PC クラスタ上における頻出飽和パターン抽出並列化手法の提案

岩橋　永悟[†]　　平手　勇宇[†]　　山名　早人[††,†††]

† 早稲田大学大学院理工学研究科　〒169-8555 東京都新宿区大久保 3-4-1
†† 早稲田大学院理工学術院　〒169-8555 東京都新宿区大久保 3-4-1
††† 国立情報学研究所　〒101-8430 東京都千代田区一ツ橋 2-1-2
E-mail: †{eigo,hirate}@yama.info.waseda.ac.jp, ††yamana@waseda.jp

あらまし　データマイニング分野における頻出パターン抽出問題では，大規模なデータに対して処理を行うため，メモリ容量不足やディスクアクセス増加といった問題に直面する．このようなリソース面の制約を緩め，現実時間で頻出パターンを抽出するために，様々な並列化手法が提案されている．しかし，従来の並列化手法の多くは全ての頻出パターンを抽出するため，結果として莫大な数のパターンが抽出されてしまう．そこで，本稿では，FPclose に基づき頻出飽和アイテム集合を並列抽出する手法を提案する．さらに，並列化において問題となる，タスク負荷の偏りを平坦化する手法を提案する．提案した手法を PC クラスタ上で実装し，評価を行った．本手法を 32 ノード PC クラスタで実行した結果，最小サポートが 2% の場合 30.9 倍の速度向上を得ることができた．また，データの大規模化に対応しうるスケーラビリティを得ることができた．

キーワード　データマイニング，頻出パターン抽出，PC クラスタ，並列化

Parallel Closed Frequent Pattern Mining on PC Cluster

Eigo IWAHASHI[†], Yuu HIRATE[†], and Hayato YAMANA[††,†††]

† School of Science and Enginnering, Waseda Universityhskip1em Okubo 3-4-1, Shinjuku-ku, Tokyo, 169-8555 Japan
†† Science and Enginnering, Waseda Universityhskip1em Okubo 3-4-1, Shinjuku-ku, Tokyo, 169-8555 Japan
††† National Institute of Informatics 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430 Japan
E-mail: †{eigo,hirate}@yama.info.waseda.ac.jp, ††yamana@waseda.jp

Abstract　Frequent patterns mining is one of the important problem in data mining research. Since frequent pattern mining processes very huge data, frequent pattern mining faces the lack of memory spaces or the increase of disk access. For the purpose of mining frequent patterns in real time by lowering such resource constraints, various parallel algorithms are proposed. However, since many of traditional parallel algorithms mine all frequent patterns, a large number of patterns are mined as a result. In this paper, we propose the parallel closed frequent pattern mining method besed of the FP-growth algorithm. In addition, we propose the load balancing method, which is indispensable to parallel methods. As a result of the evaluation using 32 node PC cluster, our method is approximately 12 times faster than sequential FPclose, when minimum support is 1.0%. In addition, our method copes with data scalability.

Key words　Data Mining, Frequent Pattern Mining, PC Cluster, Parallelization

1. は じ め に

近年，ネットワーク環境の整備，記憶装置の低価格化が進むにつれて，大量のデータを蓄積することが可能となっている．しかし，データは記号の列にすぎず，データから情報を見出すのは本来ユーザの役割である．そこで，無秩序に集められた大規模なデータから，有用な知識を抽出するデータマイニング技術が注目されている．

データマイニング技術における重要な手法の一つとして，頻出パターン抽出がある．頻出パターンは，購買データから併売パターンを抽出するバスケット解析，Web ログからユーザの行動パターンを抽出する Web ログ解析，DNA 解析などに利用される．頻出パターンを抽出するためには，大規模なデータに対して処理を行うため，効率よく頻出パターンを抽出する手法の

研究が行われている．

頻出パターン抽出手法としては，Apriori [9] や FP-growth [5] が有名である．これらの手法では，抽出されるパターン数が膨大となり，結果が冗長になってしまうという問題がある．この問題点を解決するために，パターンの冗長性を削減した極大頻出パターンや飽和頻出パターンを抽出する手法 [6] [2] が提案されている．

一方で，頻出パターン抽出では大規模なデータに対して処理を行うため，メモリ容量不足やディスクアクセス増加といった問題に直面する．このようなリソース面の制約を緩め，現実時間で頻出パターンを抽出するために，PC クラスタなどをターゲットとした並列化手法が提案されている．

従来提案されてきた並列化手法の多くは，Apriori をベースとする手法である [10] [12]．最近では，FP-growth をベースとする並列化手法も提案されている [4]．しかし，これらの並列化手法では，全ての頻出パターンを抽出するため，結果として莫大な数のパターンが抽出され，ユーザに負担が掛かるという問題がある．また，頻出パターン並列抽出処理においては，トランザクションデータベース中に出現するアイテムの特性により，ノード毎でタスク負荷が偏ってしまい，結果として並列処理効率が低下するという問題がある．

本稿では，ユーザにとって解析する負担が少ないパターンを高速に提示するために，FPclose をベースとして飽和頻出パターンを並列抽出する手法を提案する．さらに，頻出パターン並列抽出において問題となる，タスク負荷の偏りを平坦化する手法を提案する．提案した手法を PC クラスタ上で実装し，性能評価を行なう．

2. 関 連 研 究

第 2 節では，頻出パターン抽出問題の関連研究について述べる．

2.1 飽和頻出パターン抽出問題

従来の頻出パターン抽出問題は，以下のように定義される [9]．アイテムの集合を $I=\{i_1,i_2,\cdots,i_m\}$，トランザクションデータベースを $T=\{t_1,t_2,\cdots,t_n|t_i\subseteq I\}$ とする．T の各要素 t_i をトランザクションとする．アイテム集合 X のサポート $support(X)$ は，T 全体に対して X を含むトランザクションの割合を表す．頻出パターンとは，ユーザが与えた最小サポートを満たすアイテム集合である．従来の頻出パターン抽出問題は，ユーザが与えた最小サポートを満たす全ての頻出パターンを，T から抽出することである．

従来の頻出パターン抽出アルゴリズムでは，結果として莫大な数のパターンが得られてしまう．莫大な数のパターンから，有用な知識を得ることは困難である．この問題点を解決するために，飽和頻出パターン（$CFI: ClosedFrequentItemset$）を抽出するアルゴリズムが提案されている [6] [8]．パターン P が飽和パターンであるということは，以下の二つの条件を同時に満たす P' が存在しないことである．

（1） P' が P のスーパーセットである．

（2） P を含む全トランザクションで，P' も含まれる

飽和パターン P が，最小サポートを超えるサポート値を持っているとき，P は CFI である．

2.2 FP-tree 構造と FP-growth アルゴリズム

2000 年に Han らによって提案された FP-growth は，候補パターンを生成せずに全ての頻出パターンを抽出する手法である [5]．FP-growth は，FP-tree 構造と呼ばれる特殊なデータ構造を利用する．この FP-tree 構造は，巨大なトランザクションデータベースをコンパクトに圧縮したデータ構造であるため，パターン抽出に必要なスキャン回数を削減することができる．

2.2.1 FP-tree 構造

FP-tree 構造においては，頻出アイテムのみが頻出パターン抽出に使われる．頻出アイテムを発見するために，データベースを 1 回スキャンする必要がある．抽出された頻出アイテムをサポートの値により，頻度が降順になるように並び替える（このリストを F-list とする）．そして，空（null）のラベルを持つ木のルートを作る（この木を T とする）．

頻出 1-item から FP-tree 構造を構築するために，2 回目のスキャンが行われる．具体的には，以下の手順で FP-tree 構造を構築する．

各トランザクションに対して，

（1） トランザクションから頻出アイテムを抽出し，F-list に従ってソーティングを行う．

（2） T が F-list の要素である子を持っていれば，その子のカウントを 1 増やす．そうでないときは，新しくカウント 1 を持つ子を作る．

（3） F-list の最後の要素まで 1，2 の操作を繰り返す．

全てのトランザクションで処理を終えたら，同じ名前（アイテム ID）を持つノードにリンクを付ける．さらに，各頻出アイテム a_i に対して，先頭の a_i を指すヘッダテーブルを作成する．以上の手順で構築された FP-tree 構造には，頻出パターン抽出に必要な情報が全て含まれている．つまり，元のデータベースから頻出パターンを抽出するためには，構築された FP-tree に対してマイニングを行えばよい．

2.2.2 FP-growth アルゴリズム

FP-growth アルゴリズムでは，FP-tree 構造の 2 つの特徴を利用する．1 つ目の特徴は，どんな頻出アイテム a_i に対しても，先頭の a_i を示すヘッダテーブルから，a_i のノードリンクをたどることにより，a_i を含む生成可能な頻出パターンを全て得ることができるということである．2 つ目の特徴は，パス P にあるノード a_i を含む頻出パターンを数えるためには，パス P におけるノード a_i の prefix-path を求めるだけでよく，prefix-path にあるノードのカウントは，ノード a_i のカウントと等しいということである．ここで，あるアイテム m に対応する prefix-path を m 条件付きパターンベースとよぶ．m 条件付きパターンベースから m 条件付き FP-tree を構築し，新たな条件付きパターンベースが生成できなくなるまで FP-growth を再帰的に繰り返すことで，m を含む全てのパターンを抽出することができる．

2.3 FPclose

FP-growth によって生成されるパターンの数は莫大である．

この問題を解決するために，2003 年に Grahne らによって FP-close アルゴリズムが提案された [2]．FPclose は，FPgrowth をベースにして飽和頻出パターンを抽出するアルゴリズムである．FPclose は，2003 年 11 月時点で，Closed Pattern Mining アルゴリズムの中で最速と判定されている [1]．FPclose では，FP-growth 同様に FP-tree を構築し，構築した FP-tree から頻出パターン生成を行う．生成された頻出パターンを CFI-tree(Closed Frequent Itemset tree) に挿入し CFI を抽出する．

2.4 FP-growth 無共有並列実行

近年，大規模なデータを処理するにあたり，並列計算環境が不可欠になりつつある．特に，PC クラスタのようにコストパフォーマンスの高い分散型並列計算機が注目されている．従来の並列化手法の多くは，Apriori に基づくものである [10], [12]．これは，FP-tree 構造のような独自のデータ構造を利用する手法では，データ構造の一貫性を保つのが困難とされていたからである．

2003 年には，Iko らによって PC クラスタ上で FP-growth を並列実行する手法が提案された [4]．この手法では，条件付きパターンベース処理が他のアイテムの条件付きパターンベース処理と独立して行える点に注目している．

手順としては，まず，1 回目のスキャンでは各ノードがローカルトランザクションからアイテムの数え上げを行い，マージしてグローバルな F-list を得る．2 回目のスキャンでは，各ノードがローカルトランザクションから FP-tree を構築する．ローカル FP-tree が構築されると，各ノードは条件付きパターンベースを生成する．生成されたローカル条件付きパターンベースは，割り付けられたアイテムを処理するノードに集約される．それぞれのノードが完全な条件つきパターンベースを受けてから，独立してその条件つきパターンベースの処理を完成させる．

さらに，[4] で提案された手法では，十分な台数効果を得るために「パス深さ」と呼ばれるパラメータを活用して，実行ノード間の負荷を動的に均等化するメカニズムを提案している．

3. 提案手法

本節では，FPclose をベースとした並列化手法を提案する．FPclose は，Closed Pattern Mining アルゴリズムの中で最速のアルゴリズムである（2003 年 11 月）．FPclose を並列化することで，冗長性の少ないパターン（CFI）を従来よりも高速に抽出することができる．

FPclose は，FPgrowth と同様に頻出パターンを生成したのち，生成されたパターンを検証し CFI を抽出する手法である．よって，提案する FPclose 並列化手法における頻出パターン生成処理でも，FP-growth 並列化手法 [4] に基づいて拡張を行う．並列化の手順を以下で述べる．

3.1 FP-tree 構築並列化

並列化によって得られる大きな利点の一つとして考えられるのが，トランザクションデータに対するスキャンを並列化できることである．例えば，p 台のノードのディスクにデータが格納されていれば，シーケンシャルにデータを読む場合に比べて，単位時間あたり p 倍のデータ量を読むことができる．この利点を生かして，FP-tree 構築を並列化する．

まず，従来手法 [10] [4] と同様に，あらかじめトランザクションデータ TDB を先頭から各ノード p_i に分配しておく．各ノード p_i は，ローカルトランザクション TDB_i からローカルな FP-tree 構造 FPT_i を生成する．

3.1.1 ローカルサポートの数え上げ

1 回目の TDB に対するスキャンでは，頻出 1-item を数える．各ノード p_i には，同数のトランザクションが分配されている．各ノード p_i は，分配されたトランザクション TDB_i に出現するローカルなアイテムを数え上げる．ローカルなアイテムを数え上げている間，各ノードは独立して動作する．

3.1.2 グローバルサポートの数え上げ

各ノードにおいてローカルなアイテムの数え上げが終了したら，入力データ全体でのサポートを得るために，通信を行ってローカルサポートをマージする．マージにあたっては，各ノードが他ノードにローカルサポートを送信することによって，グローバルサポートが数えられ，頻出でないアイテムが除かれる．見つかった頻出アイテムは，サポートが降順に並ぶようにソートされる．このリストが FP-growth における F-list である

3.1.3 ローカル FP-tree 構築

ローカル FP-tree を構築するために，2 回目の TDB に対するスキャンを行う．各ノードはローカルなトランザクションからローカル FP-tree を構築する．各トランザクションに対して F-list にある頻出アイテムだけが集められる．F-list の順序にしたがって各アイテムの頻度が降順に並ぶようにソートされる．各トランザクション内でソートされたアイテムは，ローカル FP-tree を構築するために，以下のように使われる．

（1） トランザクションに出現するアイテムに対して，root に子ノードが存在するかどうかをチェックする．

（2） 子ノードが存在すれば，子ノードのサポートをインクリメントする．子ノードが存在しなければ，出現したアイテムのために，新しいノードを追加し，サポートを 1 とする．

（3） 読み込んだアイテムを新しい root として，トランザクションに出現する次のアイテムに対しても手順 1. を繰り返す．

以上の手順で，各ノードのローカルディスクからローカル FP-tree を構築する．ヘッダテーブルには，データベース全体で頻出するアイテムと，そのサポートが格納されている．また，ヘッダテーブルは，各ノードのローカル FP-tree で最初に出現する各アイテムへのリンクを持つ．

3.2 パス深さを考慮した負荷分散

生成された FPT_i からローカルな条件付きパターンベース CPB_i を生成する．しかし，CPB_i だけでは TDB 全体に頻出するパターンを抽出できないため，通信を行ってグローバルな条件付きパターンベースを求める必要がある．そこで，各ノード p_i が CPB_i を生成したら，どのノードがどのアイテムに基づくグローバル条件付きパターンベースを処理するかを決定し，CPB_i をマージする．

ここで，[4] では，処理ノードをハッシュ関数によって決定している．しかし，ノード毎のタスク負荷の偏りが問題となる並列マイニングにおいては，負荷をより平坦化する処理ノード決

定法が必要である．

そこで，本手法では，負荷をより平坦化するために，FPT_i に含まれる情報を利用し，処理ノードを決定する．条件付きパターンベースの処理ノードを決定する時点では，各ノードには TDB_i を読み込んで構築した FPT_i が既に存在する．つまり，この時点で TDB における出現アイテムの特性をある程度得ることができる．具体的には, [4] のタスク分割の際に用いられるパス深さ(注1)を用いることで，条件付きパターンベース処理時間を予測し，タスク負荷の平坦化を図る．そこで，条件付きパターンベースを処理するノードを決定する前に，各ノードの FPT_i から各条件付きパターンベースのパス深さの値を予測し，その値をもとに条件付パターンベースを処理するノードを決定する．これによって，条件付きパターンベース処理に必要なタスクが平坦化され，全体としての処理の高速化が期待できる．

具体的には以下の手順をとる．まず，各ノードはローカルな条件付きパターンベースから，パス深さを得る．ローカルなパス深さを得たら，トランザクションデータベース全体でのパス深さを得るために，通信を行って全ノードで各アイテムの最大パス深さを得る．続いて，全アイテムの最大パス深さを合計し，これを総タスク量とする．この総タスク量をノード数（PU 台数）で割った値が，１つのノードあたりの処理すべきタスク量 α である．処理ノードを決定する際は，1 つのノードの処理タスク量が α を超えないように，Flist の先頭から順に割り振っていく．これによって，各ノードの処理タスク量がある程度均等化される．

3.3 CFI 抽出並列化

ローカルな条件付きパターンベースをマージした結果，各ノードがグローバルな条件付きパターンベースを取得したら，各ノードは独立して頻出パターンを生成する．生成された頻出パターンは，各ノードの CFI-tree に挿入され，CFI であるか否かのチェックが行われる．ここで，CFI-tree 挿入に関する処理は，各ノードで独立して実行することができる．なぜならば，X 条件付き CFI-tree に必要な情報は，すべて X 条件付きパターンベースに含まれるからである．つまり，各ノードに割り当てられたアイテム X を含むパターンが CFI であるか否かをチェックするために必要な情報は，X が割り当てられたノードが持つ，グローバルな X 条件付きパターンベースに格納されているからである(注2)．以上の手続きを踏むことで，あらかじめ負荷を平坦化した並列化手法を用いつつ，冗長性の少ないパターンである CFI を抽出することができる．図 1 に一連の手順を擬似コード化したものを示す.

4. 性 能 評 価

3. で述べた手法を MPI を用いて実装し，PC クラスタ上で性能評価を行った．ただし，3.2 で述べた負荷分散機構に関し

(注1)：パス深さとは，条件付きパターンベース内の最小サポート値を満たす最長パターンの長さである．条件付きパターンベースの処理時間は，この「パス深さ」に比例することが知られている．

(注2)：FPclose アルゴリズムでは，CFI-tree は FP-tree と同じ数だけ構築される．

```
input : database TDB, Items I, minimum support min_sup

SEND Process:
{
    sum_path_depth = 0;
    dest_node = 0;
    local_support = count_support(TDB, I);
    global_support = merge_support(local_support);
    FList = create_FList(global_support, min_sup);
    FPtree = construct_FPtree(FList,TDB);
    /* get total_path_depth */
    local_path_depth = get_path_depth(FList, I);
    approximate_path_depth = max(local_path_depth);
    total_path_depth = sum(approximate_path_depth);
    foreach item in FList
    {
            cond_pbase = create_cond_pbase(FPtree, item);
            path_depth = get_path_depth(FPtree, item);
            sum_path_depth = sum_path_depth + get_path_depth(FList, item);
            if(sum_path_depth > (dest_node + 1) * total_path_depth / num_node)
                    dest_node++;
            send_cond_pbase(dest_node, cond_pbase);
    }
}
RECV Process:
{
    cond_pbase = collect_cond_pbase();
    cond_FPtree = construct_fptree(cond_pbase, FList);
    FPclose(cond_FPtree, NULL);
}
```

図 1　Parallel FPclose 擬似コード

てはまだ実装を終えておらず，現時点では CFI 抽出単純並列化（ハッシュによる処理ノード決定法）に関するデータを測定した．今回の実験では，並列化を施していない FPclose と比較した速度向上率と，大規模データに対するスケーラビリティを評価した．

4.1 実 験 環 境

3. で述べた手法を 32 ノード PC クラスタ上で実装した．各ノードは，Intel Pentium4 プロセッサ 2.40GHz と，1GB(512MB × 2) のメモリを持つ．また，各ノードは，1000Mbps イーサネットで接続されている．並列アルゴリズムの実装には，フリーな通信ライブラリである MPICH(Version 1.2.5) を用いた．また，性能を評価するために，IBM 人工データセット生成プログラム [3] を用いた．このプログラムは，入力パラメータによって性質の異なるデータセットを生成する．

4.2 逐次 FPclose と比較した速度向上率

今回の実験では，T10I4D100k，T10I4D500k，T10I4D1000k としてデータセットを生成した．ノード台数を 1 台から 32 台まで変化させて，最小サポート値 2.0%，1.5%，1.0%，0.5% の場合について実験を行った．ノード台数が 1 台の場合は，並列化を施していないプログラムを実行し，実行時間を測定した．実験結果を，PU 台数と速度向上率の関係でグラフに表したのが 2，図 3，図 4 である．図 2 は T10I4D100k に対する実験結果，図 3 は T10I4D500k に対する実験結果，図 4 は T10I4D1000k に対する実験結果を表している．

図 2 を見ると，どのサポート値に対しても，4PU 実行時に速度向上の最高値を得ることが分かる．また，最小サポートが小さくなるにつれて，同数の PU を投入したとしても，速度向

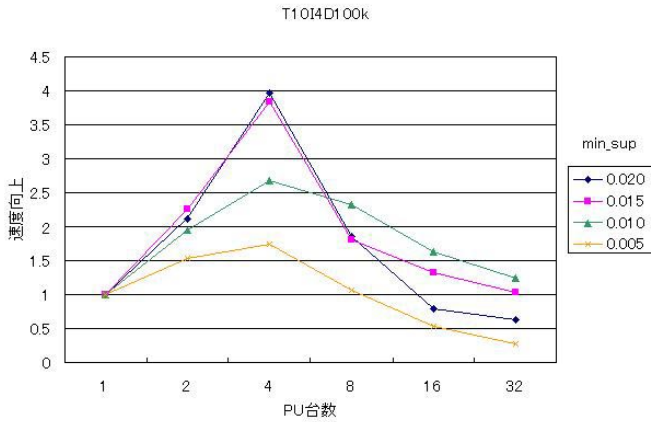

図 2　PU 台数と速度向上（トランザクション数 100k）

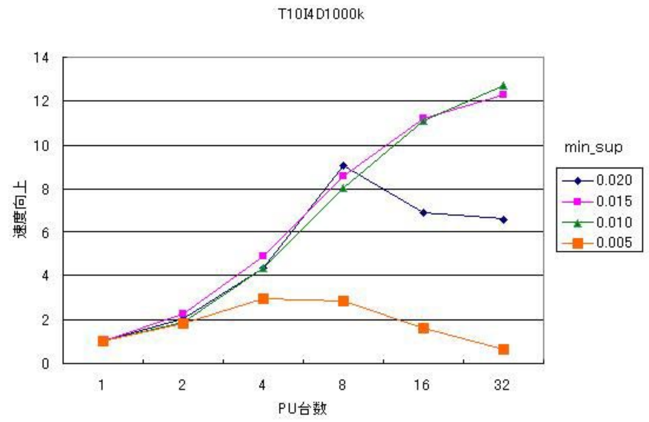

図 4　PU 台数と速度向上（トランザクション数 1000k）

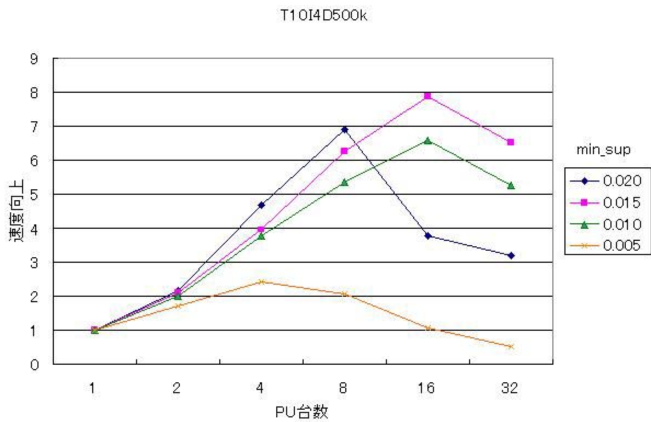

図 3　PU 台数と速度向上（トランザクション数 500k）

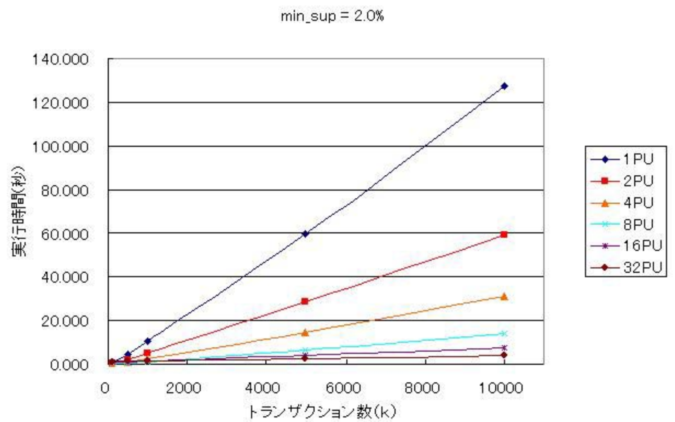

図 5　スケーラビリティの評価

上が低下してゆく．これらの原因は，最小サポートが小さい場合，抽出される頻出アイテムや，計算すべき条件付きパターンが増加し，それに伴って通信量も増加する(注3)ためである．通信量が増加することによって，全体としての実行時間が増加してしまう．

図 3，図 4 に注目すると，トランザクション数が増加するにつれて，速度向上の最大値が大きくなるということと，そのときの PU 台数の値が大きくなることが分かる．具体的には，図 3（トランザクション数 500k）では 16PU 投入時に約 8 倍の速度向上を得るのに対して，図 4（トランザクション数 1000k）では 32PU 投入時に約 12 倍の速度向上を得る．これは，トランザクション数が大きくなるにつれて，全体の実行時間に対してディスクアクセス時間が占める割合が大きくなるためである．

また，図 3 と図 4 を見ると，最小サポートが最大（0.020）となる場合に，速度向上の最大値が得られていないことが分かる．これは，処理ノード決定法がハッシュに基づくためであると考えられる．現時点では，ハッシュに基づいて処理ノードを決定しているため，処理時間の大きいパターンをあるノードが他ノードよりも多く割り当てられる可能性がある．結果的に，ノード毎でタスク負荷の偏りが発生し，処理時間の掛かるノードがボトルネックとなり，速度向上が低下してしまう．

4.3　大規模データに対するスケーラビリティ評価

本手法の大規模データに対するスケーラビリティを評価するために，同じく IBM の実行データ生成プログラム [3] を用いて，トランザクション数の違う新たなデータセット T10I4D100k，T10I4D500k，T10I4D1000k，T10I4D5000k，T10I4D10000k を生成した．これら 5 つのデータセットを対象に，本手法の実行時間を測定した．最小サポートを 2%，PU 台数は 8 台として実験を行った．図 5 にトランザクション数と実行時間の関係を示す．

図 5 が示すように，本手法を 8PU で並列実行した場合の実行時間は，単一ノードで CFI 抽出を実行した場合の実行時間よりも，トランザクション数，つまりデータの規模に影響を受けることが少なかった．具体的には，単一ノードで実行した場合，トランザクションが 1000k から 5000k に増えると，実行時間は 50 秒程度余計に掛かった．一方，本手法により，8PU で並列実行した場合，トランザクションが 1000k から 5000k に増えても，実行時間は 5 秒程度しか増加しなかった．これは，データが大規模になるにつれて，ディスクアクセスに必要な時間が増加するためであると考えられる．PC クラスタなどの分

(注3)：現時点の実装では，条件付きパターン収集処理において通信が多く発生するため，改善の余地はある．

散並列計算環境においては，ディスクアクセスを並列化することによって，総実行時間を大幅に短縮することができるため，大規模なデータを対象にしても，パターン抽出を高速に実行することができる．

5. おわりに

データマイニング分野の重要な問題として，頻出パターン抽出がある．頻出パターン抽出は，バスケット解析や DNA 解析などに適用されるが，大規模なデータを対象に処理を行うため高速化の研究が不可欠である．高速化に関する研究の中でも，メモリ容量不足やディスクアクセス増加といったリソース面での制約を緩めるために，PC クラスタなどをターゲットとした並列化手法が提案されている．従来提案されてきた並列化手法の多くは，Apriori や FP-growth をベースとする手法である．これらの並列化手法では，全ての頻出パターンを抽出するため，結果として莫大な数のパターンが抽出され，ユーザに負担が掛かるという問題がある．また，頻出パターン並列抽出処理においては，トランザクションデータベース中に出現するアイテムの特性により，ノード毎でタスク負荷が偏ってしまい，結果として並列処理効率が低下するという問題がある．本稿では，ユーザにとって解析する負担が少ないパターンを高速に提示するために，FPclose をベースとして飽和頻出パターンを並列抽出する手法を提案した．さらに，頻出パターン並列抽出において問題となる，タスク負荷の偏りを平坦化する手法を提案した．この提案した手法を PC クラスタ上で実装し，性能評価を行なった結果，32 ノード PC クラスタ上で 30.9 倍程度の速度向上を得ることができた．また，本手法がデータの大規模化に対応しうることも確認できた．

今後の課題としては，現時点の実装では条件付きパターンベース収集処理において，冗長な通信を行っているため，並列処理効率が低下しているという問題がある．また，提案した負荷分散機構を実装し，その有用性を検証する必要がある．

文　献

[1] Goethals, M. J. Zaki, "FIMI '03: Workshop on Frequent Itemset Mining Implementations, " In Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Inplementations, 2003.

[2] Gosta Grahne and Jianfei Zhu, Efficiently Using Prefix-trees in Mining Frequent Itemsets, Proceeding of the First IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), 2003.

[3] IBM Quest Data Mining Project. Quest synthetic data generation code. http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html

[4] Iko Pramudiono, Masaru Kitsuregawa, "Tree Structure based Parallel Frequent Pattern Mining on PC Cluster ," In Proceedings of 14th International Conference on Database and Expert Systems Applications (DEXA'2003), pp.537-547, 2003.

[5] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," In Proceedings of the ACM SIGMOD Conference on Management of Data, pp.1-12, 2000.

[6] J.Pei, J.Han, and R.Mao, "CLOSET: An e ± cient algorithm for mining frequent closed itemsets, " In DMKD '00, 2000.

[7] J.S. Park, M. Chen, and P.S. Yu, "An effective hash-based algorithm for mining association rules," In Proceedings of the ACM SIGMOD Conference on Management of Data, pp.175-186, 1995.

[8] J.Wang, J. Han, and J. Pei, "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets, " In Proceedings of the ACM SIGKDD Conderence, Aug. 2003.

[9] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," In Proceedings of the International Conference on Very Large Data Bases, pp. 487-499, 1994.

[10] R. Agrawal and R. Srikant, "Parallel mining of association rules," IEEE Transactions on Knowledge and Data Engineering, 8(6), 1996.

[11] S. Brin, R. Motowani, J. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," In Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 255-264, 1997.

[12] T. Shintani and M. Kitsuregawa, "Hash based parallel algorithms for mining association rules," In Proceeding International Conference on Parallel and Distributed Information Systems, pp.19-30, 1996.