3K4-1in1

オンライン頻出パターンマイニングの並列分散化に向けて

Towards Parallel Frequent Itemset Mining for Big Streaming Data

山本 泰生 *1*2　　岩沼 宏治 *1
Yoshitaka Yamamoto　　Koji Iwanuma

山梨大学大学院総合研究部
Graduate Faculty of Interdisciplinary Research, University of Yamanashi

科学技術振興機構，さきがけ
JST, Presto

Frequent itemset mining from transaction streams (FIM-TS) is one of the most fundamental tasks in streaming data mining. For the past decade, the task of FIM-TS has been intensively studied and several algorithms have been developed mainly in the context of one-pass approximation framework. However, they are not necessarily adequate for the era of big data: we are still required to drastically improve the scalability of the previously proposed FIM-TS methods to tolerate streaming *big transactions* which arrive at high speed, including many items. In this paper, we address this challenging issue by introducing the parallel distibuted process in the state-of-the-art FIM-TS method and report a preliminary result obtained by utilizing the OpenMP framework.

1. はじめに

本研究ではトランザクションストリーム上の頻出パターン (アイテム集合) をリアルタイム検出する問題 (*Frequent Itemset Mining from Transaction Stream*, 以下 FIM-TS と略す) を扱う FIM-TS では，高速に到着し続ける**トランザクション** (可変サイズのカテゴリ属性アイテムの集合) を逐次的に処理しながら，その中の出現パターンとその頻度情報を管理することが求められる．管理すべき出現パターンはトランザクションの任意の部分アイテム集合であり，FIM-TS ではパターンの組み合わせ爆発に関する本質的な難しさを有している．著者らは近年，圧縮技術を用いてこの組み合わせ爆発を部分的に緩和する 1 パス近似アルゴリズムを提案している [2, 3, 5]．提案アルゴリズムでは，**漸近交差** (*incremental intersection*) [1] と呼ばれる飽和アイテム集合の逐次更新法と**リソース指向近似** (*resource-oriented approximation*) と呼ばれる使用メモリの上限を設定した決定性近似計算 (*deterministic approximation*) [4] の 2 つの技法を利用している．

IOT 技術が急速に普及する今日，いわゆる**エッジヘビーデータ**を有効活用するストリーム処理技術がますます重要になっている．FIM-TS は直接的にはストリーム上で同時間帯に頻繁に発生するイベント集合をリアルタイムに検出する問題である．検出される共起イベント集合はストリームデータの素性とみなすことができ，データ全体のトレンドや異常を特徴づける説明変数として利用することができる．また FIM-TS が扱うパターンの組み合わせ爆発は，ストリームデータマイニングのタスク全般が克服すべき共通課題であり，これにアドレスする要素技術は FIM-TS 以外にも応用可能である．本稿では，著者らが提案している FIM-TS アルゴリズムの逐次処理を並列分散化することを検討する．逐次処理の主要な計算は，新規トランザクションとメモリに保持される各パターンの共通部分を求めることである．この計算はパターン毎に並列実行することが可能である．そこで，この共通部分計算を OpenMP を用いて分散並列化することでアルゴリズムの性能改善を図る．

連絡先: 〒 400-8510 甲府市武田 4-4-37 A3-K212, yyamamoto@yamanashi.ac.jp

2. 準備

$I = \{x_1, x_2, \ldots x_u\}$ をアイテムの全体集合とするとき，I の空でない部分集合を**アイテム集合**と呼ぶ．データストリーム $\mathcal{S}_n$ は，到着するトランザクションの列 $\langle t_1, t_2, \ldots, t_n \rangle$ として定義される．ただしトランザクション t_i は時刻 i に到着するアイテム集合であり，n は到着予定の (未知の) トランザクション数である．アイテム集合 α を含む $\mathcal{S}_n$ 中のトランザクションの個数を **頻度**と呼び $sup(\alpha, n)$ と表す．最小サポート σ $(0 \leq \sigma \leq 1)$ に対し，$sup(\alpha, n) \geq \sigma n$ を満たすとき，α を**頻出アイテム集合**と呼ぶ．FIM-TS では各時刻 i $(1 \leq i)$ におけるデータストリーム $\mathcal{S}_i$ の頻出アイテム集合をすべて抽出することがタスクとなる．いまトランザクションに含まれるアイテムの個数をそのトランザクションの長さと呼び，$\mathcal{S}_n$ の最大トランザクション長を L と表すと，解となりうる候補集合は少なくと 2^L 程度存在することになる．α をアイテム集合とする $\beta \supset \alpha$ を満たす任意のアイテム集合 β に対し，$sup(\alpha, n) \neq sup(\beta, n)$ のとき，α を**飽和アイテム集合**と呼ぶ

静的データベースを扱う問題とは異なり，ストリームデータ全体をメモリに保持し続けることは原理的に困難である．このため，FIM-TS では 1 パス型の近似アルゴリズムがよく利用されている [4]．このタイプの決定性アルゴリズムでは，任意のアイテム集合 α について，α の時刻 i における**見積り頻度** $c(\alpha, i)$ と**誤差計数** $\Delta(\alpha, i)$ の 2 つが与えられる．2 つのアイテム集合 α, β に対して，$\alpha \subseteq \beta$ かつ $c(\beta, i) - \Delta(\beta, i) \leq sup(\alpha, i) \leq c(\beta, i)$ を満たすとき，時刻 i において α は β に Δ **カバーされる**といい，$\alpha \preceq_\Delta \beta$ と表記する．Δ カバー関係は飽和性を一般化した概念であり，これを用いてアイテム集合族を圧縮することができる．F と T を 2 つのアイテム集合族とする．任意の $\alpha \in T$ に対して，$\beta \in F$ が存在し，$\alpha \preceq_\Delta \beta$ となるとき，T は F に Δ **カバーされる**という．いま時刻 i における頻出アイテム集合族を $FI(i)$ とすると，$FI(i)$ を Δ カバーするアイテム集合族 $DFI(i)$ が求まれば，$DFI(i)$ から任意の頻出アイテム集合を復元できる．また時刻 i における最大誤差計数を $\Delta(i)$ とすると，その頻出アイテム集合の頻度も $\Delta(i)$ の誤差範囲内で復元できる．すなわち $DFI(i)$ は頻度誤差を $\Delta(i)$ だけ許容した $FI(i)$ の非可逆圧縮とみなすことができる．

3. 漸近交差法とリソース指向近似

時刻 i までに出現した全飽和アイテム集合族を $CI(i)$ とすると，$CI(i+1)$ は次式で与えられる [1].

$$CI(i) \cup \{t_{i+1}\} \cup \{\beta \mid \beta = \alpha \cap t_{i+1},\ \beta \neq \emptyset,\ \alpha \in CI(i)\}.$$

本稿では，この漸化式を用いて逐次的に飽和アイテム集合族を更新する手法を**漸近交差** (*incremental intersection*) と呼ぶ．Borgelt らは [1] 漸近交差を用いて頻出飽和アイテム集合族を求める厳密解法を提案している．また著者らは [2, 3, 5]，漸近交差に決定性近似計算を導入した 1 パス近似アルゴリズムを提案しており，その出力は頻出アイテム集合族を Δ カバーする $CFI(i)$ となることが保証されている．以下に文献 [5] で用いられたリソース指向型の近似アルゴリズムを示す．

Algorithm 1 RO-ComStream algorithm

Input: サイズ定数 k，最小サポート σ，ストリーム $\mathcal{S}_n$
Output: $FI(n)$ Δ カバーする $CFI(n)$

```
1:  set i as 1 (i := 1)                          ▷ 現時刻 i
2:  Δ(0) := 0                                    ▷ i における最大誤差計数 Δ(i)
3:  initialize T_0
4:  while i ≤ n do read t_i
5:      T_i := intersectTraverse(T_{i-1}, t_i)
6:      while |T_i| > k do
7:          m := getMin(T_i)                     ▷ 最小エントリのアイテム集合
8:          Δ(i) := c(m, i)                      ▷ Δ(i) の更新
9:          delete(T_i, m)                       ▷ m のエントリの削除
10:     end while
11:     i := i + 1
12: end while
13: for each itemset α in T_n s.t. c(α, n) > σn do
14:     output α                                 ▷ α を出力する
15: end for
```

```
1:  function IntersectTraverse(T_i, t_{i+1})
2:      initialize C                             ▷ 候補エントリの集合 C
3:      id := get(t_{i+1}, T_i)                  ▷ t_{i+1} に対応する T_i 中のエントリ
4:      if id is null then
5:          add the entry ⟨ id_new, t_{i+1}, Δ(i), Δ(i) ⟩ to T_i
6:      end if
7:      for each entry id in T_i do
8:          β := α(id, i) ∩ t_{i+1}              ▷ 共通部分計算
9:          id_β := get(β, C)                    ▷ β に対応する C 中のエントリ
10:         if id_β is null then
11:             add ⟨ id_new, β, c(id, i) + 1, Δ(id, i) ⟩ to C
12:         else if c(id_β, i) < c(id, i) + 1 then
13:             c(id_β, i) := c(id, i) + 1, Δ(id_β, i) := Δ(id, i)
14:         end if
15:     end for
16:     for each entry id_C in C do
17:         id_{T_i} := get(α(id_C, i), T_i)
18:         if id_{T_i} is null then
19:             add the id_C entry to T_i
20:         else
21:             replace the id_{T_i} entry with the id_C entry
22:         end if
23:     end for
24:     return T_i                               ▷ 次の時刻の頻度表に相当する
25: end function
```

提案アルゴリズム (以下，RO-ComStream と呼ぶ) では，3 つ組 $\langle \alpha, c(\alpha, i), \Delta(\alpha, i) \rangle$ を一つのエントリとしてメモリ内に保持している．エントリ全体を**頻度表**と呼び，時刻 i の頻度表を T_i と書く．毎時刻 i において新規トランザクション t_i を呼び込み，頻度表の更新を行う (漸近交差)．この更新処理は 5 行目で呼び出される *intersectTraverse* 関数により実現される．また更新後の T_i のエントリサイズが，事前に与えられたサイズ定数 k を上回る場合，見積り頻度が最も小さいものから順にエントリを削除し頻度表を縮退させる (リソース指向近似)．また削除されたエントリ内の見積り頻度を用いて最大誤差計数 $\Delta(i)$ を更新する (6-10 行目に相当)．

先行研究 [5] では，IBM マーケットバスケットデータ生成器による合成データ (アイテム種類数 24,000，ストリーム長 10,000) を用いて，最大トランザクション長 L に対する RO-ComStream のスケーラビリティを確認している．図 1 と図 2 はそれぞれ頻度誤差率 $\frac{\Delta(n)}{n}$ と平均更新処理時間の推移を示している．

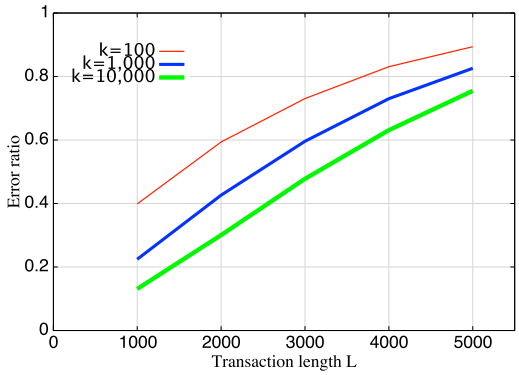

図 1: L に対する頻度誤差率 (%)

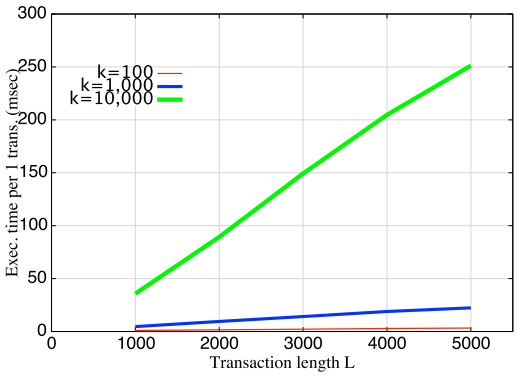

図 2: L に対する平均更新処理時間 (msec)

頻度誤差率と更新処理時間ともに L に対して線形的に増加していることがわかる．ただし，頻度表サイズ k に対する挙動はそれぞれで異なる．更新処理時間では k に比例して増加している．時間コストの大部分を占める処理は共通部分計算を行う際の頻度表走査である点からこの結果は妥当といえる．他方，k が増加する頻度誤差率は減少する．この結果は，登録エントリ数 k が増加することで頻度表の縮退処理の回数が相対的に減少したためと解釈できる．

4. 逐次更新処理の並列分散化

より大きなトランザクション長 L をもつストリームデータへの応用を考えたとき，誤差率と更新処理時間の 2 点を抑制

することが必要となる．ただし，これらは頻度表サイズ k に対してトレードオフの関係にある．そこで本章では，最も時間コストが大きい漸近交差法の逐次更新を並列分散化することを検討する．基本となるアイデアはシンプルである．すなわち新規トランザクションとの各エントリの共通部分計算が独立に実行できる性質を利用する．いま利用可能なノード数 n とすると，各ノードは n 分割された頻度表の各パートの更新作業を並列して実行することができる (図 3 参照)．

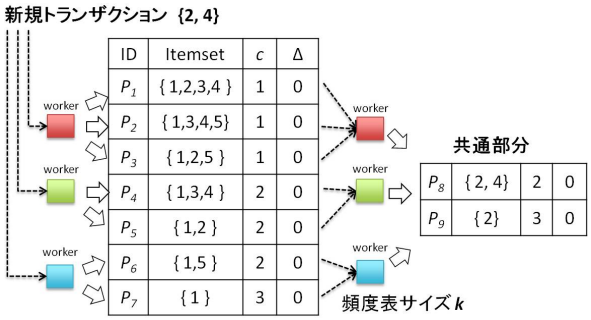

図 3: 共通部分計算の並列分散化

5. 予備実験結果

この頻度表更新の並列分散化処理を OpenMP を用いて実装し，先行研究 [5] のベンチマーク問題を用いて性能を評価した．実験には Xeon E5-2698v3 (32 コア) を利用している．はじめに各スレット数における台数効果 (実行時間 1 スレッド時の実行時間) を図 4 に示す．ただし $k = 100,000$ としている．

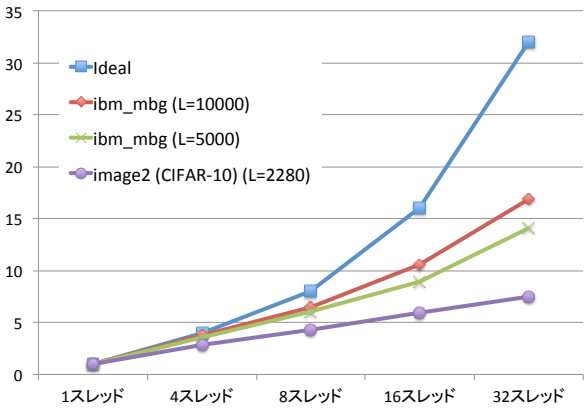

図 4: 実行時間に対する台数効果

スレッド数倍だけ速度向上する理想には届いていないものの，32 スレッドまで線形以上の効果が確認できる．特にトランザクション長が大きいベンチマーク問題において，その効果はより顕著になっている．次の表は，32 スレッド並列実行したときの $k = 100,000$ と $k = 1,000,000$ における誤差率 (%) と平均更新時間 (msec) ならびにメモリ使用量 (MB) を示している．更新時間とメモリ使用量ともに k に比例して増加しているのに対し，誤差率は限定的であるものの (特に ibm_mbg) 若干改善された．ibm_mbg は合成データであり，実データを用いて検証する必要はあるが，頻度表サイズを単純に増加するだけでは誤差率を十分抑えることが難しいことがわかる．

$k = 100,000$	誤差率	更新	メモリ
ibm_mbg ($L = 10,000$)	95.9	41.2	1,426
ibm_mbg ($L = 5,000$)	96.5	37.0	1,137
image2 ($L = 2,280$)	10.8	18.2	510
$k = 1,000,000$	誤差率	更新	メモリ
ibm_mbg ($L = 10,000$)	94.5	446.6	10,955
ibm_mbg ($L = 5,000$)	95.5	406.5	8,952
image2 ($L = 2,280$)	6.7	220.4	1,225

表 1: マルチスレッド環境における性能

6. まとめと今後の課題

本稿では，ストリームデータから頻出アイテム集合を抽出するオンラインアルゴリズムの並列分散化を検討し，マルチスレッド環境化における性能を示した．スレッド数に対する台数効果を確認するとともに，大規模な頻度表を用いた場合の誤差率と更新処理時間を調査した．リアルタイムな応答を求められるストリームデータ処理において，並列分散化による速度改善は意義深い．他方，解の精度を決定付ける誤差率の観点からいえば並列化の効果は十分といえない．マルチスレッド環境下において誤差率上昇をいかに抑制するか検討したい．また大規模なクラスタ環境化での性能評価も今後の課題である．

謝辞

本研究では，JST さきがけおよび ISPS 科学研究費補助金 (No. 25330256) の援助を受けている．また OpenMP の実装ならびに性能評価の実験に際して，HPC システムズ株式会社様より技術サポートを頂いた．

参考文献

[1] C. Borgelt, X. Yang, R. N. Cadenas, P. C. Saez and A. P. Montano: Finding closed item sets by intersecting transactions, *Proc. of Int. Conf. on EDBT*, 367–376 (2011).

[2] S. Fukuda, K. Iwanuma and Y. Yamamoto: An online frequent closed itemset mining, *Proc. of SIG-FPAI-B404-01*, 1-6 (in Japanese, 2015).

[3] K. Iwanuma, Y. Yamamoto and S. Fukuda: An on-line approximation algorithm for mining frequent closed itemsets based on incremental intersection, *Proc. of Int. Conf. on EDBT*, to appear (2015).

[4] Y. Yamamoto, K. Iwanuma and S. Fukuda: Resource-oriented approximation for frequent itemset mining from bursty data streams, *Proc. of Int. Conf. on SIGMOD'14*, 165-179 (2014).

[5] Y. Yamamoto and K. Iwanuma: Online-pattern mining for high-dimensional data streams, *Proc. of Int. Conf. on Big Data*, 2880–2882 (2015).