ハイパフォーマンス
コンピューティング 87－11
（２００１．７．２６）

決定木生成手法の並列化方式とその評価

久保田　和人† 仲瀬 明彦† 小柳 滋†

{kazuto, nakase, 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

決定木生成手法と呼ばれるデータマイニング手法を PC クラスタ上に実装し評価した。効率の良い並列決定木生成手法として、Agrawal らによる SPRINT と Kumar らによる ScalParC が知られている。これら二つの手法は、全データを管理する管理テーブルの保持の仕方が異なる。前者はテーブル全体を全てのプロセッサで重複して持ち、後者はテーブルをプロセッサ間で分割して保持する。我々はこれら二つの手法をインプリメントしベンチマークデータを用いてメモリ使用量、処理時間の面から比較した。その結果、処理速度、メモリ使用量の両面においてプロセッサ台数が少ないときはテーブルを重複して保持する方法が有利で、プロセッサ台数が多い場合はテーブルを分割して保持する方法が有利だと分かった。また、プラットフォームの通信性能を向上させると両手法の処理性能のクロスポイントが変化することもわかった。

Performance Evaluation of Data Management in Parallel Decision Tree Generation

KAZUTO KUBOTA†, AKIHIKO NAKASE† and SHIGERU OYANAGI†

Parallel Application Toshiba Laboratory Real World Computing Partnership†

Abstract

The implementation of the parallel decision tree construction algorithm on a PC cluster are described and evaluated. SPRINT by Agrawal et al. and ScalParC by Kumar et al. are known as efficient parallel decision tree generation algorithms. Both of these algorithms use a data management table, but they manipulate the table in different ways. We implemented these two methods on a PC cluster and evaluated them by using a benchmark data. The result shows that processing speed and memory usage is better in the former method when the number of processors is small, but the latter method achieves better performance with large number of processors. We have also clarified that high speed network changes the intersecting point of the execution speed of two methods.

1. まえがき

本稿では、データマイニング手法の代表的手法の一つである決定木生成手法を取り上げる。決定木生成はデータベース中のレコードを分類するクラス分類手法の一つである。効率の良い並列決定木生成手法としては、Agrawal らによる SPRINT アルゴリズム [1] が知られている。SPRINT は、入力データを各 PE (Processing Element) で分割して保持するが、処理の途中で生成される管理テーブルは全 PE で重複して保持されるので記憶容量に関するスケーラビリティがない。処理速度面では、SPRINT は管理テーブルの生成には PE 間通信が必要であるが、参照には PE 間通信が必要ないので処理の手間が小さい。SPRINT では、この管理テーブルを probe structure と呼んでいるが、以下では変換テーブルと呼ぶことにする。

一方、SPRINT をベースとした並列決定木生成手法として Kumar らによる ScalParC[4] がある。ScalParC は、入力データと変換テーブルの両方を各 PE で分割して保持する。したがって、ScalParC は記憶容量に関するにスケーラビリティがある。しかし、変換テーブルの生成、参照ともに PE 間通信が必要となるので処理の手間が大きい。以下では、SPRINT 方式の変換テーブル管理方法をテーブル重複方式、ScalParC 方式の変換テーブル管理方法をテーブル分散方式と呼ぶことにする。

プロセッサ台数が小さい時は処理の手間の小さいテーブル重複方式の方法が高速であると考えられる。プロセッサ台数が大きい時は、どちらの方法が高速であるのかを一概に結論づけることはできず、その結果は PE 台数やプラットフォームの通信性能および入力したデータの特徴に依存すると考えられる。両手法の特徴を定量的に明らかにすることは、大規模なデータに対するマイニングを行っているユーザやそのためのプラットフォームの導入を検討しているユーザにとって有益な情報となりうる。

本稿では、両手法を PC クラスタをターゲットとして実装し、PE 数を 1 台から 16 台まで変化させ、様々な特徴を持つベンチマークデータを用いた際の処理時間およびメモリ使用量を比較した。また、プラットフォームのネットワーク性能が変化した場合の両手法の処理時間の変化の予測を行っ

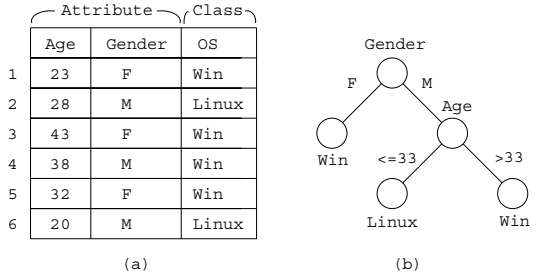


図 1 トレーニングセットと決定木。(a) トレーニングセット。(b) 決定木。

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

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

図 2 決定木生成アルゴリズム

た。なお、ここでは SPRINT と ScalParC を直接比較したのではなく、両者をベースとしたプログラムを開発し、変換テーブルに関する実装を二種類行い、それらを比較した。

2. 決定木と基本アルゴリズム

2.1 決定木とは

決定木生成は、データマイニングで用いられる代表的な手法であり、クラス分類と呼ばれる手法の一つである。入力としてトレーニングセットと呼ばれるレコードの集合が与えられる (図 1(a))。レコードは、複数の属性 (Attribute) と 1 つのクラス (Class) を持つ。属性は、連続値をとる場合 (Age) とカテゴリ値と呼ばれる離散値を取る場合 (Gender) がある。図 1(b) は、図 1(a) のトレーニングセットから生成された決定木である。決定木はノードと葉から成る。各ノードには分岐条件が与えられ、葉にはクラスが与えられる。

決定木は、テストセットと呼ばれるクラスを持たないレコードのクラス値を属性値から予測するために用いられる。テストセット中の個々レコードに対して、決定木の分岐条件がルートから葉方向に向かって適用され、個々のレコードはいずれかの葉へと分類される。分類された葉のクラス値が予測値となる。

2.2 基本アルゴリズム

決定木生成の基本アルゴリズムを図 2 に示す。これは分割統治法に基づくものである。FormTree は再帰関数であり、一回の呼び出しが決定木の一つのノードに関する処理となる。FormTree は、分割条件の計算 (EvalAtt) とデータ分割処理 (DivData) からなる。

EvalAtt では、トレーニングセットを様々な分割の方法で分割した際の評価値が計算され、その中の最大のものが分割方法として採用される。候補となる分割方法としては、カテゴリー属性の場合は、全てのカテゴリーに分割する方法や 2 分割、多分割する方法が候補となる。連続属性の場合は、データを 2 分割する全ての分割方法が候補となる。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、カイ自乗検定を利用した方法などが考案されている。

図 3 属性リストの構造の変更

DivData では、EvalAtt で求まった分割方法を用いてトレーニングセットが分割される。分割されたデータが全て同じクラスを持つ場合、あるいはノード内のデータが、ある基準を満たした場合にはノードは葉となりクラスのラベルが付加される。それ以外のデータに対しては、再帰的に FormTree が適用される。

3. SPRINT、ScalParC の基本アルゴリズム

ここでは、SPRINT および ScalParC が共通して採用している基本アルゴリズムを我々の実装をベースとして説明する。SPRINT および ScalParC は変換テーブルの操作以外、ほぼ同様のアルゴリズムを採用している。詳細に関しては、原著 [1)4)] を参照されたい。

SPRINT や ScalParC では属性リストと呼ばれるデータ構造が用いられており、これは属性の値 (Val)、クラス (Class)、レコード番号 (Id) という構造をとる。我々の実装では、この構造体にノード番号を表すフィールド (Node) を追加している (図 3)。これは、データの分割を実際のデータの移動ではなく、属性リストの Node フィールドの書き換えで行うためである。

変換テーブルは、SPRINT や ScalParC で属性リストを管理するために利用されるテーブルを我々が単純に実装したものである。変換テーブルは一次元の配列であり、インデックスがレコード番号を、値がノード番号を示す。

SPRINT はノードの生成順序に関しては規定がないが、我々の実装では並列化時の負荷分散の均等化を考慮して同一の深さのノードは同時に処理されるものとした。

逐次アルゴリズムは以下の通りである。

- **Step 1.** ファイルからデータ読み込む。属性毎に属性リストが生成される。

図 4(a) は、図 1(a) のトレーニングセットから生成された属性リストである。初期状態では全てのレコードがルートノードに属しているので、Node フィールドは 0 となっている。

- **Step 2.** 連続値属性を持つ属性リスト (この例では Age 属性リスト) は属性の値によってソートされる (図 4(b))。
- **Step 3.** 全ての同じ深さにあるノードは一つのグループにまとめられ、一括して Step 3-1 から Step 3-5 の処理が行われる。
  - **Step 3-1.** それぞれのノードに関して全ての分割候補が列挙され、評価値が計算される。
  - **Step 3-2.** Step 3-1 で計算された評価値の中で最大値をとる分割方法がノード毎に選ばれる。
  - **Step 3-3. (変換テーブル生成)** Step 3-2 で選択された分割方法を用いて、各レコードの分割先のノードを計算する。結果は変換テーブルへと格納される。

ルートノードの分割条件として $Gender = F$ が選ばれたとする。Gender 属性リスト (分割属性リストと呼ぶ) は、この条件を用いて更新される。すなわち、$Gender = F$ のレコードには、ノード番号 1 が書き込まれ、$Gender = M$ のレコードには、ノード番号 2 が書き込まれる (図 5(a))。続いて、ノード番号更新後の分割属性リストを用いて変換テーブルが生成される (図 5(b))。

  - **Step 3-4. (データ分割)** 分割属性リスト以外の属性リストのレコードの Node フィールドを、変換テー

Age	Class	Id	Node
23	Win	1	0
28	Linux	2	0
43	Win	3	0
38	Win	4	0
32	Win	5	0
20	Linux	6	0

Gender	Class	Id	Node
F	Win	1	0
M	Linux	2	0
F	Win	3	0
M	Win	4	0
F	Win	5	0
M	Linux	6	0

(a)

Age	Class	Id	Node
20	Linux	6	0
23	Win	1	0
28	Linux	2	0
32	Win	5	0
38	Win	4	0
43	Win	3	0

Gender	Class	Id	Node
F	Win	1	0
M	Linux	2	0
F	Win	3	0
M	Win	4	0
F	Win	5	0
M	Linux	6	0

(b)

図 4 図 1(a) のテストセットに関する属性リスト。(a) ソート前、(b) ソート後.

Gender	Class	Id	Node
F	Win	1	1
M	Linux	2	2
F	Win	3	1
M	Win	4	2
F	Win	5	1
M	Linux	6	2

(a)

Trans.	Tbl
1	1
2	2
3	1
4	2
5	1
6	2

(b)

図 5 変換テーブルの生成。(a)Node 番号の割り当て。(b) 変換テーブルの生成。

Age	Class	Id	Node
20	Linux	6	2
23	Win	1	1
28	Linux	2	2
32	Win	5	1
38	Win	4	2
43	Win	3	1

図 6 属性リストの更新。

ブルを用いて更新する。

Age 属性リストは、図 5(b) の変換テーブルを用いることで、図 6 のように更新される。

- **Step 3-5.** 新しく生成されたノード中のレコードが全て同じクラスを持つか、あるいは、ある基準を満たす場合は、そのノードを葉としてクラスを与え、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.** 処理深さのレベルを 1 増やし、Step 3 へ戻る.
- **Step 5.** 終了.

上記アルゴリズムの並列化に際しては、個々の属性リストがレコード数が均等になるように分割され各 PE で保持される。処理手順は逐次版と同様であり、評価値の計算およびデータの分割が全体で同期をとりながら行われる。並列アルゴリズムの詳細に関しては、文献 [1)4)] を参照されたい。

4. 変換テーブルの実装

第 3 節で示したアルゴリズムを並列化する際には、変換テーブルの保持方法としてテーブル重複方式とテーブル分散方式が考えられる。ここでは両方式について説明する。

4.1 テーブル重複方式

第 3.1 節で示したアルゴリズムの Step 3-3、3-4 の処理手順は以下の通りに並列化される。

- **Step 3-3. (変換テーブル生成)**
  - **Step 3-3-1.** 各 PE において、求まった分割条件と分割属性リストを用いて個々のレコードの分類先のノードを求める。
  - **Step 3-3-2.** 各 PE において、Step 3-3-1 求まった値を変換テーブルに反映させる。
  - **Step 3-3-3.** 各 PE の変換テーブルを合成することによって変換テーブル全体を完成させる。変換テーブルは、テーブル全体が全 PE で重複して保持される。
- **Step 3-4. (データ分割)**
  - 各 PE において、変換テーブルを参照しながら各属性リストの Node フィールドを更新する。

全ての PE が変換テーブル全体を保持しているので、属性リスト更新時には PE 間通信は必要ない。

以下では、変換テーブルの操作と木の生成手順を例を用いて説明する。図 7(a) はトレーニングセットを表し、8 レコード、2 属性を持っている。図 7(b) はルートノードの処理が終わりノード 1, 2 が生成された段階の木と各ノードに属するレコードの番号を示している。このときの属性リストは、図 7(c) のようになる。ここで評価値計算が行われ、ノード 1 に関しては $Attr\#1 < 14$ が、ノード 2 に関しては $Attr\#2 < 3.3$ が、それぞれ分岐条件として選ばれたとする。この条件を用いて変換テーブルの一部 (図 8(a)) を個々の PE 上に生成する。これは、Attr#1 属性リスト中で $Node = 1$ であるレコードについて、$var < 14$ のレコードにはノード番号 3 を $var \geq 14$ のものには 4 を割り当て、Attr#2 属性リスト中で $Node = 2$ であるレコードについて、$var < 3.3$ のレコードにはノード番号 5 を $var \geq 3.3$ のものにはノード番号 6 を割り当てることで行う。個々の PE で生成された変換テーブルを全体でマージすることで、図 8(b) に示した全レコードに対する変換テーブルが生成される。これは、例えば、テーブルの空の部分を 0 にしておき、リダクション加算することで行える。この時点で各 PE はテーブル全体を保持しているので、属性リストの Node フィールドの更新はローカルに行える。更新後の属性リストおよび決定木は図 8(c),(d) のようになる。

4.2 テーブル分散方式

第 3.1 節で示したアルゴリズムの Step 3-3, Step 3-4 の処理は以下の通りに実現される。

- **Step 3-3. (変換テーブル生成)**
  - **Step 3-3-1** 求まった分割条件を用いて、PE 毎に個々のレコード (レコード番号を id とする) がどのノード (ノード番号を node とする) に分類されるのかを求め (id,node) という二つ組を作る。
  - **Step 3-3-2** 変換テーブルの id の部分を保持する PE を求め (id,node) を送信する。
  - **Step 3-3-3** 各 PE で、送られてきた (id,node) を用いて変換テーブルの id の部分に node の値を書き込む。
- **Step 3-4. (データ分割)** 属性リスト毎に以下の処理を行う。
  - **Step 3-4-1** 各 PE において、属性リストの個々の要素のレコード番号 id を調べ、変換テーブルの id の部分を保持する PE を求める。(id,?) という二つ組を作成し、求まった PE に送信する。
  - **Step 3-4-2** (id,?) を受け取った PE では、変換テーブルを用いて id から node を求め、(id,?) を (id,node)

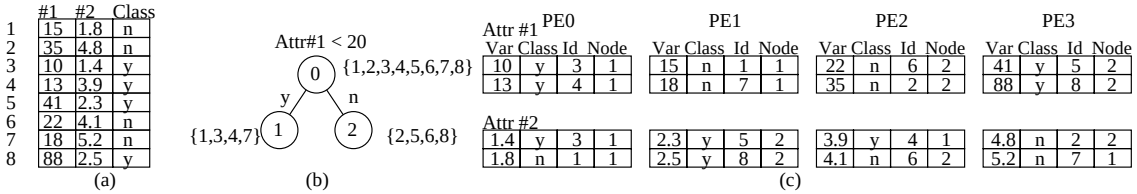


図 7 変換テーブルの操作とノード生成処理 (初期状態)

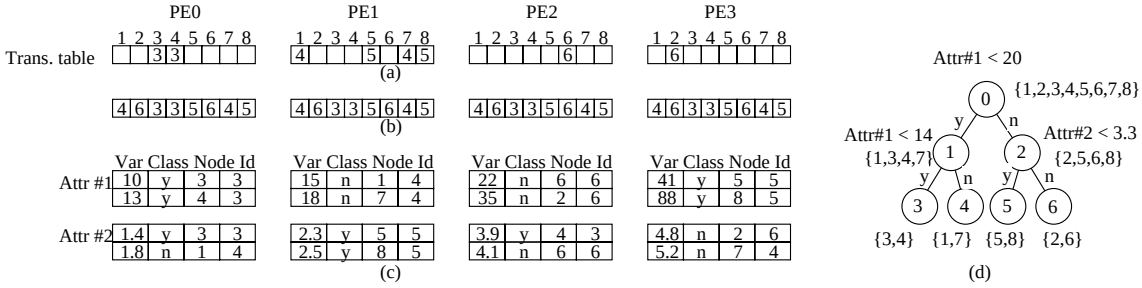


図 8 テーブル重複方式による変換テーブルの操作とノード生成処理

に書き換える。(id,node) を送信元の PE に送り返す。

– **Step 3-4-3** 送信元では送り返された (id,node) を用いてレコード番号が id である属性リストの要素の Node フィールドを node とする。

以下では図 9 を用いてテーブル分散方式の処理手順の例を示す。図 7 に引き続いて図 9 の処理が行われるものとする。分岐条件も図 8 で用いたものと同様であるとする。図 7(c) において、分岐条件決定後、PE 0 が保持する属性リストからは、レコード 3 と 4 がノード 3 へと分割されることがわかる。よって、図 9(a) に示すように、PE 0 に関しては (3,3)、(4,3) という二つ組が生成される。図 9(a) の making initial tuples の部分に、各 PE で求められたノード分割のため二つ組を示す。つづいて、これらを変換テーブルの対象部分を持つ PE へと送信する。変換テーブル送信後の各 PE 上の二つ組は図 9(a) の Data exchange に示す通りになる。送付後の二つ組を用いて、各 PE 上の変換テーブルは図 9(b) のように更新される。

次に図 9(c) を用いてデータ分割の手順を示す。ここでは Attr#1 のみ例を示すが、残りの属性リストに関しても同様の処理を行うことになる。各 PE で Attr#1 属性リストから、(id,node) の組を作る (Making initial tuples)。Node フィールドは、まだ決まっていないため?が入っている。この二つ組を変換テーブルの id 番目の要素を持つプロセッサに送信する (Data exchange 1)。受け取った PE では、送られてきた二つ組の id 番号を参照し、対応するノード番号を?の部分に書き込む (Fill node field)。その後、この二つ組を送信元の PE へと送り返す (Data exchange 2)。送り返された情報を用いて個々の PE で属性リストの Node フィールドを更新する (図 9(d))。送信は個々の二つ組毎に行うのではなくバッファリングして行う。したがって、一連の作業で用いる PE 間通信は全対全通信となる。

5. 性能評価

我々は、第 3 節、第 4 節で示したアルゴリズムを PC クラスタをターゲットとして実装した。変換テーブルに関する処理は二種類の手法が切替え可能である。ここでは、ベンチマークデータを用いて両手法の比較を行う。また、PC クラスタのネットワーク性能が変化したときの両者の優位性の変化について予測する。

表 1 PC クラスタのノードの仕様

CPU	Dual Pentium-III 800 MHz
Chip set	ServerWorks HE-SL
memory	PC100 SDRAM 1 GB (up to 4 GB)
HDD	IDE 45 GB (7200 rpm)
NIC	100 BASE-TX Ethernet

5.1 実験条件

実験に用いた PC クラスタは 16 台構成である。ノード PC の仕様を表 1 に示す。 各 PC は、48 port の Switching HUB(Cisco 社製 Catalyst3500) で接続されている。OS は、LINUX RedHat6.2(2.2.19kernel) を用いた。なお、本システムでは 1 ノードに 2 CPU を持っているが、今回行った実験では 1 CPU のみ使用している。

プログラムは C 言語を用いて記述し、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 でコンパイルした。入力データは、あらかじめ前処理を行い各 PE のローカルディスクに配置した。前処理はソート処理が大部分を占め、この部分の高速化が処理全体に与える影響は大きい. しかし、現在は単純な実装しか行っていないため今回は前処理に関する議論は見送り、以下では木の生成時間について着目する。

ベンチマークデータとしては、文献 [3] に示されている synthetic database の F7 を用した。このベンチマークデータでは、レコードサイズや属性数を自由に設定することができる。ここでは、レコード数 × 属性数が 80M となる 4 通りの組合せのデータを使用した (表 2)。これらのデータに対し、PE 台数を 1, 2, 4, 8, 16 と変化させ、テーブル重複版とテーブル分散版の実行時間とメモリ使用量を計測した。実行にあたっては、変換テーブルおよび計算中の属性リストはメモリ上に置き、それ以外の属性リストはディスク上に退避しておくことにした。

また、高速なネットワークを仮定した時の木生成の処理時間を予測した。高性能な通信装置としては、ギガビットイー

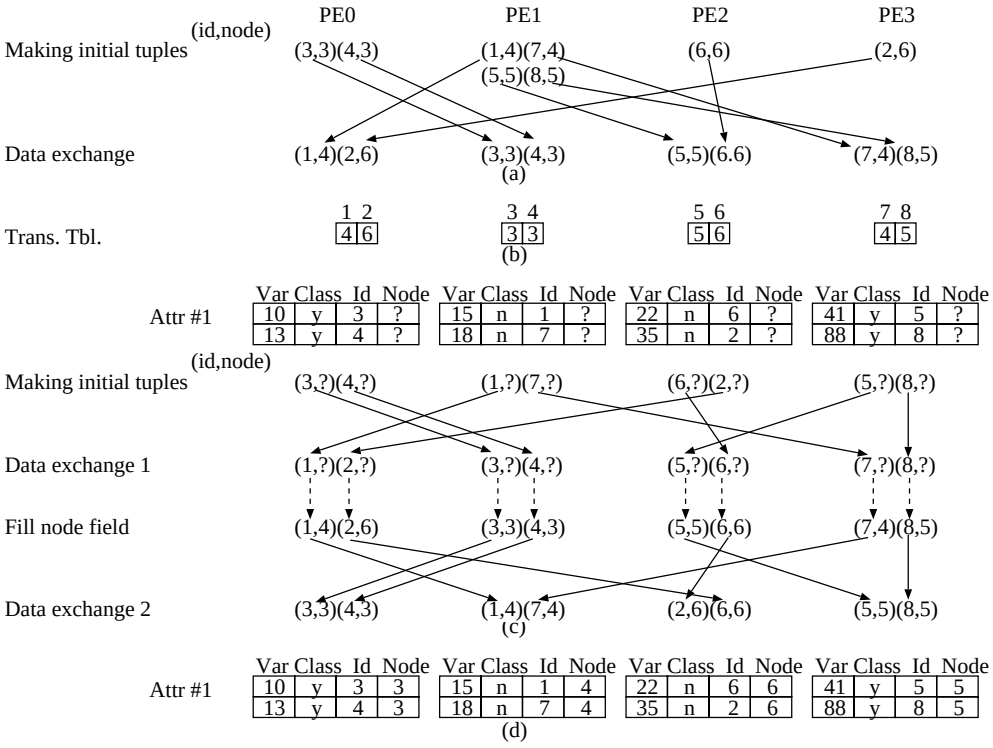


図 9 テーブル分散方式による変換テーブルの操作とノード生成処理

表 2 使用データのサイズおよび属性数

レコード数	8M	4M	2M	1M
属性数	10	20	40	80

サーや myrinet がある。これらの通信装置は 100 Base-T と比較して約 10 倍のバンド幅を持つ。今回の決定木生成手法で用いた通信は、長いメッセージ通信が主であり、バンド幅が 10 倍になれば通信時間は 1/10 になると仮定できる。したがって、100 Base-T 使用時の実行時間を計算部分と通信部分にブレークダウンし、高速ネットワーク使用時には、通信部分の処理時間が 1/10 になると仮定して処理時間を算出した。

5.2 評価結果

図 10 において左側の列が木生成の処理時間であり、真中の列が使用メモリサイズを示している。これは、各 PE で使用した最大メモリサイズの合計である。右側の列は、高速ネットワーク使用時の処理時間の予測値をグラフ化したものである。

5.2.1 処理時間

殆んどのパラメータにおいてテーブル重複版が高速であるという結果が得られた。これは、テーブル分散版における変換テーブルの操作処理が複雑であり、特にデータ分割の際に PE 間通信を伴うためである。4M レコード 20 属性で PE 数が 16 の時、および、8M レコード 10 属性で PE 数が 8 以上の時はテーブル分散版が有利であった。すなわち、レコード数が多く、属性数が少ないデータではテーブル分散版が有利であった。これは、属性数が少ないとテーブル重複版の変換テーブル生成処理のオーバーヘッドがテーブル分散版のデータ分割処理のオーバーヘッドと比べて相対的に大きくなるためである。

5.2.2 メモリ使用量

レコードサイズが小さくなるとメモリ使用量も減っている。これは、今回の実験では処理中の属性リストのみメモリ上に保持しているためである。PE 数が小さい時はテーブル重複方式の方がメモリ使用量が小さく、PE 数が大きい時はテーブル分散方式の方がメモリ使用量が小さい。これは、PE 数が小さい時はテーブル分散方式では変換テーブルの生成、参照に使うバッファ領域がメモリを消費するためである。また、PE 数が大きい時はテーブル重複方式では変換テーブルの重複分がメモリを消費するためである。

5.2.3 通信性能の影響

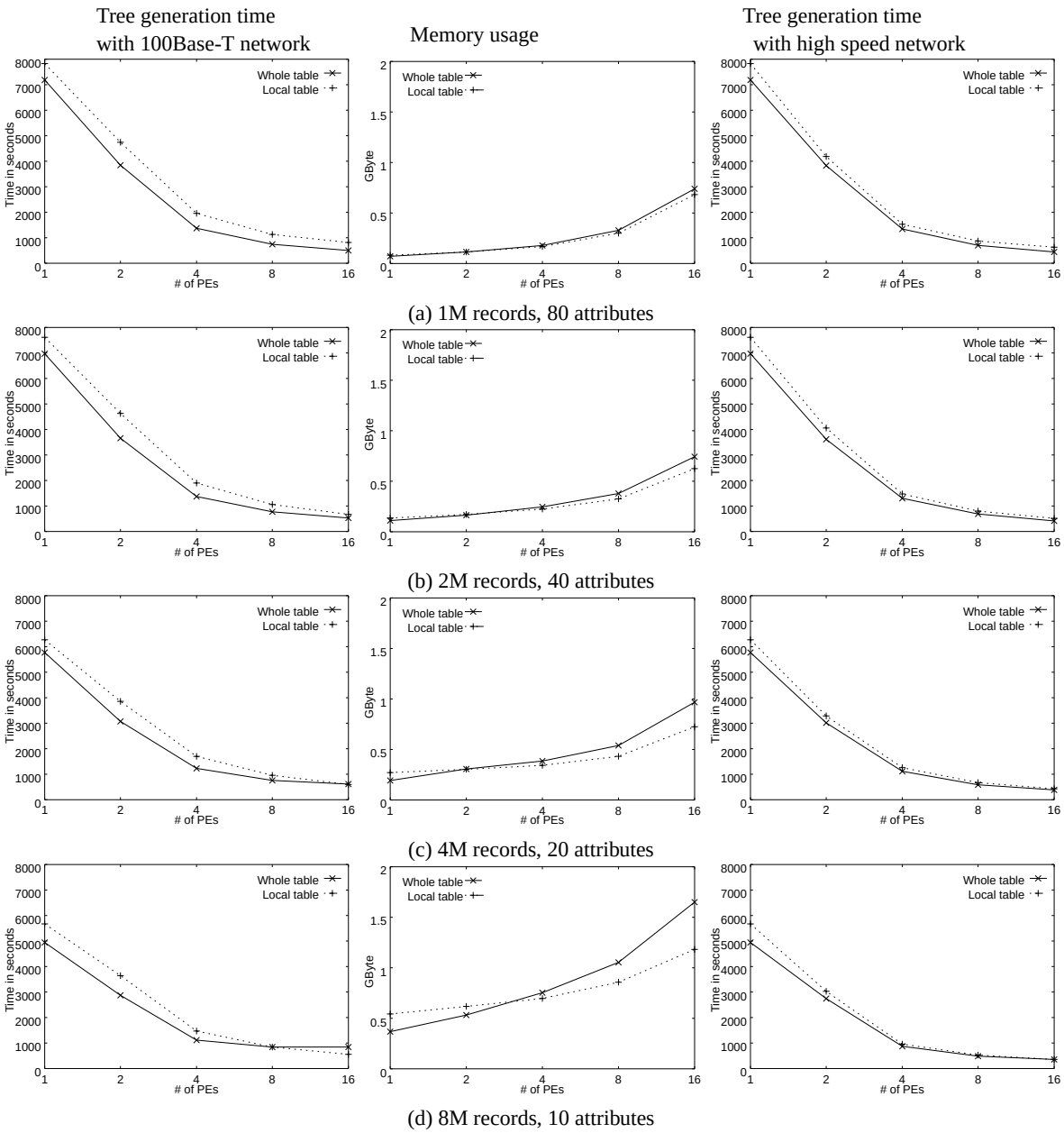
処理時間の基本的な傾向は 100Base-T を用いた場合と同じであるが、高速ネットワークの使用を仮定した場合、テーブル重複方式とテーブル分散方式の処理時間の差は小さくなっている。これは、両方式ともに処理時間に含まれる通信時間が大きいためである。今回の予測値では、8M レコード 10 属性、16PE の時のみテーブル分散版の処理速度がテーブル重複版を上回っている。すなわち、100Base-T を利用した場合と比べて、両手法の性能の逆転ポイントが PE 台数が大きい方向にシフトしたと言える。

5.3 テーブル重複方式と分散方式の比較

今回の実験から、二つの手法はどちらかが優れているというわけではなく、扱うデータの性質、PE 数、プラットフォームのメモリサイズ、通信性能によって優劣が分かれるということがわかった。実験結果をまとめると表 3 のようになる。

6. まとめ

本稿では、PC クラスタをターゲットとする並列データマイニングについて二種類の実装方法を比較、評価した。ベンチマークデータを用いた実験により、両手法の優劣を一意



(a) 1M records, 80 attributes

(b) 2M records, 40 attributes

(c) 4M records, 20 attributes

(d) 8M records, 10 attributes

図 10 性能評価

表 3 テーブル重複方式と分散方式の比較

	テーブル重複版	テーブル分散版
属性数	大きい時に有利	大きい時は不利
PE 数	小さい時に有利	大きい時に有利
メモリ使用量	PE 数小の時有利	PE 数大の時有利
通信性能	高い時、両者の性能差は縮まりクロス ポイントは PE 数の大きい方にシフトする	

に決めることはできず、扱うデータの性質、PE 数、プラットフォームの通信性能によって優劣が分かれるということがわかった。

今後は、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の構築を考えている。その際は大規模な PE 数のクラスタの利用が必須であると考えられるため、テーブル分散方式の方が有利であろう。

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. Classification and Regression Trees,” Wadsworth, Belmont, 1984.
4) Mahesh V. Joshi and George Karypis and Vipin Kumar, “A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets,” 1998 International Parallel Processing Symposium, 1998.