マルチコア CPU 上における文書ストリーム上のバースト検出手法
Burst Detection Method for Document Stream on Multi-core CPU

平原　海詞*　田村　慶一**　北上　始**　田村　真吾*
Kaishi Hirahara　Keiichi Tamura　Hajime Kitakami　Shingo Tamura
広島市立大学大学院情報科学研究科
Email:*{ mu67026, mw67022}@edu.ipc.hiroshima-cu.ac.jp, **{ktamura, kitakami}@hiroshima-cu.ac.jp

Abstract—**Online documents on the social media sites are represented as a document stream, because the documents have a temporal order. This has resulted in numerous studies on extracting a frequent phenomenon (involving, e.g., keywords, users, and locations) known as a burst. In this paper, we propose a novel parallelization method for the parallel processing of Kleinberg's burst detection algorithm for a large-scale document stream. The proposed parallelization method combines the inter-task with the intra-task parallelization model. A combination of inter- and intra-task parallelization can achieve seamless dynamic load balancing, and detect a burst in a large-scale document stream on memory.**

I.　はじめに

近年，ソーシャルメディアへの関心の高まりとともに，ソーシャルメディアのサイト上で生成される文書データは集合的な知識を有するようになっている．特に，ソーシャルメディアの文書データから特徴的なパターンを抽出することは，マーケティング，観光情報学，検索エンジンの機能強化や社会的な話題の分析などに必要不可欠であり，重要な課題となっている．

そこで，ソーシャルメディアのサイト上で生成される文書データを，時系列に到着するストリームデータ（以下，文書ストリームと呼ぶ）として扱い，文書ストリーム上に頻繁に現れてくる事象（ユーザ，語句や場所など）をバーストとして検出する手法が盛んに研究されている[1],[2]．バーストとは，ある事象の出現頻度が通常の出現頻度と比較して急激に増加していることを示す指標である．バーストを検出することができれば，ユーザが着目している事象を検出することができ，イベントやトピックを抽出することができる．

文書ストリーム上に現れる語句のバーストを検出する手法として，Kleinberg のバースト検出アルゴリズムが提案されている[3]．近年，ソーシャルメディアのサイト上で生成される文書データは指数関数的に増加しており，Kleinberg のバースト検出アルゴリズムを適用すると非常に処理時間を必要とすることが課題となっている．Kleinberg のバースト検出アルゴリズムでは語句の発生回数×状態数の領域が必要となる．通常，各語句の発生回数は偏りが多く，例えば，数カ月の期間において，数十万以上発生する語句から数十件しか発生しない語句まで発生回数の差がある．

そこで，本論文は，Kleinberg のバースト検出アルゴリズムに焦点をあて，マルチコア CPU 上での並列化手法を提案する．本論文で提案する手法の特徴は以下の通りである．

(1) 文書ストリーム上に現れる各語句に対してバーストを検出することをタスクと定義する．また，このタスクを複数のコアで実行することをタスク間並列化と定義する．ここで，マルチコア CPU 上のコア間で競合が発生しないようにするために，各タスクを複数の区間に分割し，分割したタスクを同時に処理することをタスク内並列化と定義する．

(2) タスク間並列化にタスク内並列化を併用する．語句の発生回数が閾値 σ 未満の場合は，タスクをそのまま実行し，語句の発生回数が閾値 σ 以上の場合は，当該タスクはタスク内並列化で並列処理する．分割時に分割する系列の幅を調整することで，メインメモリやキャッシュのサイズを意識して処理を行うことができる．

(3) タスク内並列化とタスク内並列化が同時に動いたとしてもシームレスに動作させるために，タスク内並列化で分割したタスクもひとつのタスクとして扱う．通常のタスクを通常タスクと呼び，分割後のタスクを分割タスクと呼ぶ．

提案手法を，マルチコア CPU を搭載した PC 上で実装し，2009 年 6 月から 2009 年 12 月までに Twitter でつぶやかれたツイート 128 万件を用いて評価実験を行った．評価実験の結果，提案手法の有効性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 章では，バーストと Kleinberg のバースト検出アルゴリズムを説明する．第 3 章では問題設定と提案手法である並列化手法について述べる．第 4 章で性能評価実験の実験結果を示し，第 5 章で本論文のまとめを行う．

II. バースト検出

文書ストリームとは，文書データ doc_t が到着した後，x_t の間隔において次の文書データ doc_{t+1} が到着する時系列の文書データ集合からなるストリームデータのことを示す．社会的に関心の高いトピックが発生すると，そのトピックを表す語句を含む文書データが次々と生成される．そして，生成される文書データ間の到着間隔は短くなる．到着間隔が短くなっていることをバーストと呼ぶ．

Kleinberg のバースト検出アルゴリズムでは，文書データの到着間隔 x_i は隠れマルコフモデルの内部状態に応じて確率的に出力される記号であるとみなす．そして，s_1 から s_m までの m 個の状態を持つ隠れマルコフモデルを仮定する．ここで，個々の到着間隔 x_i においてそれぞれの状態 s_i であるコストと各状態の状態遷移のコストを定めると，文書到着間隔列 $x = (x_1, x_2, ..., x_n)$が与えられた時，次のコスト式を最小にする最適の状態遷移列 $s=(s_1,s_1,..,s_n)$ (各 s_1 は状態番号を表す)を求める問題として定義される．

$$C(s|x) = \left(\sum_{i=1}^{n-1} \tau(s_i, s_{i+1})\right) + \left(\sum_{i=1}^{n} -ln\, f_{s_i}(x_i)\right).$$

ここで，この式の第一項は，内部状態が状態 s_i ら s_{i+1} に遷移する際のコストの総和であり，関数 τ は状態 i から状態 j に遷移に必要となるコストを返す関数であり，次のように定義される．

$$\tau(i,j) = \begin{cases} (j-i)\gamma, & if\ j > i, \\ 0, & otherwise. \end{cases}$$

上記の式では，高い状態への状態遷移はユーザが指定したパラメータ γ に比例したコストが必要となり，低い状態への状態遷移のコストは 0 となっている．

また，関数$f_k(x_i)$は状態 k で到着間隔 x_i を発生するために必要なコストであり，第二項はその総和となる．

$$f_k(x_i) = \lambda_k e^{-\lambda_k x_i}.$$

ここで， λ_k は以下のように定義される．n は観測時間 T に到着した文書データの数であり，n/T は単位時間当たりの文書データ数を示す．

$$\lambda_k = \frac{n}{T}\beta^k.$$

$C(s|x)$ を最小とする最適の状態遷移列 $s=(s_1,s_2,..,s_n)$は，隠れマルコフモデルに対するビタビアルゴリズムを用いて次の動的計画問題を解くことで求めることができる．

$$C_j(i) = -\ln f_j(x_i) + \min_l(C_l(i-1) + \tau(l,j)).$$

Kleinberg のバースト検出アルゴリズムは動的計画問題となり，$O(n\times m)$のメモリ領域が必要となる．そのため，系列数が増加すると，非常に大量のメモリ領域を必要とする．

III. 提案手法

本章では，問題定義，タスク間並列化とタスク内並列化のそれぞれの並列化モデル，タスク間並列化にタスク内並列化を併用する手法について説明をする．

A. 問題定義

文書ストリーム $DS=\{doc_1,doc_2,...,doc_n\}$上の文書データ doc_i はその文書が到着した時刻 $altime_i$，テキストデータ $text_i$ の二つの要素から構成され，$doc_i=<altime_i,text_i>$と表す．語句ごとに語句を含む文書データの到着時刻のみを取り出して到着時刻ごとに並べ変えた系列を，

$$TALT_i = \left(talt_{i,1}, talt_{i,2}, \dots, talt_{i,|c_i|}\right), talt_{i,j} \in ALT \quad ,$$

と表記する．ただし，ALT は $ALT=\{altime_1, altime_2, ..., altime_n\}$と，すべての到着時刻の集合である．本研究では文書ストリーム上の m 個の語句について，$TALT_i$が与えられたときに，すべての語句について状態遷移列を Kleinberg のバースト検出アルゴリズムで求める処理の並列化を目指す．

B. 並列化モデル

本節では，提案手法の基本的となるタスク間並列化とタスク内並列化を説明する．

1) タスク間並列化

並列化において，ひとつの語句のバーストを検出する処理をひとつのタスクと定義する．タスク並列化の処理手順は以下の通りである．

(1) 各語句の到着間隔列$TALT_i$すべてをタスクプールにタスクとして挿入する．

(2) 各コアは，タスクプールからタスクを取り出し，到着間隔列$TALT_i$を入力として，状態遷移列 s を求める．

2) タスク内並列化

各到着間隔$TALT_i$を以下のように p 個の区間$PTALT_i^k$に分割する．

$$PTALT_i^k = \left(talt_{i,d\times k}, talt_{i,d\times k+1}, \dots, talt_{i,d\times k+d-1}\right),$$
$$talt_{i,j} \in ALT, 1 \le \mathrm{k} \le \mathrm{p}$$

ただし，$d = \frac{|TALT_i|}{p}$である．

次に，区間 $PTALT_i^k$に対して，部分状態遷移列 s^kを取り出し，すべての部分状態遷移列 s^kを順番に併合すると状態遷移列 s となる．

タスク内並列化の処理手順は次の通りである．

(1) 各語句の到着間隔列$TALT_i$すべてをタスクプールに挿入する．

(2) タスクプールからタスクを取り出し，到着間隔列$TALT_i$を p=（PE の数）個の区間に分割する．そして，k 番目のコアに入力として$PTALT_i^k$を与え，部分状態遷移列 s^kを求める．

(3) 部分状態遷移列 s^k を集め，順番に併合し，状態遷移列 s とする．語句 $term_i$ と状態遷移列 s のペアを結果格納プールに挿入する．

C. タスク間にタスク内並列化を併用する並列化モデル

タスク間並列化は，単純な方法であるが，大規模な系列を対象とするとコアが一斉に大量のメモリ空間を消費するという課題がある．そこで，タスク間並列化にタスク内並列化を併用する．提案手法では，$|TALT_i| < \sigma$の場合，コアはタスクをそのまま実行するが，$|TALT_i| \geq \sigma$の場合，タスクを $range$ 区間ごとに$div = |TALT_i|/range$個に分割し，分割した div 個の区間に対して部分状態遷移列を求める処理を新たにタスクとして生成する．ここで，通常のタスクを通常タスク，分割後のタスクを分割タスクとする．

タスク間並列化にタスク内並列化を併用する手法についてのその処理手順を示す．

(1) 各語句の到着間隔列$TALT_i$すべてを通常タスクとして，タスクプールに挿入する．

(2) タスクプールからタスクを取り出す．

(ア) もし，タスクが通常タスクならば，$|TALT_i| < \sigma$の場合，到着間隔列$TALT_i$を入力として，Kleinberg のバースト検出アルゴリズムを用いて，状態遷移列 s を求める．$|TALT_i| \geq \sigma$の場合，タスクを $range$ 区間ごとに $div = |TALT_i|/range$個に分割し，分割した div 個の各パーティションに対して部分状態遷移列を求める処理を新たに分割タスクとして生成し，タスクプールに挿入する．

(イ) もし，タスクが分割タスクならば，パーティション$PTALT_i^k$を入力として，Kleinberg のバースト検出アルゴリズムを用いて，部分状態遷移列 s^k を求める．

次のタスクをタスクプールから取り出し実行を繰り返し，タスクプールのタスクが空になるまで処理を続ける．

IV. 評価実験

提案手法を，マスタ・ワーカモデルを用いてマルチコア CPU を搭載した PC 上で実装し，2009 年 6 月から 2009 年 12 月までに Twitter でつぶやかれたツイート 128 万件を用い，データに含まれる語句 100，1,000，10,000 件の系列を用いた．実験環境として， CPU：Phenom Ⅱ X6 1090T Six-Core/3.2G/L3：6MB，メモリ 4GB，OS：Ubuntu11.04 を搭載した PC を用いた．

図 1，図 2，図 3 に速度向上比の結果を示す．タスク間並列化と，タスク間並列化とタスク内並列化を併用した手法の結果を比べると，タスク数が 100，1000 の場合にはタスク間並列化とタスク内並列化を併用した手法が良いという結果が得られ，タスク数 10000 の場合にはタスク間並列化のほうが良いという結果が得られた．タスク数が多いケースでは、タスクの大きさの偏りによるスレッドごとの計算量の差が出にくいため，タスクを分割・併合する処理がオーバヘッドとなってしまい，良い結果が得られないのだと考えられる．

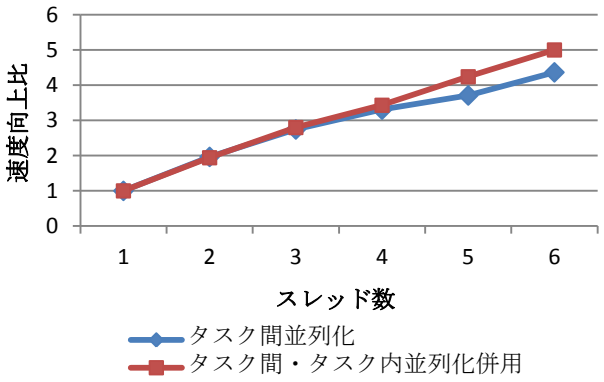

図 1　速度向上比（タスク数：100）

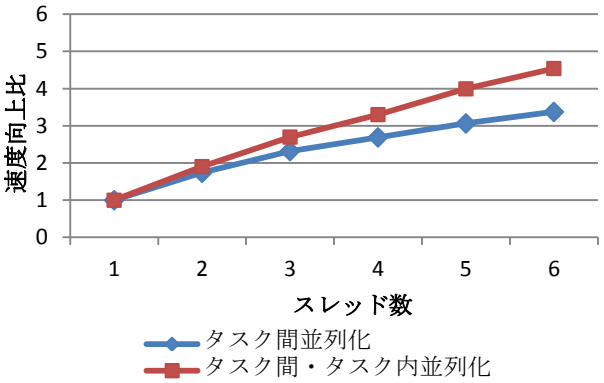

図 2　速度向上比（タスク数：1000）

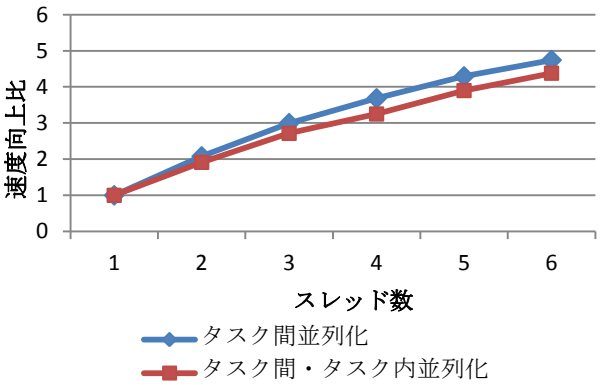

図 3　速度向上比（タスク数：10000）

図 4 にタスク数が 1000 の場合のスレッドごとの処理時間を示す．タスク間並列化では，タスクの大きさに偏りがあると，スレッドごとの処理時間にばらつきが生じ，結果的に全体としての処理時間が遅くなってしまう．しかしながら，タスク間・タスク内並列化を併用する手法では，タスクを分割しスレッドごとの負担を平均的に分散することで，スレッドごとの処理時間の差を減らすことができる．

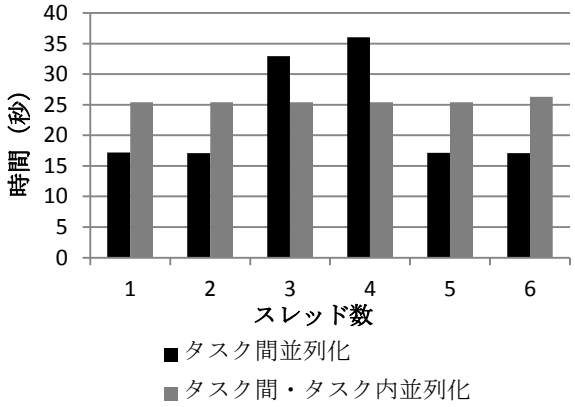

図 4　各スレッドの処理時間

表 1　大規模系列を使用した測定結果

スレッド数	タスク間並列化（秒）	タスク間・タスク内並列化併用（秒）
1	630.712922	644.104303
2	324.371184	326.789699
3	222.144265	223.178109
6	792.594904	130.757424

表 2　大規模系列を使用した CPU の待ち率

スレッド数	タスク間並列化	タスク間・タスク内並列化併用
1	0.253932	0.238033
2	0.242325	0.246182
3	0.240978	0.246493
6	0.330431	0.255649

次に，メモリ上で計算できる長さのほぼ上限となる系列長 230 万のデータを 24 件用意し，それぞれの手法について計測を行った．タスク間・タスク内並列化併用における分割長は 1000 とする．スレッド数ごとの実行時間の結果を表 1 に，CPU の待ち率を表 2 に示す．1 スレッドから 3 スレッドまでの処理については特に違いは見られないものの，同時に 6 スレッドが動く場合については，タスク間並列化ではタスクの競合が激しくなり，著しく性能が下がっている．一方，タスク間・タスク内並列化では，巨大なタスクについても任意の長さに分割をすることで，メモリの大きさに関わらず安定して処理を行うことができる．

V.　まとめ

本論文は，Kleinberg のバースト検出アルゴリズムに焦点を当て，大規模文書ストリームを対象としたバースト検出アルゴリズムのマルチコア CPU 上での並列化手法を提案した．タスク間並列化にタスク内並列化を併用することで，シームレスに動的負荷分散を行うことができ，かつオンメモリ上で大規模な文書ストリームにおけるバースト検出を行うことができる．提案手法を実装し評価実験を行った結果，タスク間並列化よりもタスク内・タスク間並列化を併用する手法のほうが良い結果が得られた．また，データの分割については，ある程度小さいタスクを用意するほうが速度向上比の点からも，メモリの大きさに限界がある点からも良いということが分かった．今後の課題としては，タスクを分割することによる精度低下と速度向上のトレードオフの関係について，最適となる点を検討していくことが挙げられる．

謝辞

本研究の一部は，文部科学省・科学研究費補助金(若手研究(B)，課題番号：23700124)，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)の支援により行われた．

参考文献

[1] J. Allan, R. Papka, and V. Lavrenko, “On-line new event detection and tracking,” in Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR ’98, pp. 37–45, 1998.

[2] J. Kleinberg, “Temporal dynamics of on-line information streams,” in IN DATA STREAM MANAGEMENT: PROCESSING HIGH-SPEED DATA, Springer, 2006.

[3] J. Kleinberg, “Bursty and hierarchical structure in streams,” in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD ’02, pp. 91–101, 2002.

問い合わせ先
〒731-3194
広島市安佐南区大塚東 3 丁目 4 番 1 号
広島市立大学　情報科学研究科　知能工学専攻
平原　海詞