グリッド環境下での分散型ワーカモデルを用いた Modified PrefixSpan 法の動的負荷分散方式

高木　允† 田村　慶一†† 北上　始††

† 広島市立大学大学院情報科学研究科　〒 731–3194 広島市安佐南区大塚東三丁目 4 番 1 号
†† 広島市立大学情報科学部　〒 731–3194 広島市安佐南区大塚東三丁目 4 番 1 号
E-mail: †makoto@db.its.hiroshima-cu.ac.jp, ††{ktamura,kitakami}@its.hiroshima-cu.ac.jp

あらまし　我々は，アミノ酸配列からモチーフとなりうる頻出パターンを高速に抽出するために，グリッド上での Modified PrefixSpan 法（MPS）の動的負荷分散方式について研究を行っている．MPS は，タスク間の負荷の偏りが極端で，タスクの負荷を予め見積もれないという特徴を持っているため，ある PC クラスタに極端に負荷が偏る．本論文では，MPS のグリッド化モデルとして分散型ワーカモデルを用い，PC クラスタ間の動的負荷分散方式としてマルチキャストとキャッシュ機能を併用した方式である *Cache-based Multicast Steal*（CMS）を提案する．分散型ワーカモデルを用いることで PC クラスタ数の増加に柔軟に対応できる．動的負荷分散方式として CMS を用いることで，MPS の特徴である負荷の偏りや通信遅延による性能低下を抑え，各 PC クラスタの負荷を均一化できる．仮想的なグリッド環境下で実験を行い，提案方式の有効性を示す．

キーワード　配列パターン抽出，データマイニング，並列処理，グリッド，動的負荷分散

Dynamic Load Balancing for Modified PrefixSpan on Grid Environment Using Distributed Worker Model

Makoto TAKAKI†, Keiichi TAMURA††, and Hajime KITAKAMI††

† Graduate school of Information Sciences, Hiroshima City University　3–4–1 Ozuka-Higashi, Asa-Minami-Ku, Hiroshima, 731–3194 Japan
†† Faculty of Information Sciences, Hiroshima City University　3–4–1 Ozuka-Higashi, Asa-Minami-Ku, Hiroshima, 731–3194 Japan
E-mail: †makoto@db.its.hiroshima-cu.ac.jp, ††{ktamura,kitakami}@its.hiroshima-cu.ac.jp

Abstract　We are developing the dynamic load balancing for a Modified PrefixSpan on the grid environment to extract the frequent patterns that can became motif in amino acid sequences at high speed. The Modified PrefixSpan has two characteristics: one is extremely biases of the load and the other is not able to estimate the load of the task. The load is extremely biased to a certain PC cluster because of these characteristics. In this paper, the distributed worker model is applied to the Modified PrefixSpan on the grid environment. Moreover, we propose the *Cache-based Multicast Steal* (CMS) which combined multicast and cache function as a dynamic load balancing technique. The distributed worker model has the flexibility for the increase of the number of PC clusters. The CMS can reduce the overhead generated by the communication delay and can uniform the load of all the PC clusters.

Key words　sequential pattern extraction, data mining, parallel processing, grid, dynamic load balancing

1. はじめに

モチーフはアミノ酸配列上における特徴的なパターンであり，生物進化の過程で保存されてきたタンパク質の機能に関係していると考えられている．データマイニングの立場から，アミノ酸配列から取り出した特徴的なパターンよりモチーフを発見する手法が注目されている．PROSITE [1] や DDBJ [2] に登録されているモチーフは，可変長のワイルドカード領域を含む．可変長のワイルドカードを含む特徴的なパターンを抽出することは，文字配列上の頻出パターンを取り出すことになる．

複数のアミノ酸配列から頻出パターンを抽出するためのアルゴリズムとして Modified PrefixSpan 法 [3], [4] が提案されてい

る．Modified PrefixSpan 法は PrefixSpan 法 [5] を拡張したものであり，複数のアミノ酸配列から可変長ワイルドカード領域と固定長ワイルドカード領域を含む頻出パターンを抽出することができる．Modified PrefixSpan 法は，優れた抽出能力を持つ．しかしながら，機能性を重視したこの方法は，非常に長い計算時間を要す．そこで，頻出パターンを高速に抽出することが重要な課題となる．

ひとつの PC クラスタ上で Modified PrefixSpan 法を並列化する研究が様々行われている [6]〜[9]．しかしながら，ひとつの PC クラスタ上での並列化では性能向上に限界がある．近年，インターネットに繋がった複数の PC クラスタ（グローバルグリッド）を使って科学技術計算などを高速に行う研究が注目されている．グリッド環境下の豊富な資源を使うことで，より複雑な頻出パターンを実用的な時間で抽出可能となる．我々は，Modified PrefixSpan 法のグリッド化の研究 [10] を行っている．Modified PrefixSpan 法を並列化すると，PC 間での極端な負荷の偏り，負荷を予め見積もることができないといった特徴 [7] を持っている．グリッド環境下においても Modified PrefixSpan 法の特徴により，ある PC クラスタに極端に負荷が偏り，台数を増加させても効率的に全ての計算機を使用できないという問題が起こる．さらに，PC クラスタ間の通信には通信遅延を考慮しなければならない．

本論文では，負荷の極端な偏り・負荷を見積もることができないという Modified PrefixSpan 法の特徴を考慮し，分散型ワーカモデルをグリッド環境での Modified PrefixSpan 法の並列化モデルとして用いた．また，動的負荷分散方式として，キャッシュ機能とマルチキャスト方式を組み合わせた動的負荷分散方式である *Cache-based Multicast Steal* を提案し，PC クラスタ間の通信遅延による性能低下を抑えつつ負荷の偏りを解消させた．

分散型ワーカモデルをグリッド環境下での Modified PrefixSpan 法の並列化モデルとして適用し，*Cache-based Multicast Steal* を動的負荷分散方式として実装した．仮想的なグリッド環境下で，提案した動的負荷分散方式と他の動的負荷分散方式との比較を行った．実験結果より，提案手法は通信遅延が変化しても通信遅延による処理時間の増加を抑えることができ，各 PC クラスタの負荷を均一化できたことを確認した．

本論文の構成は以下の通りである．2. では Modified PrefixSpan 法について説明する．3. でグリッド環境下での Modified PrefixSpan 法のタスクの定義と並列化モデルについて述べ，4. でグリッド環境下での Modified PrefixSpan 法の動的負荷分散方式の説明を行う．5. で関連研究について述べ，6. で性能評価を示す．7. で本論文のまとめを行う．

2. Modified PrefixSpan 法

本章では，アミノ酸配列から頻出パターンを抽出するアルゴリズムである Modified PrefixSpan 法について説明を行う．より詳細なアルゴリズムは文献 [4] を参照されたい．Modified PrefixSpan 法ではユーザが最小支持数（min_sup），最大ワイルドカード数（max_wc），最大誤差数（ε_{max}）を予め指定する．

2.1 問題の形式

アミノ酸配列データベースを S と表す．1 つのアミノ酸配列を s_{sid} と表すと（sid は配列の識別子である），S はタプル $\langle sid, s_{sid}\rangle$ の集合となる．1 つのアミノ酸配列を $s_{sid} = \langle a_1 a_2 \cdots a_m\rangle$ と表現する．アミノ酸配列 s_{sid} は 20 種類のアルファベットを組み合わせた文字列で構成される．アミノ酸配列に含まれる文字要素を Σ = {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y } と定義すると，$a_j \in \Sigma$ が成り立つ．

アミノ酸配列 s_{sid} の第 l 番目の文字要素は配列データとして $s_{sid}[l]$ で参照・更新ができる．ここで，表 1 をアミノ酸配列の例として考える．S は 5 つのタプルから構成される．$s_2 = \langle$ SFVKTA $\rangle$ であり，$s_2[1] = \langle$S$\rangle$ である．

長さ k のパターンを以下のように表す．

$$pat^k = \langle A_1 - x(i_1, j_1) - A_2 - x(i_2, j_2) - \cdots - x(i_{k-1}, j_{k-1}) - A_k\rangle.$$

記号 A_j は 20 文字のアルファベットのうち 1 文字の文字要素を示す．記号“ - ”は，隣同士がお互いに連続していることを表現するための記号である．記号 $x(i_n, j_n)$ はワイルドカード領域を示し，i_n，j_n はワイルドカード領域の長さを表し，$i_n \leqq max\_wc$ が成り立つ．$x(i_n, j_n)$ は，文字要素 A_{n-1} と A_n の間に長さ i_n から長さ j_n までのワイルドカード領域を含むということを意味する．$i_n < j_n$ のとき，可変長ワイルドカード領域を表す．$i_n = j_n$ のとき，固定長ワイルドカード領域を表し，$x(i_n, j_n)$ を $x(i_n)$ と表現する．このとき，$i_n - j_n \leqq \varepsilon_{max}$ が成り立つ．

例えば，$F * * * K * A$ というパターンを考える．文字要素“ * ”は任意の文字要素（ワイルドカード）1 文字を示す．このパターンは $\langle F - x(3) - K - x(1) - A\rangle$ と表現される．$x(3)$ はワイルドカード 3 文字が存在することを示している．また，可変長ワイルドカードを含むパターンの例として，長さ 2 のパターンである $\langle F - x(0, 3) - A\rangle$ を考える．このパターンは，$\langle F-x(0)-A\rangle$, $\langle F-x(1)-A\rangle$, $\langle F-x(2)-A\rangle$, $\langle F-x(3)-A\rangle$ の 4 つの基本パターンのいずれかを含むことを示している．

長さ k のパターン $\langle pat^k\rangle$ のアミノ酸配列データベース S における支持数は，パターン $\langle pat^k\rangle$ を含むタプルの数（配列の数）となる．

パターン $\langle pat^k\rangle$ を含むタプルの数が最小支持数以上のとき，パターン $\langle pat^k\rangle$ を k-頻出パターンと呼ぶ．支持数が cnt である k-頻出パターン $\langle pat^k\rangle$ を“ $\langle pat^k\rangle : cnt$ ”と表現する．ここで，n 個の k-頻出パターンが S より見つかったとすると，これらの k-頻出パターンを以下のように定義する．

$$P_k = \{\langle pat_1^k\rangle : cnt_1, \langle pat_2^k\rangle : cnt_2, \cdots, \langle pat_n^k\rangle : cnt_n\}$$

2.2 アルゴリズム

Modified PrefixSpan 法では，k-頻出パターンの最右端に続く 1 文字をすべてアミノ酸配列より取り出し，k-頻出パターン

表 1　アミノ酸配列データの例

sequence id	sequence
1	FKYAKWL
2	SFVKTA
3	ALR
4	MSKPL
5	FSKFLMAW

の最右端に結合することにより長さ（$k+1$）のパターンを作成する．もし，作成した長さ（$k+1$）のパターンが最小支持数を満たすならば長さ（$k+1$）のパターンは（$k+1$）-頻出パターンとなる．

k-頻出パターンから長さ（$k+1$）のパターンを作り出すために，アミノ酸配列データベース中に存在するすべての k-頻出パターンについて，各 k-頻出パターンの最右端文字の右隣の位置（オフセット）を記録する．このオフセットの集合を射影データベース（Projected Databases）と呼ぶ．頻出パターン $\langle pat^k \rangle$ の射影データベースの定義は次の通りである．

$$PDB(\langle pat^k \rangle) = \{(sid, pos) | s_{sid} \in S, pos \text{ は } \langle pat^k \rangle \text{ の} \\ \text{最右端文字の右隣の位置}, 1 \leqq pos \leqq \| s_{sid} \|\}$$

以下に Modified PrefixSpan 法の基本アルゴリズムの概要を示す．アルゴリズムは 2 つのフェーズから構成される．

- **フェーズ 1:**

まず，アミノ酸配列データベース S をスキャンし，現れるアルファベット α の支持数を数え上げていく．スキャン時に最右端文字の右隣のオフセットを記憶し，射影データベース $PDB(\langle \alpha \rangle)$ を構築していく．スキャンが終わったときに最小支持数よりも支持数が多いアルファベット α が 1-頻出パターン $\langle pat^1 \rangle$ となる．1-頻出パターン $\langle pat^1 \rangle$ を P_1 にそれぞれ挿入する．最後に 1-頻出パターン $\langle pat^1 \rangle$ に対応する射影データベース $PDB(\langle pat^1 \rangle)$ を $PDBLIST^1$ に挿入する．

- **フェーズ 2:**

もし $PDBLIST^k$ が空ならば処理を終了する．$PDBLIST^k$ の中から $PDB(\langle pat^k \rangle)$ $(k \geqq 1)$ を 1 つ取り出す．射影データベース $PDB(\langle pat^k \rangle)$ を使用して k-頻出パターンより長さ（$k+1$）のパターンを作成し，それぞれ支持数を数え上げていく．作成時に最右端文字の右隣のオフセットを記憶し射影作成と支持数の数え上げが終わったときに最小支持数よりも支持数が多い長さ（$k+1$）のパターンが（$k+1$）-頻出パターン $\langle pat^{k+1} \rangle$ となる．（$k+1$）-頻出パターン $\langle pat^{k+1} \rangle$ を P_{k+1} にそれぞれ挿入する．最後に，($k+1$)-頻出パターン $\langle pat^{k+1} \rangle$ に対応する射影データベース $PDB(\langle pat^{k+1} \rangle)$ を $PDBLIST^{k+1}$ に挿入する．

3. グリッド環境下における Modified PrefixSpan 法

本章では，グリッド環境下における Modified PrefixSpan 法について説明する．

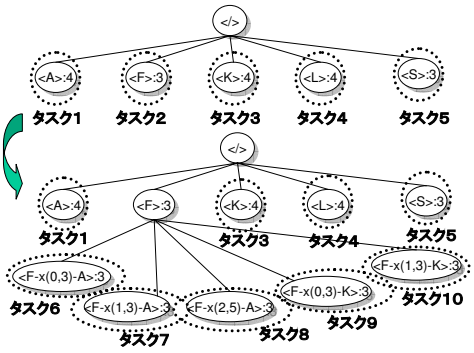

図 1　小粒度タスクの例

3.1 タスクの定義

Modified PrefixSpan 法の並列化においては，ひとつのタスクを「小粒度タスク」と定義する．小粒度タスクの特徴を以下に示す．また図 1 に小粒度タスクの例を示す．

（1） k-頻出パターンと k-頻出パターンの射影データベースの組み合わせで構成されている

（2） 小粒度タスクを 1 回実行すると k-頻出パターンから $(k+1)$-頻出パターンを抽出する（図 1）

（3） 小粒度タスクは完全に独立して実行可能である

（4） 小粒度タスクを 1 回実行して生成された $(k+1)$-頻出パターンは部分解（頻出パターン）であり，新たな小粒度タスクとなる（図 1）

（5） 1 つの小粒度タスクからそれ以降に抽出される頻出パターンの数，処理時間は見積もることはできない

（6） 小粒度タスクはそれ以降の全ての頻出パターンを抽出してもよいし，それ以降に生成されるどの小粒度タスクを他の PC または PC クラスタに移動させてもよい

上記の特徴を持った小粒度タスクを用いることで，極端に負荷の偏る PC クラスタから柔軟に負荷を移動させることができる．

3.2 並列化モデル

グリッド環境下においては，PC クラスタ内でのタスクのやりとりだけでなく，PC クラスタ間でのタスクのやりとりも必要となる．グリッド環境化でマスタワーカモデルを用いると，マスタの役割を担う 1 台のプロセスに処理が集中してしまう．PC クラスタ数が増加した場合，全ての PC クラスタを統括するマスタに処理が集中してしまい，マスタでの処理がボトルネックとなる．

本研究では，グリッド環境下での Modified PrefixSpan 法において，分散型ワーカモデルを適用する．つまり，PC クラスタ間の通信はマスタを置かず，各 PC クラスタが独立して他の PC クラスタとの通信を行う．

一般に，分散型ワーカモデルでは動的負荷分散方式として *receiver-initiated*（受信側起動）[11] ベースのランダムスティール法が用いられる．ランダムスティール法とは，タスクを処理し終えたワーカが他のワーカをランダムに選び，タスクリクエスト送信して，タスクを受け取る方式である．動的負荷分散方

式としてランダムスティール法を実装したグリッド環境下での Modified PrefixSpan 法の処理手順を以下に示す．各プロセスはタスクプールを持っており，小粒度タスクを格納することができる．

ここでは，PC クラスタ内の並列化モデルを限定しておらず，マスタワーカモデルでも分散型ワーカモデルでもよい．PC クラスタ内の並列化モデルとして分散型ワーカモデルを適用している場合には，処理を開始する役割や PC クラスタ間で通信を行う役割を務めるワーカが 1 台必要となる．このワーカをリーダと呼ぶ．以下に示す処理手順は PC クラスタ内の並列化モデルがマスタワーカモデルならばマスタ，分散型ワーカモデルならばリーダの処理手順を示す．

（1） 全ての PC クラスタのマスタ/リーダが 1-頻出パターンを抽出する．

（2） ラウンドロビン法を用いて，マスタ/リーダは抽出した 1-頻出パターンから 2-頻出パターンを抽出する処理を 1 つの小粒度タスクとして自身のタスクプールに挿入する．小粒度タスクのデータ表現は $\langle pat^1 \rangle$ と $\mathrm{PDB}\langle pat^1 \rangle$ とのペアとなる．

（3） PC クラスタ内のタスクが全てなくなるまで PC クラスタ内で動的負荷分散を行いながら処理を続ける．PC クラスタ内のタスクが全てなくなると（4）へ（PC クラスタ間で動的負荷分散）．

（4） ランダムに他のワーカを選択してタスクリクエストを送信する．(5）へ．

（5） 選択した PC クラスタから受け取ったタスク要求メッセージの返事により次の処理をおこなう．

(a) タスク要求メッセージの返事として小粒度タスクが返ってきた場合，タスクをマスタ/ワーカのタスクプールに挿入する．手順（3）に戻る．

(b) タスク要求メッセージの返事としてタスクなしの返事が返ってきた場合，(4）へ．既に全ての PC クラスタにタスクリクエストを送信済みで，タスクを受け取れなかった場合（6）へ．

（6） マスタ/リーダが全ての PC クラスタのマスタ/リーダに自身が所属している PC クラスタの処理を終了する合図を出し，PC クラスタ内の処理を全て終了する．

手順（3）と手順（4）で示したように，PC クラスタ内と PC クラスタ間の動的負荷分散の連携をとることで，できるだけ PC クラスタ間で通信を行わない．この連携により PC クラスタ間の通信回数を削減できる．

各 PC クラスタのマスタ/リーダは，上記の通常処理に加えて，タスク要求メッセージを受け取ると以下の処理を行う．タスク要求メッセージを受け取るタイミングや PC クラスタ内の動的負荷分散方式は様々考えられるが，実装依存であるためここでは手順のみを示す．この処理は通常のマスタ/リーダの処理と平行して動作しているプロセスである．

（1） 他の PC クラスタからタスクリクエストを受け取るまで待つ．タスクリクエストを受け取ると（2）へ．自身が所属している PC クラスタのマスタ/リーダから終了の合図を受け取ると処理を全て終了する．

（2） 自身が所属する PC クラスタのマスタ/リーダにタスクがあるかどうか調べる．タスクが全くなければ手順（3）へ進む．タスクがあれば手順（4）に進む．

（3） マスタ/リーダは PC クラスタ内にタスクがあるかどうか調べる．ここでは，PC クラスタ内でどのような動的負荷分散方式を用いてもよい．タスクがあれば，マスタ/リーダが小粒度タスクを回収し，自身のタスクプールへ挿入．(4）へ．PC クラスタ内にタスクが全くなければ，タスク要求メッセージの送信元にタスクなしを示す返事を送信し，(1）へ．

（4） マスタ/リーダのタスクプールより小粒度タスクを取り出す．タスク要求メッセージの送信元に返事として取り出した小粒度タスクを送信する．(1）へ．

上述した処理手順に従い，グリッド環境下で Modified PrefixSpan 法を動作させると，ある PC クラスタへの極端な負荷の偏りが原因で，PC クラスタ間のタスクリクエスト数が増加する．これは，動的負荷分散方式としてランダムスティール法を用いているため，タスクを全て処理し終えた PC クラスタが，タスクのない PC クラスタを選択してタスクリクエストを送る可能性が高くなるためである．PC クラスタ内と PC クラスタ間の動的負荷分散の連携を行っても PC クラスタ間のタスクリクエスト数が増加してしまう．

また，グリッド環境下では通信遅延を考慮した動的負荷分散を行わなければならない．次章では上記の問題点である負荷の極端な偏りと通信遅延を考慮した，グリッド環境下での Modified PrefixSpan 法の動的負荷分散方式を提案する．

4. グリッド環境下における Modified PrefixSpan 法の動的負荷分散方式

本章では，既存の動的負荷分散方式について説明し，グリッド環境下での Modified PrefixSpan 法の動的負荷分散方式を提案する．

4.1 既存の動的負荷分散方式

本節では，分散型ワーカモデルで用いられている動的負荷分散方式について説明する．グリッド環境下では PC クラスタ内のタスクがなくなった場合，どの PC クラスタを選択してタスクを要求するかが重要な問題となる．表 2 に各動的負荷分散方式の問題点への対応を示す．

- *Random Steal*（RS）

通信遅延や負荷の偏りに関係なく，タスクがなくなった PC クラスタはランダムに他の PC クラスタを選択してタスクリクエストを送信する動的負荷分散方式を *Random Steal* と呼ぶ．負荷が偏った場合に通信回数が増加する可能性が高い．ランダムに PC クラスタを選択するため表 2 では，問題点への対応が両方 “×” となっている．

- *Cache-based Random Steal*（CRS）

RS とキャッシュ機能を組み合わせた動的負荷分散方式を *Cache-based Random Steal* と呼ぶ．RS でタスクを受け取ることができた場合，タスクを受け取ることができた PC クラスタの ID を記憶する．次回以降はその ID を持つ PC クラスタへタスクリクエストを送信する．RS と同様，通信遅延による性能低下は考慮していないが，極端な負荷の偏りに対応できる．

負荷が偏った場合，RS に比べ通信回数を抑えることができるため表 2 では負荷の偏りへの対応が “○” となっている．

- *Multicast Steal*（MS）

タスクがなくなると全ての PC クラスタへタスクリクエストを送信する動的負荷分散方式を Multicast Steal と呼ぶ．早く受け取ったタスクから順に処理していく．通信遅延が最も小さい PC クラスタから順にタスクを受け取ることができるので，通信遅延が大きな PC クラスタとの通信による待ち時間を隠すことができる．よって表 2 では通信遅延への対応は “○” となっている．しかしながら，毎回全ての PC クラスタにタスクリクエストを送信するため，タスクを持っていない PC クラスタへの無駄な通信が増加する．よって表 2 の負荷の偏りへの対応は “△” である．

- *Nearest Steal*（NS）

予め PC クラスタ間の通信遅延を測定しておき，どの PC クラスタと通信を行うか決めておく動的負荷分散方式を Nearest Steal と呼ぶ．通信遅延が一定の場合，最も通信遅延が小さい PC クラスタにタスクリクエストを送ることができる．しかしながら，通信遅延の変化に対応できないため表 2 では通信遅延への対応が “△” となっている．また，通信遅延のみを考慮してタスクリクエストを送信するため，負荷の偏りが極端な場合，通信回数が増加する可能性が高い．

4.2 Cache-based Multicast Steal 方式

4.1 で説明した動的負荷分散方式は，極端な負荷の偏りという Modified PrefixSpan 法の特徴と，グリッド環境下において発生する通信遅延の両方を考慮した動的負荷分散方式ではない．本研究では，Modified PrefixSpan 法の特徴と通信遅延の両方を考慮した動的負荷分散方式である *Cache-based Multicast Steal*（CMS）を提案する．CMS は MS と CRS の長所を組み合わせた方式である．PC クラスタ内の 1 台が外部の PC クラスタと通信を行う．外部の PC クラスタと通信を行う PC をグローバル PC と呼ぶ．以下に CMS におけるグローバル PC の処理手順を示す．また，図 2 に CMS の動作を示したものを示す．

（1） 全ての PC クラスタのグローバル PC に変数 Cache ID（CID）を与え，初期値を-1 に設定する．全てのグローバル PC が 1-頻出パターンを抽出する．

（2） グローバル PC はラウンドロビン法を用いて抽出した 1-頻出パターンから 2-頻出パターンを抽出する処理を 1 つの小粒度タスクとして自身のタスクプールに挿入する．

（3） PC クラスタ内の小粒度タスクが全てなくなるまで PC クラスタ内で動的負荷分散を行い処理を続ける．PC クラスタ内の小粒度タスクが全てなくなると（4）へ．

（4） CID の値を参照し，CID の値により以下の処理を行う．

（a） マルチキャスト（図 2（a））

CID の値が-1 ならば，グローバル PC は全ての PC クラスタのグローバル PC にタスクリクエストを送信する．(5) へ．

（b）キャッシュベース

CID の値が 1 以上ならば，CID に記憶されている ID を持っている PC クラスタのグローバル PC にタスクリクエストを送信する．(5) へ．

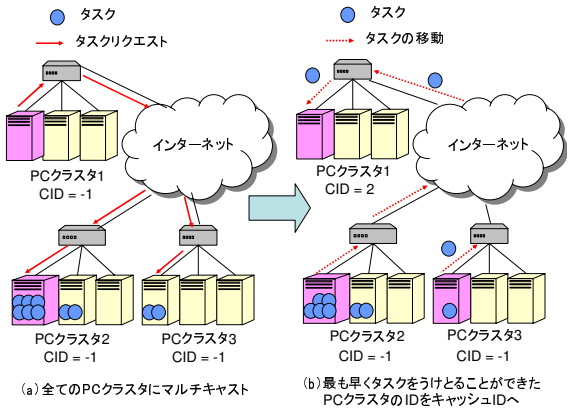

図 2 *Cache-based Multicast Steal*（CMS）の動作

（5） タスク要求メッセージの返事により次の処理をおこなう．(4) でマルチキャストを選択した場合は，手順（a）に進み，CID 先を選択した場合は手順（b）に進む．

（a）マルチキャスト（図 2（b））

タスク要求メッセージの返事として小粒度タスクが返ってきた場合，最も早く小粒度タスクを得ることができた PC クラスタの ID（1 以上）を CID に保存し，タスクをグローバル PC のタスクプールに挿入する．手順（3）に戻る．全ての PC クラスタからタスクを受け取れなければ，手順（6）へ．

（b）キャッシュベース

タスク要求メッセージの返事として小粒度タスクが返ってきた場合，小粒度タスクをタスクプールに挿入する．手順（3）に戻る．タスク要求メッセージの返事として小粒度タスクが返ってこなかった場合，CID を-1 に変更し，手順（4）に戻る．

（6） グローバル PC が全てのグローバル PC に，自身が所属している PC クラスタの処理を終了する合図を出し，PC クラスタ内の処理を全て終了する．

タスクリクエストを受け取った際の処理手順は 3.2 と同様であるので省略する．

CMS を用いると，通信遅延が最も小さい PC クラスタからタスクを受け取ることができる．最も早くタスクを受け取ることができた PC クラスタの ID をキャッシュし，マルチキャスト以降はキャッシュベースの動的負荷分散を行う．よって，通信遅延が少なく，小粒度タスクを保持している PC クラスタから小粒度タスクを受け取れる可能性が，既存の動的負荷分散方式と比較して大幅に増加する．

マルチキャストとキャッシュ機能を組み合わせることで，Modified PrefixSpan 法の特徴である極端な負荷の偏りと，PC クラスタ間の遅延による性能低下抑える動的負荷分散が可能となる．

5. 関 連 研 究

本章では，関連研究として，並列化モデルの視点から “階層的マスタワーカモデル” について，動的負荷分散の視点から

表 2　動的負荷分散方式と問題点への対応

問題点 / 動的負荷分散方式	負荷の偏り	通信遅延
Random Steal（RS）	対応できない（×）	対応できない（×）
Cache-based Random Steal（CRS）	対応できる（○）	対応できない（×）
Multicast Steal（MS）	対応できるが無駄な通信が増加（△）	対応できる（○）
Nearest Steal（NS）	対応できない（×）	対応できるが遅延が変化した場合対応できない（△）

“*Cluster-aware Random Stealing*” について述べる．

5.1　並列化モデルについて

グリッド環境下での並列化モデルとして，階層的マスタワーカモデル [12] が提案されている．1 台のスーパバイザがマスタワーカモデルで構成された複数の PC クラスタを統括するモデルである．階層的マスタワーカモデルは，特に分枝限定法のアプリケーションを対象としており，暫定解の同期による枝刈りを行い，処理の高速化を図っている．そのため，並列化モデルとしては暫定解の同期を取りやすいマスタワーカモデルが適している．しかしながら，階層的マスタワーカモデルを直接 Modified PrefixSpan 法のグリッド環境下での並列化モデルとして適用すると以下のような問題点がある．

階層的マスタワーカモデルでは，各 PC クラスタはスーパバイザのみと通信を行うので，PC クラスタとスーパバイザの間が距離的に遠ければ，通信遅延による性能低下が著しい．たとえば，PC クラスタ同士が距離的に近くに存在しても PC クラスタ同士で通信を行わず，スーパバイザを介して通信を行うので PC クラスタ間とスーパバイザ間の通信遅延が大きく影響する．階層的マスタワーカモデルでは，スーパバイザをどこに配置するのかといったネットワークトポロジーが重要な問題となる．

一方，分散型ワーカモデルでは，PC クラスタが散在していても PC クラスタ同士で通信を行うことができるので，通信遅延の影響を受けにくい PC クラスタを選択してタスクを受け取ることができる．分散型ワーカモデルではネットワークトポロジーを意識する必要はない．

本研究で扱う Modified PrefixSpan 法は PC クラスタ間でタスクの同期をとる必要はなく，PC クラスタ間の負荷の偏りが極端なアプリケーションである．そのため，PC クラスタ間でタスクの移動を積極的に行わなければならない．よって，本研究では分散型ワーカモデルをグリッド環境下での並列化モデルとした．

5.2　動的負荷分散方式について

マルチクラスタ環境で分散型ワーカモデルを用いた動的負荷分散方式として *Cluster-aware Random Stealing* [13] が提案されている．*Cluster-aware Random Stealing* はある PC のタスクがなくなると PC クラスタ内と PC クラスタ間の両方にタスクリクエストを送信する．PC クラスタ内と PC クラスタ間の両方にタスクリクエストを送信することで，PC クラスタ間の通信遅延による待ち時間を隠すことができる．また，他の PC クラスタからタスクを受け取るまでは PC クラスタ内にのみタスクリクエストを送信することで PC クラスタ間の通信回数を削減している．

Modified PrefixSpan 法の動的負荷分散方式として *Cluster-aware Random Stealing* を用いると，PC 間の極端な負荷の偏りが原因で，PC クラスタ間のタスクリクエスト数が増加する．Modified PrefixSpan 法は，PC クラスタ間だけでなく PC クラスタ内でも負荷の偏りが極端である．タスクを処理し終えた PC が毎回 PC クラスタ間のタスクリクエストを送信すると極端に PC クラスタ間のタスクリクエスト数やタスクの移動回数が増加し，通信によるオーバヘッドが大きくなる．よって，本研究では PC クラスタ内と PC クラスタ間の動的負荷分散の連携をとり，極端な負荷の偏りと通信遅延を考慮した *Cache-base Multicast steal* を用いている．

6. 性 能 評 価

グリッド環境下における Modified PrefixSpan 法の動的負荷分散方式として CMS，MS，CRS，RS，NS を実装し，性能評価を行った．今回の性能評価では，PC クラスタ内の並列化モデルとしてマスタワーカモデルを使用した．

6.1　実 験 環 境

実験には 4 台の PC クラスタを 3 つ，計 12 台の計算機を用いた．仮想的なグリッド環境を構築するために，PC クラスタ間に遅延機能を備えたプログラム（DummyNet [14]）を搭載した PC を配置した．3 つの PC クラスタ間に DummyNet を配置することによって，PC クラスタ間の通信の遅延を人工的に発生させ，動的に変化させることができる．

各計算機の性能は全て同じであり，次に示す通りである．それぞれ，Intel Pentium4 プロセッサ 2.8GHz，1.0GB メモリを搭載し，OS は Fedora Core 2 を使用している．各計算機は，PC クラスタごとに 1000 Mbit/sec イーサネットスイッチで接続され，通信にはソケット通信と MPICH の version 1.2.6 を MPI ライブラリとして使用した．DummyNet に用いた計算機の性能は次の通りである．Intel Pentium4 プロセッサ 2.53GHz，1.5GB メモリを搭載し，OS は FreeBSD 5.3-RELEASE を使用している．

6.2　配列データ

この実験で使用した配列データは PROSITE [1] が提供している，Kunitz というモチーフを含む配列データ（以下，Kunitz データセットと呼ぶ），Leucine Zipper というモチーフを含む配列データ（以下，Leucine データセットと呼ぶ）を使用した．

Kunitz データセットの詳細は，データ件数：70 件，総長：23385 バイト，平均長：334 バイト，最大長：3176 バイト，最小長：53 バイトである．Leucine データセットの詳細は，デー

タ件数：370 件，総長：139422 バイト，平均長：376 バイト，最大長：3224 バイト，最小長：3 バイトである．

6.3 評　　価

PC クラスタ 1 と PC クラスタ 3 の間の通信遅延を 100ms，PC クラスタ 2 と PC クラスタ 3 の間を 200ms，PC クラスタ 1 と PC クラスタ 2 の間を 100ms から 500ms まで 100ms ずつ変化させて実験を行った．

6.3.1 Kunitz データセットでの性能評価

図 3 に Kunitz データセットで評価を行ったときの処理時間を，図 4 に全体の通信回数を，図 5 に総通信量の比較を示す．グラフの横軸は PC クラスタ 1 と PC クラスタ 2 の間の通信遅延を示し，縦軸は処理時間，通信回数，総通信量を示している．

図 3 より，NS は他の全ての方式に比べ通信遅延を増加させると処理時間も大幅に増加している．NS に続き，MS，RS，CRS の順に性能が低下している．図 4 より，NS が最も多くメッセージを送信していることが分かる．NS は通信遅延が最も小さい PC クラスタから順位タスクリクエストを送信するが，負荷の偏りを考慮していないため，タスクがない PC クラスタに何度もタスクリクエストを送信したことがメッセージ数の増加に繋がっている．図 5 に示すように，NS の通信量が通信回数に比例していないことから，タスクを受け取れないタスクリクエストが多く発生している．

RS はランダムであるため，極端な性能低下は見られないが通信遅延による性能低下を抑えることができていない．図 4，5 より，通信回数や通信量がまばらであり遅延の状況に柔軟に対応できていない．

MS は NS と比較すると通信遅延による処理時間の増加を抑えられている．しかし，通信遅延の増加の影響により処理時間が増加している．最も早く受け取ったタスクから処理を行えるが，毎回全ての PC クラスタにタスクリクエストを送信するので，通信回数は少ないが（図 4），タスクがない PC クラスタへの無駄な通信が生じる．タスクを受け取るまでタスクの処理ができないので，負荷が偏った場合，通信回数・通信量が増加する．図 5 より，MS はほぼ通信量が最も多い．

CRS は RS と比較するとキャッシュ機能により，通信回数・通信量は減少している．しかしながら，遅延の増加に伴い，処理時間も増加している．キャッシュ機能のみでは通信遅延に対応できていないことが分かる．

図 3 より，CMS は通信遅延を変化させてもほぼ一定の時間で処理を終えている．PC クラスタ 1 と PC クラスタ 2 の間の通信遅延を増加させているにも関わらず，処理時間が増加していないのは，PC クラスタ 3 に極端に負荷が偏っているために，遅延を変化させている PC クラスタ間の通信がほとんど発生していないためである．

Kunitz データセットにおける性能評価では，CMS が最も通信遅延による性能低下を抑えられていた．

6.3.2 Leucine データセットによる性能評価

図 6 に Leucine データセットで評価を行ったときの処理時間を，図 7 に全体の通信回数を，図 8 に総通信量の比較を示す．

図 6 より，Kunitz データセットでの性能評価と同様，CMS は通信遅延を増加させても他の方式と比べ，大幅に処理時間が増加していない．図 8 より，CMS の通信量も他の方式と比べ最も抑えられている．CMS 以外の方式では，通信遅延を増加させると処理時間も大幅に増加している．

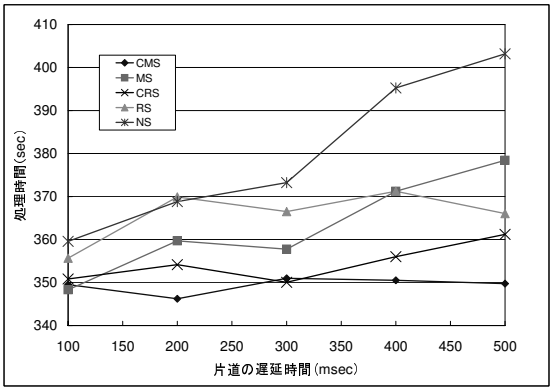

図 3　処理時間（Kunitz データセット）

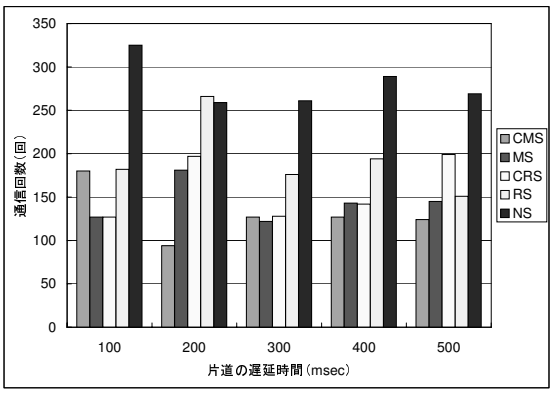

図 4　通信回数（Kunitz データセット）

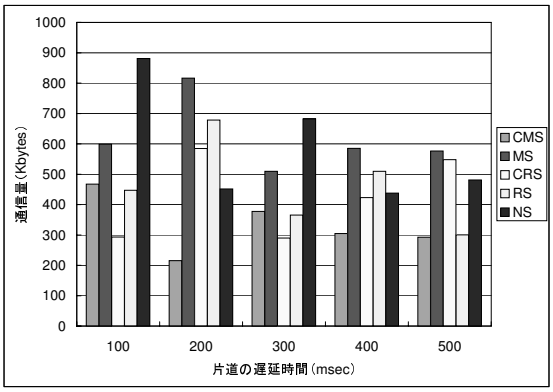

図 5　総通信量（Kunitz データセット）

評価実験の結果，CMS がグリッド環境下での Modified PrefixSpan 法の動的負荷分散方式において最も通信遅延に対応できた方式であった．

7. おわりに

本研究では，グリッド環境下での並列化モデルとして分散型ワーカモデルを用いた．また，グリッド環境下での Modified

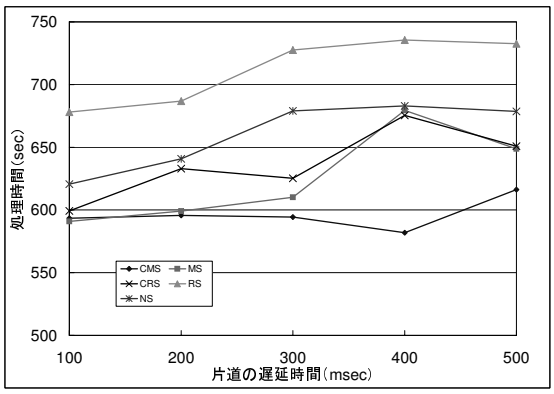

図 6　処理時間（Leucine データセット）

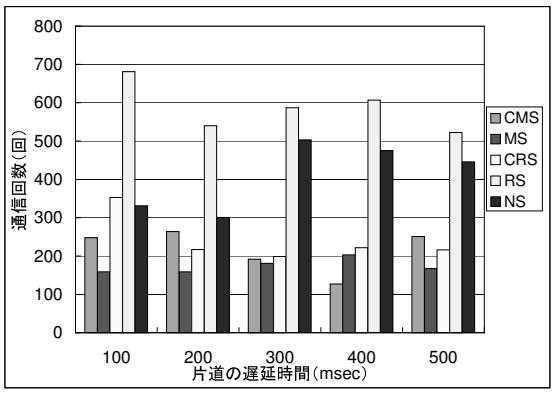

図 7　通信回数（Leucine データセット）

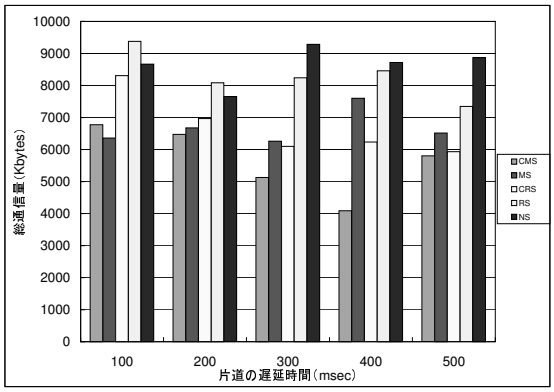

図 8　総通信量（Leucine データセット）

PrefixSpan 法の動的負荷分散方式として *Cache-based Multicast Steal*（CMS）を提案した．CMS により，PC クラスタ間の通信遅延による性能低下を抑えることができ，Modified PrefixSpan 法の特徴である負荷の極端な偏りを解消させることができた．

今後の課題として，パラメータや通信遅延を変化させての性能評価，より多くの PC クラスタでの性能評価，実際のグリッド環境下での性能評価が挙げられる．さらに，PC クラスタ内の PC 数や PC の性能も考慮した動的負荷分散方式の開発を行っていく予定である．

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）（一般），課題番号：17500097），広島市立大学・特定研究費（一般研究費（コード番号： 3106）），文部科学省・科学研究費補助金（課題番号：16700114）の支援により行われた．

文　　献

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

[2] http://www.ddbj.nig.ac.jp/.

[3] Hajime Kitakami, Tomoki Kanbara, Yasuma Mori, Susumu Kuroki, and Yukiko Yamazaki. Modified PrefixSpan Method for Motif Discovery in Sequence Databases. In *PRICAI2002, Proceedings of Trends in Artificial Intelligence, 7th Pacific Rim International Conference on Artificial Intelligence*, Vol. 2417 of *Lecture Notes in Computer Science*, pp. 482–491. Springer, 2002.

[4] 塔野薫隆, 北上始, 田村慶一, 森康真, 黒木進. Modified PrefixSpan 法を用いた頻出正規パターンの抽出をめざして．日本データベース学会 Letters, Vol. 3, No. 1, pp. 61–64, 2004.

[5] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE Trans. Knowl. Data Eng.*, Vol. 16, No. 11, pp. 1424–1440, 2004.

[6] Makoto TAKAKI, Keiichi TAMURA, Tosihide SUTOU, and Hajime KITAKAMI. Dynamic Load Balancing for Parallel Modified PrefixSpan. In *Proceedings of The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA 2004)*, pp. 352–358. CSREA Press, 2004.

[7] 高木允, 田村慶一, 周藤俊秀, 北上始. 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. SIG 10 (TOM12), pp. 138–152, 2005.

[8] 高木允, 田村慶一, 周藤俊秀, 北上始. 分散型ワーカモデルによる Modified PrefixSpan 法の並列処理とその動的負荷分散手法. 電子情報通信学会第 16 回データ工学ワークショップ, 2005.

[9] Makoto Takaki, Keiichi Tamura, Toshihide Sutou, and Hajime Kitakami. New Dynamic Load Balancing for Parallel Modified PrefixSpan. In *ICDE Workshops*, pp. 96–99, 2005.

[10] 周藤俊秀, 高木允, 田村慶一, 北上始. グリッド環境下における Modified PrefixSpan 法の並列処理とその動的負荷分散方式. 先進的計算基盤システムシンポジウム (SACSIS2005)，情報処理学会, pp. 161–168, 2005.

[11] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. A Comparison of Receiver-initiated and Sender-initiated Adaptive Load Sharing (extended abstract). In *Proceedings of the 1985 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pp. 1–3. ACM Press, 1985.

[12] Kento Aida, Wataru Natsume, and Yoshiaki Futakata. Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm. In *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pp. 156–163. IEEE Computer Society, 2003.

[13] Rob V. van Nieuwpoort, Thilo Kielmann, and Henri E. Bal. Efficient load balancing for wide-area divide-and-conquer applications. In *PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pp. 34–43. ACM Press, 2001.

[14] 林也寸夫, 馮富久, 荒川美穂, 野口隆史, 鷹見成一郎. FreeBSD Expert 2003. pp. 260–271, 2002.