の前身である ID3[6] がある．Cart と C4.5 は，それぞれ IND-Cart，IND-C4 へと改良されている [7]．また，Cart を並列化したものに，Darwin がある．並列版の決定木アルゴリズムとしては，SPRINT の他に Agrawal らは SLIQ[8]，を発表している．また，SPRINT を SMP 向けに並列化した研究も行われている [9]．

我々は C4.5 を SMP 向けに並列化した並列決定木システムを構築し評価を行っている [10]．本稿は，並列版の決定木手法をインプリメントして性能評価を行なったものである．メモリサイズを考慮した動的処理手順の変更を提案し，その効果を調べている．

9. ま と め

本報告では，データマイニングで用いられる決定木アルゴリズムを，PC クラスターゲットとして並列化した．SPRINT という決定木アルゴリズムをベースとして並列版の決定木プログラムを実装した．また，動的な処理手順変更アルゴリズムを提案した．ベンチマークデータを用いた実験により，並列版決定木プログラムでは，16PE を用いた場合，11 倍から 24 倍程度の高速化が実現された．また，動的処理手順変更手法に関しては，1PE を使った結果から効果が期待できることが確認された．

今後は，「今後の課題」で述べた内容に取り組み，将来的には，数百ギカ～数テラバイトクラスのデータを実用的な処理時間で扱うことのできるシステムへと発展させていくことを考えている．

参 考 文 献

1) John Shafer, Rakesh Agrawal, and Manish Mehta, "SPRINT: A Scalable Parallel Classifier for Data Mining," Proc. of the 22nd VLDB Conf. 1996.
2) J. Ross Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufman, 1993.
3) Rakech Agrawal, Tomass Imielinski, and Arun Swami, "Database mining: A performance perspective," IEEE Trans. on Knowledge and Data Engineering, 5(6):pp.914–925, Dec. 1993.
4) L.Breiman, J.H.Freidman, R.A.Olshen, and C.J.Stone," Classification and Regression Trees," Wadsworth, Belmont, 1984.
5) 福田 剛志, 森本 康彦, 徳山 豪, "多値属性を用いた最適なデータセグメンテーションを生成するアルゴリズム", 信学技報, DE98-22, pp.83–91, Oct. 1998.
6) J. Ross Quinlan, Induction of decision trees. Machine Learning, 1:pp.81–106, 1986.
7) NASA Ames Research Center, Introduction to IND Version 2.1, GA23-2475-02 edition, 1992.
8) Manish Mehta, Rakesh Agrawal, and Jorma Rissanen, "SLIQ: A fast scalable classifier for data mining," Proc. of the Fifth Int'l Conf. on Extending Database Technology, March, 1996.
9) Mohammed Javeed Zaki, Ching-Tien Ho, Rakesh Agrawal, "Scalable Parallel Classification for Data Mining on Shared-Memory Multiprocessors," IEEE ICDE, pp.198–205, March 1999.
10) 久保田 和人, 仲瀬 昭彦, 酒井 浩, 小柳 滋, "決定木の並列化とその評価", 信学技報, 99-HPC-77(SWoPP'99), pp.161-166, 1999.