A-001

マルチコア計算機クラスタ上における段階的一般化法の並列処理

Parallel Processing of the Step-wise Generalization Method on a Multi-Core Computer Cluster

田村 慶一 †　　北上 始 †
Keiichi Tamura　Hajime Kitakami

1 はじめに

曖昧な問合せ処理（文献 [1, 2, 3, 4, 5, 6, 7, 8, 9]）は，テキストデータベースや配列データベースに対する類似する部分文字列の検索をさし，データマイニングの分野で重要な要素技術となっている．曖昧な問合せの結果として多くの類似する部分文字列が得られる．本論文では，この類似する部分文字列の集合をミスマッチクラスタと呼ぶ．

ミスマッチクラスタをユーザが直接閲覧してその規則性を把握することは困難である．そこで，ミスマッチクラスタを表現する最小汎化集合，すなわち極大な汎化配列パターンの集合と汎化できなかった部分文字列を抽出する研究 [10, 11, 12, 13] が行われている．最小汎化集合とは，ミスマッチクラスタをカバーする最小の汎化集合をさす，すなわち，それ以上に汎化するとミスマッチクラスタの要素とは無関係の要素を含んでしまう汎化集合である．ミスマッチクラスタを表現する最小汎化集合を抽出することができれば，ミスマッチクラスタの規則性を把握することが容易となる．

最小汎化集合を効率的に抽出する手法として，段階的一般化法 [12, 13] が提案されている．段階的一般化法では，ミスマッチクラスタ MIS の冪集合 2^{MIS}（$2^{|MIS|}$ 個の部分集合 $SubMIS$ の集まり）に含まれる要素を，ボトムアップに小さいサイズの要素から順に列挙することで最小汎化集合を抽出する．段階的一般化法では，列挙木の枝刈りを行い，最小汎化集合を効率的に抽出することができる．しかしながら，ミスマッチクラスタを構成する部分文字列数が増えると計算時間が膨大となる．

そこで，本論文では，マルチコア計算機クラスタ上における段階的一般化法の並列化手法を提案する．近年，CPU のマルチコア化が急速に進んでおり，計算機クラスタを構成する計算機の CPU はマルチコアとなっている．本論文では，マルチコアの CPU を搭載する計算機で構成される計算機クラスタのことをマルチコア計算機クラスタと呼ぶ．本研究では，マルチコア計算機クラスタの特徴を考慮した並列化モデルを用いて，段階的一般化法の並列化を図る．

段階的一般化法の主な処理は列挙木の探索となる．シングルコアの計算機クラスタにおいて，分散型ワーカモデルが深さの偏りが極端に異なる列挙木の並列探索において効率的であることが示されている [14]．段階的一般化法が扱う列挙木も深さの偏りが極端に異なる．そこで，マルチコアの計算機クラスタにおいて，列挙木を効率的に並列探索するために，既存の分散型ワーカモデル [14] を改良した新しい並列化モデルを提案する．

分散型ワーカモデルの新しい並列化モデルの特徴は以下の通りである．

- 各計算機で起動するワーカはひとつで，コア数に応じたスレッド（ワーカスレッドと呼ぶ）をワーカ内に複数の起動し，ワーカスレッドにおいてタスクを実行する．各コア上でワーカを起動する場合と比較して，ワーカ内でコア数に応じたワーカスレッドを起動した方が，計算機内で負荷の偏りを効率的に解消できる．
- ワーカは 2 種類のタスクプールを持つ．ワーカは 1 つのグローバルタスクプールを持ち，各ワーカスレッドが 1 つのローカルタスクプールを持つ．各ワーカスレッドがローカルタスクプールを持つことで，ワーカスレッド間の競合が少なくなる．
- ワーカスレッド間とワーカ間との 2 つの階層に分けて，キャッシュベースのランダムタスク・ステイル法 [14] を用いて負荷の偏りを解消する．キャッシュベースのランダムタスク・ステイル法を階層的に用いることで，計算機内と計算機間の負荷の偏りを効率的に解消できる．

提案する並列化モデルを用いて，段階的一般化法を並列化し，評価実験を行った．評価実験の結果，実験を行ったすべてのデータセットに対して良いスピードアップの結果が得られた．また，既存の分散型ワーカモデルと比較して，発生するメッセージ数が少なく，効率が良いことを確認することができた．

本論文の構成は以下の通りである．第 2 章で用語と問題の定義を行い，第 3 章で段階的一般化法について説明する．第 4 章で段階的一般化法の並列処理を示し，第 5 章で関連研究を述べる．第 6 章で評価実験の実験結果を示し，第 7 章で本論文のまとめを行う．

2 用語と問題の定義

本章では，用語と記号の定義，最小汎化集合の定義を簡単に述べる．

2.1 曖昧な問合せとミスマッチクラスタ

部分文字列 K と許容誤差 $r(\geq 0)$ が，問合せ Q として与えられているとする．この問合せ Q による処理で，ハミング距離 $d(K, K') \leq r$ を満たす部分文字列 K' が配列データベース DB からすべて選択されるとき，問合せ Q を曖昧な問合せと呼ぶ．また，曖昧な問合せにより配列データベース DB から検索条件を満たす長さ k の部分文字列 <$inst$> の集合が得られる．この集合のことをミスマッチクラスタ MIS と呼ぶ．

† 広島市立大学大学院情報科学研究科，Graduate School of Information Sciences, Hiroshima City University

2.2 曖昧性を表現する汎化配列パターン

アルファベット Σ の部分集合を Σ_i とするとき，k 個の Σ_i を並べたパターンを k-汎化配列パターンと呼び，$<pat^k>=<\Sigma_1\Sigma_2\cdots\Sigma_{k-1}\Sigma_k>$ と表記する．ただし，Σ_i は，たびたび括弧 [ ] の中に Σ_i の全要素を列挙した表記をする．例えば，$<[AB][CD]>$ は 2-汎化配列パターンであり，$\Sigma_1=\{A,B\}$，$\Sigma_2=\{C,D\}$ である．

2.3 インスタンスを導出する関数

k-汎化配列パターン $<\Sigma_1\Sigma_2\cdots\Sigma_{k-1}\Sigma_k>$ から k-インスタンスのすべて，すなわち長さ k の部分文字列の集合を導出する関数を $EVAL(<\Sigma_1\Sigma_2\cdots\Sigma_{k-1}\Sigma_k>)$ と表記する．例えば，$EVAL(<[AB][CD]>)$ から導出されるインスタンスは，{<AC>，<AD>，<BC>，<BD>} である．

2 つの k-汎化配列パターン $<pat_1{}^k>$ と $<pat_2{}^k>$ について，$EVAL(<pat_1{}^k>)\supseteq EVAL(<pat_2{}^k>)$ が成立するとき，$<pat_2{}^k>$ は $<pat_1{}^k>$ に含まれるという．別の言い方では，$<pat_2{}^k>$ は $<pat_1{}^k>$ に冗長であるともいう．

2.4 最汎パターン

ある k-インスタンスの集合を I^k とする．ここで，$1\le j\le k$ に対して，$\Sigma_j=\{inst[j]\mid inst\in I^k\}$ であるとする．ただし，$inst[j]$ は，$inst$ の先頭から j 番目の文字を意味する．このとき，$<\Sigma_1\Sigma_2\cdots\Sigma_k>$ を k-インスタンス集合 I^k に対する最汎パターンと呼び，この最汎パターンを $MGP(I^k)$ と表記する．例えば，インスタンスの集合 {<ABF>, <AEF>, <DBF>} の最汎パターン MGP は $<[AD][BE]F>$ となる．

2.5 最小汎化集合

ある集合 $MGS=\{G_1,G_2,\cdots,G_m\}$ が，k-汎化配列パターン $<pat^k>$ および k-インスタンス $<inst^k>$ から構成されているとする $(1\le m\le |MIS|)$．ただし，$EVAL(\{<pat^k>\})\subseteq MIS$ かつ $<inst^k>\in MIS$ を満たすものとする．

この集合 MGS が以下の性質を満たすとき，MGS を MIS に対する最小汎化集合と呼ぶ．

(1) $EVAL(MGS)=MIS$ が成立する．
(2) MGS の任意の 2 要素 G_i,G_j に対して，G_i と G_j の間には冗長な関係が存在しない $(1\le i\ne j\le m)$．
(3) MGS に含まれるどの要素 G_i も極大である $(1\le i\le m)$．すなわち，さらに汎化すると MIS に存在しないインスタンスを含んでしまうことになる．
(4) 上記の (1) ~ (3) を満たす任意の MGS' に対して，$|MGS'|\le|MGS|$ が成立する．

3 段階的一般化法

段階的一般化法では，ミスマッチクラスタ MIS の冪集合 2^{MIS} に含まれる要素を，ボトムアップに小さいサイズの要素から順に列挙しながら最小汎化集合を抽出する．図 1 に示すように，順序集合を用いて列挙木を作りながら探索を行う．また，すべての要素を列挙するのではなく，効率的に探索を行うために，部分木の枝刈りを行っている．

3.1 列挙木ノードに対する基本操作

列挙木の成長や解集合の探索のための列挙木ノードに対する 5 つの基本操作を述べる．

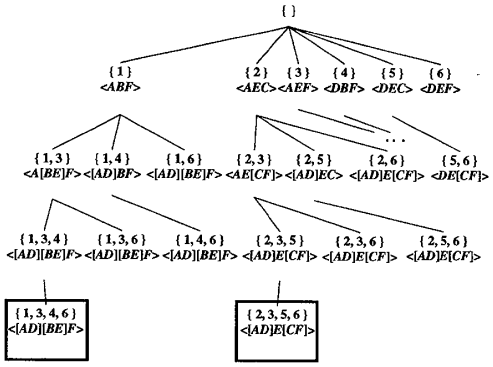


図 1 列挙木による探索処理の例

(操作 1) 親ノードから子ノードの列挙

MIS の冪集合 2^{MIS} に包含関係 $\subseteq$ を定義した順序集合 $(2^{MIS},\subseteq)$ を用いて，列挙木を生成すると，親ノードは子ノードに含まれるという性質を利用できる．親ノードが空集合である場合は，ルートノードを意味するので，その子ノードは 1 つの要素番号から構成される集合とし，すべての子ノードを幅優先で辞書式順に列挙する．親ノードが空集合でない場合は，子ノードを次のように列挙する．P' を親ノード P の兄弟ノードとするとき，$|P|+1=|P\cup P'|$ となるような P' を選択し，辞書式順に $P\cup P'$ を子ノードとして列挙する．図 1 に例を示す．図 1 をみると，親ノードを {1} とするとき，その子ノードは $\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{1,6\}$ の順に列挙される．親ノードを {2} とするとき，その子ノードは $\{2,3\},\{2,4\},\{2,5\},\{2,6\}$ の順に列挙される．親ノードを {6} とするとき，その子ノードは存在しない．

(操作 2) 最汎パターンの計算

列挙木ノードに対応する部分集合 $SubMIS$ から最汎パターン $MGP(SubMIS)$ を計算する．図 1 における列挙木ノード $\{1,3,4\}$ に対応する部分集合 $SubMIS$ は，$\{<ABF>,<AEF>,<DBF>\}$ である．従って，最汎パターン $MGP(SubMIS)$ は $<[AD][BE]F>$ となる．

(操作 3) 枝刈り処理

$MGP(SubMIS)$ に負のインスタンスが含まれた時点で $SubMIS$ を持つ列挙木ノードをルートとする部分列挙木の枝刈りが可能である．負のインスタンスとは MIS に存在しないインスタンスのことである．以下の条件式が成立するならば $EVAL(MGP(SubMIS))$ に負のインスタンスが存在すると判定できる．

$$EVAL(MGP(SubMIS))-MIS\ne\phi$$

例えば，図 1 において，列挙木ノード $\{1,3,5\}$ は存在しない．このノードの最汎パターンは $<[AD][BE][FC]>$ である．$EVAL(\{<[AD][BE][FC]>\})$ 集合には MIS には存在しない負のインスタンス $<ABC>$ を含むため，上記の式を満足してしまう．従って，このノードは枝刈りにより除去される．

(操作 4) 候補解集合の構成要素の収集

葉ノードの先祖に該当する非葉ノードは葉ノードに冗長であるので，葉ノードの部分集合 ($Leaf$ と表

記する）から計算された最汎パターン *MGP(Leaf)* は最も一般化されたパターンである．従って，*MGP(Leaf)* は候補解集合の一要素である．列挙木ノードが葉ノードであるための必要十分条件は，その列挙木ノードから子ノードを幅優先ですべて列挙したときに，(1) 親ノードとその兄弟ノードをどのように組み合わせても子ノードがもはや列挙不可能か，または (2) そのどの子ノードにも負のインスタンスが含まれてしまうことである．図 1 の列挙木ノード {1} および {1,3} については，いずれも葉ノードの条件 (2) を満たさない．{1,3,6} については，もはやその子ノードを列挙できないため，葉ノードの条件 (1) を満たす．

(操作 5) 冗長性の除去

葉ノード間に冗長性が存在する可能性がある．列挙木探索を終了した時点で，冗長な葉ノードを候補解集合から一括除去する．図 1 の例では，{1,3,4,6},{1,3,6},{1,4,6},{1,6} の順に葉ノードであるため，候補解集合に含まれるが，{1,3,6},{1,4,6},{1,6} はどれも {1,3,4,6} に冗長であるので，これらは候補解集合から除去される．

3.2 段階的一般化法の処理手順

図 2 に，段階的一般化法の処理手順を示す．図 2 の処理手順 (1) は，列挙木の深さ優先探索を行うための変数 *Stack* と候補解を格納するための変数 *Candidate* について，初期化を行う処理である．処理手順 (2)(a) は，3.1 の操作 1 に基づいて行われる親ノードから子ノードを列挙する処理である．処理手順 (2)(b)① は操作 2 に基づいて行われる最汎パターンを計算する処理であり，処理手順 (2)(b)② は操作 3 に基づいて行われる枝刈り処理である．処理手順 (2)(c) は操作 4 に基づいて行われる候補解集合の構成要素を収集する処理であり，処理手順 (3) は操作 5 に基づいて行われる冗長性除去の処理である．以上の処理を経て，*Candidate* に残された要素集合が最小汎化集合となり，それに支持数を付与して作られる結果が処理手順 (4) で出力される．

4 段階的一般化法の並列処理

本章では，改良した分散型ワーカモデルを提案し，段階的一般化法の並列処理について説明する．

4.1 並列化モデル

図 3 に提案する並列化モデルを示す．各計算機にはワーカを 1 つ起動する．ワーカは複数のワーカスレッドを持つ．ワーカは他のすべてのワーカと通信を行うことができ，ワーカスレッドは他のすべてのワーカスレッドとやり取りができる．

ワーカは，2 種類のタスクプールを持つ．1 つ目がグローバルタスクプールで，2 つ目がローカルタスクプールである．グローバルタスクプールはワーカで 1 つ設置する．ローカルタスクプールはワーカスレッドごとに 1 つ設置する．タスクの詳しい定義は 4.2 節で説明する．

ワーカスレッドは，ローカルタスクプールからタスクを取り出し，タスクを実行する．ワーカスレッドは，ローカルタスクプールにタスクがなくなった場合のみ，グローバルタスクプールからタスクを取り出す．ワーカスレッドごとにローカルタスクプールを持つことで，タスク処理に関わる競合がワーカスレッド間で発生することを回避できる．

```
(1) Stack:={{1},{2},···,{n}}; ただし，{i} ∈ Stack の i はミスマッチクラスタ MIS に存在する要素の識別番号とし，Stack の要素は辞書式順に取り出せるように管理する．
(2) Candidate := {}; ただし，Candidate を候補解を格納する領域とする．
(3) while(Stack ≠ φ)
      for each S ∈ Stack
        (a) Next := S を親ノードとして列挙された子ノード C の集合;
        (b) for each C ∈ Next
            ① <pat>:= MGP(C に対する SubMIS);
            ② if EVAL(<pat>) − MIS ≠ φ then C を捨てる;
              else Stack := Stack ∪ {C};
        (c) if S が葉ノードの条件を満たす
            then Candidate:=Candidate ∪ MGP(S に対する SubMIS);
(4) Candidate から冗長な要素を除去した後，配列データベース DB における各要素の支持数を計算し，結果として出力する;
```

図 2 段階的一般化法の処理手順

4.2 タスクの定義

段階的一般化法において，列挙木の部分木探索は他の部分木探索とは独立に行うことできる．例えば，図 1 において，ノード {1} 以下の部分木探索とノード {2} 以下の部分木探索とは独立して行うことができる．そこで，部分木を同時に探索することで並列に列挙木を探索することが可能である．

しかしながら，列挙木の部分木の深さは偏りが大きいため，部分木探索をタスクとして定義して処理を行うと負荷の偏りを効率的に解消することができない．そこで，次に示すように異なる 2 つの粒度のタスクを定義する．

部分木探索タスク

列挙木のある親ノードから生成された子ノード集合から，子ノード集合の子孫ノードをすべて探索する部分木探索を部分木探索タスクと定義する．図 4 に部分探索タスクの例を示す．親ノード {1} の子ノード集合 {{1,3}, {1,4}, {1,6}} から図 4 に示すようにそれ以降のすべての子孫ノードを探索するタスクが部分木探索タスクである．1 つのノードを起点として部分木探索タスクを定義しなかった理由は，ノードの子ノードを列挙するときにそのノードの他の兄弟ノードが必要となるためである．ただし，ルートノードを親ノードとする子集合ノードを対象とするタスクだけは，初期タスクとして子ノード集合の中の 1 つの子ノード以下の部分木を探索する部分木探索タスクとして定義する．

子ノード探索タスク

列挙木のある親ノードから生成された子ノード集合から，子ノード集合のすべての子ノード（孫ノードと呼ぶ）のみを探索することをノード探索タスクと定義する．子ノード探索タスクを実行すると，子ノード集

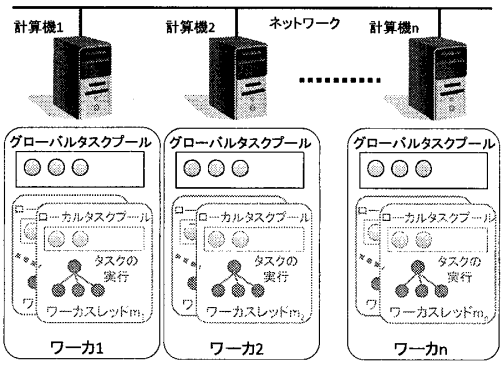


図 3　並列化モデル

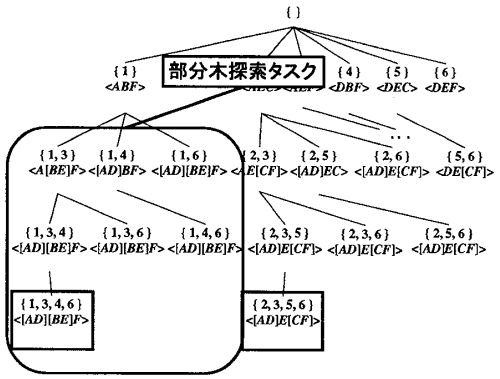


図 4　部分木探索タスク

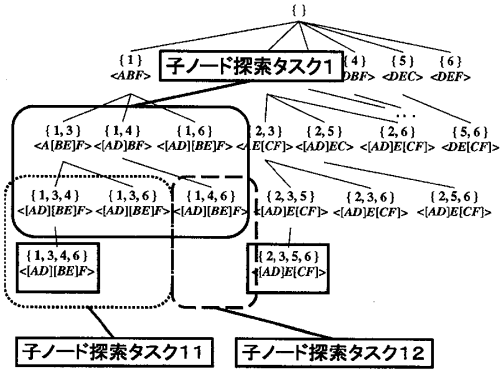


図 5　子ノード探索タスク

合の各ノードを親ノードとする新たな子ノード探索タスクが複数生成される可能性がある．図 5 に子ノード探索タスクの例を示す．親ノード {1} の子ノード集合 {{1,3}, {1,4}, {1,6}} から，孫ノード {1,3,4}, {1,3,6}, {1,4,6} を列挙するタスクが子ノード探索タスクである．このタスクを実行すると，ノード {1,3} の子ノード集合 {{1,3,4},{1,3,6}} を対象する子ノード探索タスク 11 とノード {1,4} の子ノード集合 {{1,4,6}} を対象とする子ノード探索タスク 12 が生成される．

4.3　タスク管理と実行

グローバルタスクプールでは部分木探索タスクを管理し，ローカルタスクプールでは子ノード探索タスクを管理する．ワーカスレッドは，ローカルタスクプールにタスクが存在しない場合，グローバルタスクプールから部分木探索タスクを取り出す．そして，その部分木探索タスクを子ノード探索タスクに変換し，子ノード探索タスクを実行していく．部分木探索タスクと子ノード探索タスクの違いは，すべての探索をするか，1 つ下の子ノードの探索で止まり，それ以降の探索を新たなタスクとして生成するかの違いである．よって，部分木探索タスクと子ノード探索タスクはお互いに変換が可能である．子ノード探索タスクを実行することで新たに生成された子ノード探索タスクはローカルタスクプールに格納する．ワーカスレッドは，ローカルタスクプールが空になるまで，子ノード探索タスクを実行する．

4.4　処理手順

ここでは，ワーカとワーカスレッドの処理手順を示す．ワーカは計算機数分 n 個起動する．ワーカスレッドは，コア数に応じて起動する．ワーカ i のワーカスレッド数を w_i とする．ここで，各ワーカのグローバルタスクプールを GTP，ワーカスレッド j のローカルタスクプールを LTP_j と表記する．また，結果を集めるワーカを他のワーカと区別してリーダと呼ぶこととする．各ワーカは解の候補を格納するための集合として $W\_Candidate$ とワーカスレッド j が解の候補を格納するための集合として $WH\_Candidate_j$ を持つ．

ワーカ

(1) 検索結果からミスマッチクラスタ MIS を生成する．
(2) 初期タスクをグローバルタスクプール GTP に挿入する．
(3) ワーカスレッドを w_i 個起動する．
(4) w_i 個のワーカスレッドが終了するのを待つ．
(5) 各 $WH\_Candidate_j$ を $W\_Candidate$ にまとめる．
(6) $W\_Candidate$ から冗長な要素を除去する．
(7) もし，ワーカがリーダでなければ，リーダに $W\_Candidate$ の要素をすべて送信する．もし，ワーカがリーダならば，他のすべてのワーカから候補集合を受信して，$W\_Candidate$ に追加する．
(8) リーダは，$W\_Candidate$ から冗長な要素を除去する．
(9) リーダは，結果を出力する．

ワーカスレッド

(1) グローバルタスクプール GTP から部分木探索タスクを取り出し，子ノード探索タスクとしてローカルタスクプール LTP_j に挿入する．グローバルタスクプール GTP が空の場合，負荷分散処理に入る．負荷分散処理については，4.5 節で詳しく説明する．
(2) ローカルタスクプール LTP_j から子ノード探索タスク T を 1 つ取りだす．
(3) 子ノード探索タスク T に登録されている子ノードの集合を TC とする．各子ノード $S \in TC$ について次の処理を行う．
  (a) 子ノード S を親ノードとして，S のすべての子ノード C を列挙する．列挙した子ノードの集合を $Next$ とする．生成された各子ノード $C \in Next$ について以下の処理を行う．
    (i) C に対応するミスマッチクラスタの部分集合 $SubMIS$ の最汎パターン $MGP(SubMIS)$ を求める．
    (ii) $EVAL(MGP(SubMIS)) - MIS \neq \phi$ を満

たすならば，$Next$ から C を取り除く．

(b) $Next$ が空でなければ，S を親ノード，$Next$ を子ノード集合とする子ノード探索タスク T_{new} を，ローカルタスクプール LTP_j に挿入する．$Next$ が空ならば，S は解の候補であるため，S を $WT\_Candidate_j$ に挿入する．

(4) ローカルタスクプール LTP_j が空でなければ (2) へ戻る．空ならば，(1) へ戻る．

4.5 負荷分散手法

ワーカ間とワーカスレッド間とで階層的に分けて負荷の偏りを解消する．

4.5.1 ワーカスレッド間

グローバルタスクプールが空の場合，他のワーカスレッドにタスク要求を出す．タスク要求を出すワーカスレッドの選択方法として，キャッシュベースのランダムタスク・ステイル法 [14] を用いる．他のワーカスレッドから子ノード探索タスクを返事として受け取ると，その子ノード探索タスクをローカルタスクプールに挿入する．すべてのワーカスレッドからタスクが空であるという返事が来た場合，ワーカ間の負荷分散処理に入る．

また，他のワーカスレッドからタスク要求を受け取ったワーカスレッドはローカルタスクプールに子ノード探索タスクが存在するならば，子ノード探索タスクを取り出し，タスク要求の返事として返す．もし，ローカルタスクプールが空ならば，タスクが存在しないことを示す返事を返す．

4.5.2 ワーカ間

ワーカ内でタスクが無くなると他のワーカに対してタスク要求を出す．タスク要求を出すワーカの選択方法として，同じく，キャッシュベースのランダムタスク・ステイル法を用いる．他のワーカから部分木探索タスクを返事として受け取ると，その部分木探索タスクをグローバルタスクプールに挿入する．すべてのワーカから返事が返ってきた結果，すべてのワーカにタスクが存在しない場合，ワーカスレッドは終了する．

また，他のワーカからタスク要求を受け取ったワーカは，グローバルタスクプールに部分木探索タスクが存在するならば，部分木探索タスクを取り出し，タスク要求の返事としてタスクを返す．もし，グローバルタスクプールにタスクが存在しなければ，すべてのワーカスレッドから子ノード探索タスクを奪い取り，奪い取った子ノード探索タスクを部分木探索タスクとしてグローバルタスクプールに格納する．そして，グローバルタスクプールから部分木探索タスクを 1 つ取り出し，タスク要求の返事として返す．もし，グローバルタスクプールが空ならば，タスクが存在しないことを示す返事を返す．

4.5.3 キャッシュベースのランダムタスク・ステイル法

キャッシュベースのランダムタスク・ステイル法は，ランダムにタスク要求を出すのではなく，タスクを獲得できたワーカを記録しておき，記録がある場合，記録されたワーカに最初に優先的にタスク要求を出す手法である．列挙木の探索では負荷の偏りが大きくなり，特定のワーカ群にタスクが集中する可能性がある．このような場合，タスク要求をランダムに出すと無駄なタスク要求が増えるため，全体の性能が低下する可能性が高い．キャッシュベースのランダムタスク・ステイル法では，タスクを獲得できたワーカにはタスクが存在する可能性が高いため，そのワーカに優先的にタスク要求を出すことにより，この問題が発生することを防いでいる．

5 関連研究

深さの偏りの大きな列挙木の並列探索手法として，計算機クラスタ上で頻出パターンを並列に抽出する並列 Modified PrefixSpan 法が提案されている [15]．頻出パターンを抽出する処理は列挙木の探索となる．提案されている手法では，マスタ・ワーカモデルを使用して，マスタとワーカに階層的にタスクプールを配置する方法を提案している．また，文献 [15] で提案された手法は，同じように列挙木の探索となる分枝限定法の並列化にも適用されている [16]．

文献 [14] では，大規模な計算機クラスタ上での最適な列挙木の並列探索手法を提案している．この研究では，計算機クラスタの規模が大きくなるとマスタ・ワーカモデルよりも分散型ワーカモデルの方が優れていることが示されている．また，この研究では，負荷の偏りが大きくなる列挙木の並列探索において最適な動的負荷分散手法としてキャッシュベースのランダムタスク・ステイル法を提案している．本研究では，このランダムタスク・ステイル法をワーカ間，ワーカスレッド間に階層的に適用することでマルチコアの計算機クラスタ上での効率的な動的負荷分散手法を実現している．

一方，マルチコアの CPU を持つ単一計算機上における列挙木の並列探索に関する研究 [17] が行われている．ただし，マルチコアの特徴を考慮した列挙木の並列探索方法を提案しているが，単一計算機上のみの手法に止まっている．

シングルコアの CPU から構成される計算機クラスタや単一計算機上のマルチコアでは，すでに列挙木の最適な並列探索手法が提案されている．しかしながら，本研究が対象としているマルチコア計算機クラスタでは，最適な手法が明らかになっているとはいえない．ただし，分散型ワーカモデルの優位性が既存の研究ですでに示されていることから，分散型ワーカモデルがマルチコア計算機クラスタで最適に動作するように，分散型ワーカモデルに改良を加えた．

6 評価実験

評価実験では次の 3 つの実験を行う．実験 1 では，4 種類のデータを使用して，ワーカスレッド数を変化させたときのスピードアップ（速度向上比）を測定する．実験 2 では，ワーカ間の動的負荷分散の効果を示すために，動的負荷分散がある場合と無い場合のスピードアップを比較する．実験 3 では，既存の分散型ワーカモデルを使用した場合と提案手法とを比較する．

表 1 に実験評価に用いた 4 種類のデータセットの特徴を示す．この 4 種類のデータセットは PROSITE[18] から取得したものである．最初に，4 種類のデータセットのそれぞれに対して，曖昧な問合せ処理を行う．曖昧な問合せにおける検索文字は各データセットに含まれ

表 1　データセット

データセット名	登録番号	データ件数	総長 (bytes)
Kringle	PS00021	90	59123
Homeobox	PS00027	1272	448596
PTS_EIIA_2	PS00372	50	15470
HTH_ASNC_1	PS00519	37	5766

表 2　ミスマッチクラスタの特徴

データセット名	許容誤差数	ミスマッチクラスタの要素数
Kringle	4	3577
Homeobox	7	2385
PTS_EIIA_2	7	2229
HTH_ASNC_1	11	3019

表 3　最小汎化集合の特徴

データセット名	最小汎化集合の要素数	処理時間 (sec)
Kringle	918	39
Homeobox	717	141
PTS_EIIA_2	1514	404
HTH_ASNC_1	1674	2488

表 4　マルチコア計算機クラスタの各計算機の構成

項目	内容
CPU	AMD PhenomX4 9350e(2.00GHz)
メモリ	DDR2-800 2GB
ディスク	500GB
OS	Fedora Core 12
コンパイラ	gcc 4.4.2

るモチーフの一部分を用いている．この検索文字列に可変長ワイルドカード領域が含まれる場合は，それを固定長ワイルドカード領域が含まれる部分文字列の集合に表現し，その集合内の要素を論理和で結合した条件で問合せを行っている．表 2 に問合せに使用した許容誤差数と問合せの結果得られたミスマッチクラスタの要素数を示す．

また，表 3 に各データセットのミスマッチクラスタから抽出される最小汎化集合の要素数と，最小汎化集合を抽出するのに必要となる処理時間を示す．処理時間は，表 4 に示す計算機で測定を行ったときの処理時間である．

表 4 に使用したマルチコア計算機クラスタの各計算機の構成を示す．表 4 に示す計算機から構成されるマルチコア計算機クラスタを使用して，実験を行った．各計算機は，1Gbit/sec のイーサネットで接続されている．

6.1　実験 1

実験 1 では，4 種類のデータセットについて，スピードアップ（速度向上比）を測定する．この実験では，ワーカ 1 つでワーカスレッド 1 つだけを使用したときの処理時間を複数のワーカ，複数のワーカスレッドを使用したときの処理時間で割った値を測定した．ワーカスレッド数は 1, 4, 8, 12, 16 と変化させて測定を行った．ワーカ数は，1, 4, 4, 4, 4 とワーカスレッド数が 1 以外のときは，計算機を 4 台使用した．各ワーカで起動するワーカスレッド数は一定とし，ワーカスレッド数が 4 のときは各ワーカで 1 つのワーカスレッドを，ワーカスレッド数が 8 のときは各ワーカで 2 つのワーカスレッドを，ワーカスレッド数が 12 のときは各ワーカで 3 つのワーカスレッドを，ワーカスレッド数が 16 のときは，各ワーカで 4 つのワーカスレッドを起動した．

図 6(a) に *Kringle* データセット，図 6(b) に *HomeoBox* データセット，図 6(c) に *PTS_EIIA_2* データセット，図 6(d) に *HTH_ASNC_1* データセットの測定結果を示す．

Kringle データセットは，ワーカスレッド数が 16 のときに 14.4 倍のスピードアップであった．処理時間に直すと 2.7 秒であり，他のオーバヘッドが大きくなりスピードアップの若干比率が落ちている．*HomeoBox* データセットは，ワーカスレッド数が 16 のときに 15.6 倍とほぼ理想線形加速が得られている．*PTS_EIIA_2* データセットと *HTH_ASNC_1* データセットについては，16.16 倍と 16.47 倍と超線形加速となった．両データセットともに 2GB 以上のメモリを使用しており，ワーカスレッド数が 1 のときの処理時間が OS のスラッシングにより若干遅くなったため，超線形加速が得られた．

6.2　実験 2

実験 2 では，ワーカ間の動的負荷分散の効果を示すために，動的負荷分散がある場合と無い場合のスピードアップを比較する．測定条件は実験 1 と同じである．

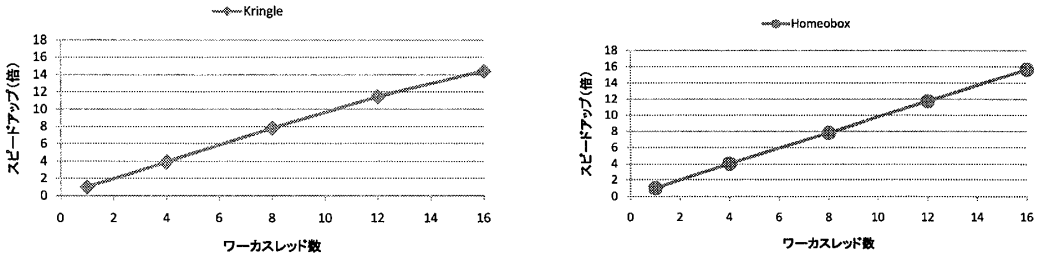
図 7(a) に *Kringle* データセット，図 7(b) に *HomeoBox* データセット，図 7(c) に *PTS_EIIA_2* データセット，図 7(d) に *HTH_ASNC_1* データセットの測定結果を示す．

図 7 のグラフから明らかなように，ワーカ間の動的負荷分散が無い場合は，ほぼ 9 倍近くのスピードアップで終わっている．測定結果より，ワーカ間の動的負荷分散が効率的に働いていることが分かる．

ワーカ間の動的負荷分散がない場合は，初期タスクの配置に依存する可能性が高い．部分木探索タスクのコストが予め分かるならば，負荷が均等になるように初期タスクを配置できるが，列挙木の探索は探索してみないと深さがどれくらいになるか分からないので，通常は，最適な初期タスクの配置をするのは困難である．本論文で提案する手法を用いると，初期タスクの配置に関係なく動的に負荷の偏りを解消することができる．

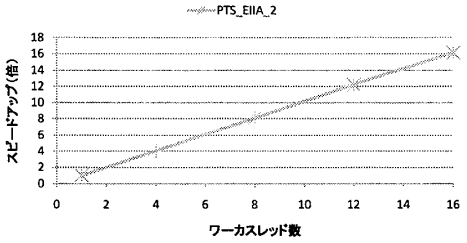
6.3　実験 3

実験 3 では，従来の分散型ワーカモデルを使用し，すべてのコアにワーカを配置した場合（以下，従来手法と表記する）と提案手法を比較するために，ワーカ間で発生する負荷分散のためのメッセージ数を比較する．この実験では，*PTS_EIIA_2* データセットと

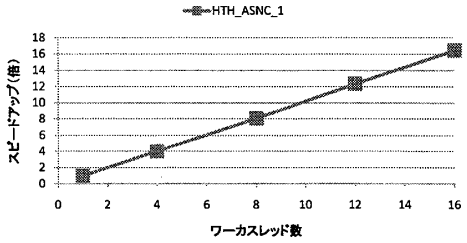


(a) *Kringle* データセット

(b) *HomeoBox* データセット

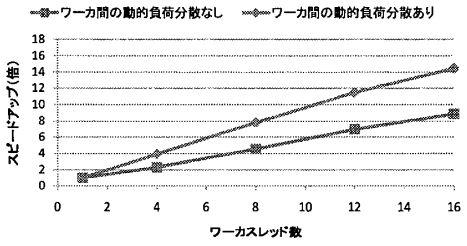


(c) *PTS_EIIA_2* データセット

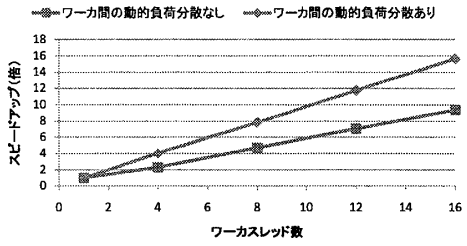


(d) *HTH_ASNC_1* データセット

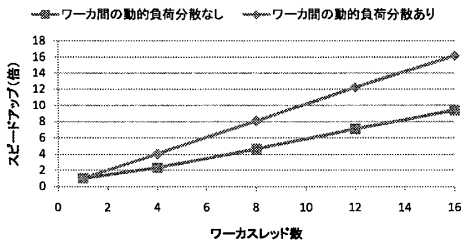
図 6　スピードアップ（速度向上比）



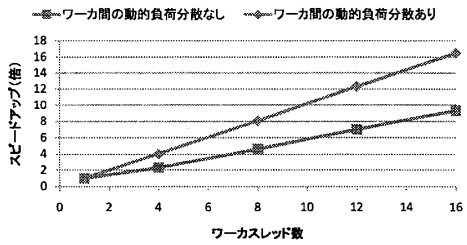
(a) *Kringle* データセット



(b) *HomeoBox* データセット



(c) *PTS_EIIA_2* データセット



(d) *HTH_ASNC_1* データセット

図 7　ワーカ間の動的負荷分散の効果

HTH_ASNC_1 データセットのみを使用する．実験では，4 つの計算機を使用する．従来手法ではワーカの数を 4，8，12，16 と変化させて，提案手法ではワーカの数は 4 で，ワーカスレッドの数を 4，8，12，16 と変化させて測定を行う．

表 5 に *PTS_EIIA_2* データセット，表 6 に *HTH_ASNC_1* データセットの測定結果を示す．提案手法は各計算機のワーカ数が一定であるためワーカスレッドを増やしたとしてもメッセージ数の増加は見られなかった．従来手法はワーカ数が増えるためメッセージ数が増加していっている．

この実験から分かるように，従来の分散型ワーカモデルを使用し，すべてのコアにワーカを配置した場合，同じ計算機に配置されているワーカ間でもメッセージのやり取りが必要となるため，提案手法と比較して効率が悪いといえる．計算機のコア数がさらに増加していくと，メッセージ数がさらに増加して性能の低下を招くと考えられる．

7　まとめ

本論文では，マルチコア計算機クラスタ上における段階的一般化法の並列処理を述べた．段階的一般化法の並列化では，マルチコアの計算機クラスタの特徴を活かすために，マルチコアに対応した分散型ワーカモデルを用いた．評価実験により評価した結果，良い性能が得ら

表 5 *PTS_EIIA_2* データセットでのメッセージ数

ワーカもしくは ワーカスレッド数	提案 手法	従来 手法
4	533	534
8	495	577
12	493	722
16	410	1183

表 6 *HTH_ASNC_1* データセットでのメッセージ数

ワーカもしくは ワーカスレッド数	提案 手法	従来 手法
4	578	578
8	419	599
12	551	772
16	445	1208

れることを確認した．

これからの課題として，計算機の台数を増やして性能評価を行うこと，計算機の構成が異なる環境下での性能評価や提案手法の他のアプリケーションへの応用があげられる．

謝辞

本研究の一部は，日本学術振興会，科学研究費補助金 (基盤研究 (C)，課題番号：20500137)，文部科学省・科学研究費補助金（課題番号：20700095）の支援により行われた．

参考文献

[1] Zvi Galil and Kunsoo Park. An improved algorithm for approximate string matching. *SIAM Journal on Computing*, Vol. 19, No. 6, pp. 989–999, 1990.

[2] Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, Vol. 46, No. 3, pp. 395–415, 1999.

[3] Jeehee Yoon Sanghyun Park, Wesley W. Chu and Chihcheng Hsu. Efficient searches for similar subsequences of different lengths in sequence databases. In *Proceedings of ICDE2000*, pp. 23–32. IEEE Computer Society, 2000.

[4] Fei Shi and Cread Mefford. A new indexing method for approximate search in text databases. In *Proceedings of CIT'05*, pp. 70–76. IEEE Computer Society, 2005.

[5] Chen Li Liang Jin, Nick Koudas and Anthony K. H. Tung. Indexing mixed types for approximate retrieval. In *Proceedings of VLDB2005*, pp. 793–804, 2005.

[6] Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *LATIN*, pp. 374–390, 1998.

[7] Laurent Marsan and Marie-France Sagot. Extracting structured motifs using a suffix tree - algorithms and application to promoter consensus identification. In *Proceedings of RECOMB2000*, pp. 210–219, 2000.

[8] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in dna sequences. In *ISMB*, pp. 269–278, 2000.

[9] Eleazar Eskin and Pavel A. Pevzner. Finding composite regulatory patterns in dna sequences. In *Proceedings of the 10th International Conference on Intelligent Systems for Molecular Biology*, pp. 354–363, 2002.

[10] Kotaro Araki, Keiichi Tamura, Tomoyuki Kato, Yasuma Mori, and Hajime Kitakami. Extraction of ambiguous sequential patterns with least minimum generalization from mismatch clusters. In *Proceedings of THE THIRD INTERNATIONAL CONFERENCE ON SIGNAL-IMAGE TECHNOLOGY and INTERNET-BASED SYSTEMS*, pp. 32–39. IEEE Computer Society Press, 2007.

[11] 荒木康太郎, 田村慶一, 加藤智之, 北上始. ミスマッチクラスタに対する最小汎化パターン抽出方式. 日本データベース学会論文誌 (DBSJ Letters), Vol. 6, No. 3, pp. 5–8, 2007.

[12] Hiroaki Kimura, Hajime Kitakami, Kotaro Araki, and Keiichi Tamura. A stepwise generalization mthod for extracting minimum generalized set from mismatch cluster. In *Proceedings of BIOCOMP'08, Vol.II*, pp. 998–1004, 2008.

[13] 田村慶一, 木村浩明, 荒木康太郎, 北上始. 段階的一般化法によるミスマッチクラスタを表現する最小汎化集合の効率的抽出. 電子情報通信学会論文誌 D「データ工学特集号」, Vol. J93-D, No. 3, pp. 189–202, 2010.

[14] Makoto Takaki, Keiichi Tamura, Toshihide Sutou, and Hajime Kitakami. A new dynamic load balancing technique for parallel modified prefixspan with distributed worker paradigm and its performance evaluation. In *Proceedings of ISHPC2005*, pp. 227–237, 2005.

[15] 高木允, 田村慶一, 周藤俊秀, 北上始. 並列 modified prefixspan 法の並列化と動的負荷分散手法. 情報処理学会論文誌 数理モデル化と応用, Vol. 46, No. SIG10, pp. 138–152, 2005.

[16] 田村慶一, 岩木稔, 高木允, 北上始. Pc クラスタにおける混合整数計画問題の並列処理とその性能評価. 情報処理学会論文誌 数理モデル化と応用, Vol. 46, No. SIG17, pp. 56–69, 2005.

[17] Shirish Tatikonda and Srinivasan Parthasarathy. Mining tree-structured data on multicore systems. *Proc. VLDB Endow.*, Vol. 2, No. 1, pp. 694–705, 2009.

[18] http://kr.expasy.org/prosite/.