Vol.2014-HPC-146 No.11
2014/10/2

ストリーム解析処理に特化した挙動解析ツールの構築

秋岡　明香[1,a)]

概要：大量のデータを横断的かつ網羅的に解析することで、新しい知見やニーズを発見しようとするビッグデータ解析が注目を集めている。こうした需要や期待の高まりに応じて、ビッグデータ解析処理を高速化かつスケールアウトする新たな計算機環境への要望も生じているが、ビッグデータ解析処理の挙動については、未だ明確になっていない。ビッグデータ解析処理はその特徴によっていくつかの種類に分類することができるが、本稿では、そのうちのストリーム解析処理に注目する。筆者はストリーム解析処理をモデル化するためのツール群と、これらのツールによってストリーム解析処理に特化したタスクグラフを生成するための StratStream (Standard Task gRAph Toolkit for STREAM mining applications) の研究開発を行なっているが、ストリーム解析手法の多くは、入力データの特徴や偏りによって実行時間に大きな分散が生じることや、同じ手法でも、入力データによって出力結果が異なるという指摘を多く受け、StratStream のようなツールキットには、代表的な入力データや入力データモデルを含めるべきだという強い意見が多く寄せられた。そこで本稿では、頻出パターン解析問題に課題を限定して、これらの指摘について検証を行なう。

1. はじめに

大量のデータを横断的かつ網羅的に解析することで、新しい知見やニーズを発見しようとするビッグデータ解析が注目を集めている。こうした需要や期待の高まりに応じて、ビッグデータ解析処理を高速化かつスケールアウトする新たな計算機環境への要望も生じているが、ビッグデータ解析処理の挙動については、未だ明確になっていない。

ビッグデータ解析処理はその特徴によっていくつかの種類に分類することができるが、本稿では、そのうちのストリーム解析処理に注目する。ストリーム解析処理とは、時系列に沿って次々と到着するデータ列をリアルタイム（あるいは、ほぼリアルタイム）に解析する処理を指し、ストリーム解析処理に特化したアルゴリズムはデータマイニング分野で集中的に研究されている。HPC の分野においても、膨大なデータを解析するアプリケーションは長年研究されてきたが、ストリーム解析処理と従来の HPC 分野の大規模データ解析処理とでは、データのアクセスパターンが全く異なるため、高速化の手法が根本的に異なるべきであることが指摘されている [1]。現在の計算機環境は、Linpack [2] や SPEC [3] といった、主に CPU 性能に強スケールするベンチマークを早くすることを大きな目標のひとつとして発展してきた。しかし、ストリーム解析処理のように、データの再利用がほとんどなく、その性能が CPU 性能に強スケールしづらいアプリケーションにとって、現在の計算機環境はあまり好都合ではない。

そこで、筆者はストリーム解析処理をモデル化するためのツール群と、これらのツールによってストリーム解析処理に特化したタスクグラフを生成するための StratStream (Standard Task gRAph Toolkit for STREAM mining applications) の研究開発を行なっている。StratStream の研究開発において、データマイニング手法や機械学習手法の多くは、入力データの特徴や偏りによって実行時間に大きな分散が生じることや、同じデータマイニング手法や機械学習手法でも、入力データによって出力結果が異なるという指摘を多く受け、StratStream のようなツールキットには、代表的な入力データや入力データモデルを含めるべきだという強い意見が多く寄せられた。

こうした背景を踏まえ、本稿では、特に頻出パターン解析処理に問題を絞り、その代表的なアルゴリズムについて、いくつかの特徴的なデータを入力し、プログラムの挙動にどのような変化が生じるのかを観察し、こうした挙動の変化と入力データの特徴の相関について考察する。本稿の構成は以下の通りである。第 2 章で、頻出パターン問題について簡単に述べ、本稿で取り上げる代表的アルゴリズムについて、それぞれの特徴的な部分を中心に概要を述べる。第 3 章では、本稿での実験の概要と、使用したデータの特徴について述べた後、実験結果について考察する。第 4 章

[1] 明治大学
Meiji University
[a)] akioka@meiji.ac.jp

(1) $L_1 = large1 - itemsets$
for $k \geq 2$ and $L_{k-1} \neq \emptyset$ **do**
  (2) $C_k = apriorigen(L_{k-1})$
  for all transactions $t \in D$ **do**
    (3) $C_i = subset(C_k, t)$
    for all candidates $c \in C_t$ **do**
      (4) c.count++
    end for
  end for
  (5) $L_k = \{c \in C_k | c.count \geq minsup\}$
end for
(6) Answer $= \cup_k L_k$

図 1 Apriori アルゴリズムの概要

で関連研究を紹介し、第 5 章でまとめる。

2. 頻出パターンマイニング

頻出パターンマイニングとは、データ列の中から頻出のパターン列を探し出して列挙する問題である。代表的な例では、スーパーなどの店舗での売り上げ記録から、頻繁に購入される商品の組み合わせを探し出して列挙する、という問題などがある。この場合、各顧客の購入商品を列挙したデータひとつひとつがトランザクションとなり、全トランザクションを集計した際に現れる商品の種類の総数がアイテム種数となる。また、「頻出な」パターンの定義については、サポート値により設定する。たとえば、「サポート値が 10%」と言った場合には、全体のトランザクションの 10% 以上にあるパターンが現れた場合に、このパターンを頻出パターンとみなす。

頻出パターンマイニングの手法は色々と提案されているが、本稿では、代表的なアルゴリズムである Apriori アルゴリズム [4] および FP-growth アルゴリズム [5] を取り上げる。以下で、それぞれのアルゴリズムの概要について述べる。

2.1 Apriori アルゴリズム

Apriori アルゴリズムの概要を図 1 に示す。以下で、$k - itemset$ とは、k 個のアイテムからなる集合を指し、L_k とは長さ $k - itemset$ の集合を指す。ここで L_k は、$k - itemset$ と対応するサポート値をメンバとする。また、C_k は、頻出パターンの候補となる $k - itemset$ であり、D はトランザクションのデータベースである。

図 1 に示すアルゴリズムでは、(2) の部分で新しい頻出パターン候補を抽出している。また、この (2) で呼び出している apriorigen とは、引数 L_{k-1} をとり、全ての $k - itemsets$ の上位集合を返す関数である。Apriori アルゴリズムの特徴は、この apriorigen 関数が、サポート値を使って頻出パターンの候補を早期に枝狩りすることにある。Apriori アルゴリズムでは、あるパターンのサポート値は、必ずその部分パターンのサポート値以下になることを利用して、問題探索空間を狭くしていく。たとえば、最終的に得られる頻出パターンの最小サポートを 10%とする。ある $k - itemset$ がサポート値 10%を満たさない場合、この $k - itemset$ を元にしてできた上位の $k+1 - itemset$ がサポート値 10%を満たすことはあり得ないことから、頻出パターンの候補から外すことができるのである。

表 1 サンプル入力データ

TID	アイテム列	含まれる頻出アイテム（参考）
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

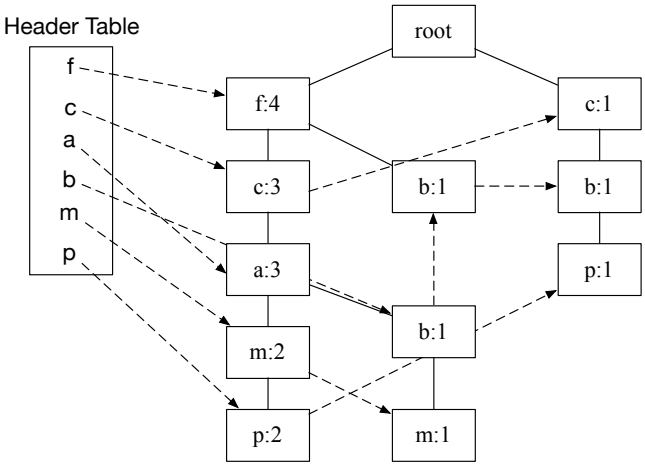

図 2 FP-tree の例

2.2 FP-growth アルゴリズム

Apriori アルゴリズムは、頻出パターンが膨大な場合、個々の頻出パターンの長さが長い場合、最小サポート値が極めて小さく設定される場合に不利であるとされている。Apriori アルゴリズムに対して、FP-growth アルゴリズムとは、頻出パターン候補を作ることなく、深さ優先探索で頻出パターンを探索する手法である。ただし、FP-tree と呼ばれる木構造で入力データを管理するため、Apriori アルゴリズムと比較してメモリ使用量は多いとされ、FP-growth アルゴリズムのメモリ使用量は頻出パターンの数に比例することが知られている。

最小サポート値（ここでは率ではなく出現回数でサポート値を決定する）を 3 とし、入力データとして表 1 のようなトランザクションを例として、図 2 に示す FP-tree の生成手法を説明する。

(1) 入力データを走査し、個々のアイテムの出現回数を数え、出現回数順にアイテムを並べ替えたデータ列 (f:4), (c:4), (a:3), (b:3), (m:3), (p:3) を得る。ここに現れない入力データは、サポート値を出現回数がサポート値を下回るために割愛される。
(2) 再度入力データを走査し、トランザクション 100 から

図 2 の最左の枝である (f:1),(c:1),(a:1),(m:1),(p:1) を得る。

(3) 引き続き、トランザクション 200 から次の枝を生成するが、先頭の f, c, a は先に作った枝と共通であるため、先の枝の a 以降を分岐させて節 (b:1) を作り、さらに (m:1) をぶら下げる。結果として、先の最左の枝は (f:2),(c:2),(a:2), (m:1),(p:1) となる。

(4) 同様にトランザクション 300 以降についても入力データから枝を作っていく。

以上の手順により生成した FP-tree を root ノードから順に辿ることで、全ての頻出パターンを抽出することができる。

3. 実験

本稿では、入力データの特性によるアルゴリズム実装の挙動の変化を比較・検討することが目的である。そこで、Apriori アルゴリズムおよび FP-growth アルゴリズムの実装に対して、以下の要領で実験を行ない、挙動の変化を観察する。

3.1 実験の概要

Apriori アルゴリズムおよび FP-growth アルゴリズムの実装には、Borgelt が公開している C 言語による実装を使用した [6], [7]。これらは一切並列化を施していない実装である。また、入力データのフィルタリングやソートを行なってから処理を施しているため、正確にはストリーム解析処理ではないが、今回は実験の公平性を重視して公開されているコードをそのまま使用した。これらの実装に、以下のデータを入力して挙動の変化を見る。なお、それぞれの入力データの特徴を表 2 にまとめる。

census 本実験で使用する Apriori アルゴリズムおよび FP-growth アルゴリズムの実装と共に、Borgelt のサイトで配布しているテストデータ [6], [7] で、国勢調査の結果である。データの例を図 3 に示す。トランザクション数に対する出現アイテム種数の割合が 0.28% であり、他のデータと比べてその割合は中程度である。またトランザクション数は今回の実験で用いるデータの中では非常に少ない。

papertitle KDD Cup 2013 Author-Paper Identification Challenge (Track 1) で使用したデータで、Kaggle のコンテストサイト [8] から入手可能である。データの例を図 4 に示す。今回は、このうちの Paper.csv から論文タイトルのみを抽出して使用した。トランザクション数に対する出現アイテム種数の割合が 44.0% であり、今回用いるデータの中では、その割合が非常に高い。トランザクション数は中程度である。

shoppers Acquire Valued Shoppers Challenge で使用したデータで、papertitle 同様、Kaggle のコンテストサイト [9] から入手可能である。今回は、このうちの transactions.csv に含まれる購入商品のカテゴリデータのみを、顧客 ID および購入日付ごとに抽出し、トランザクションデータとして用いた。データの例を図 5 に示す。トランザクション数に対する出現アイテム種数の割合が 0.0032% であり、他のデータと比べてその割合は非常に小さい。トランザクション数は今回の実験で用いるデータの中では最多である。

表 2 入力データの特徴

	トランザクション数	出現アイテム種数
census	48,842	135
papertitle	2,104,240	925,151
shoppers	26,496,646	836

```
1 age=middle-aged workclass=State-gov education=Bachelors edu_num=13
  marital=Never-married occupation=Adm-clerical relationship=Not-in-family
  race=White sex=Male gain=medium loss=none hours=full-time country=United-
  States salary<=50K
2 age=senior workclass=Self-emp-not-inc education=Bachelors edu_num=13
  marital=Married-civ-spouse occupation=Exec-managerial
  relationship=Husband race=White sex=Male gain=none loss=none hours=half-
  time country=United-States salary<=50K
3 age=middle-aged workclass=Private education=HS-grad edu_num=9
  marital=Divorced occupation=Handlers-cleaners relationship=Not-in-family
  race=White sex=Male gain=none loss=none hours=full-time country=United-
  States salary<=50K
```

図 3 census のデータ例

```
1 Stitching videos streamed by mobile phones in real-time
2 A nonlocal convection–diffusion equation
3 Area Effects in Cepaea
4 Multiple paternity in a natural population of a salamander with long-
  term sperm storage
```

図 4 papertitle のデータ例

```
1 707 6319 9753 2509 5555 9753 9909 5907 921 7344 4107 2106 814 9122
  4120 6315 907 9753 4509 2630 815 8101 5615 5824 907 9753 836 1908 904
  6401 3204 5620 3009 9753 3009 2301 3202 5620 2928 1905 3101 5823 3309
  1905 2908 3630 3626 3612 3319 3630 3410 3611
2 809 7113 6010 8101 2702 8101 6408
3 3601 5620 4106 6316 3307 2301 4402 2633 902 5613 811 908
```

図 5 shoppers のデータ例

表 3 census データ使用時の発見セット数

サポート値	発見セット数
1.25	134,780
2.5	49,648
5.0	15,928
10.0	4,415
20.0	904
40.0	117

3.2 結果と考察

census について、サポート値を 1.25 から 40 まで変化させながら、それぞれのアルゴリズムで実行した場合の発見セット数を表 3 に示す。また、それぞれのアルゴリズムでの実行時間の比較を図 6 に、Apriori アルゴリズムの実行時間の内訳を図 7 に、FP-growth アルゴリズムの実行時間の内訳を図 8 にそれぞれ示す。

サポート値が小さくなるにつれて、頻出パターン候補が膨大となり、Apriori アルゴリズムが不利となるのは、アルゴリズムの特性からも予想することができる。FP-growth

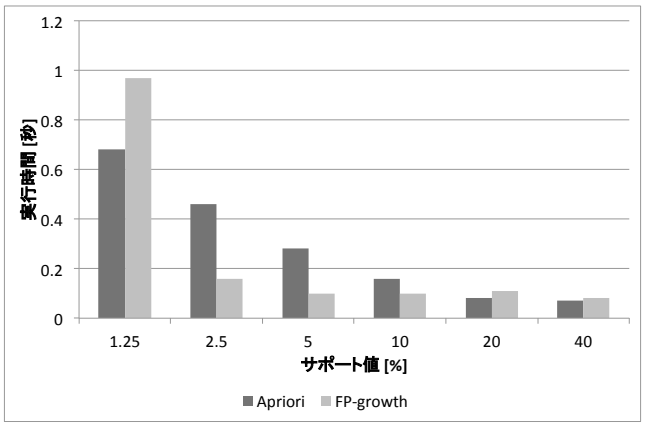

図 6 Apriori アルゴリズムと FP-growth アルゴリズムの実行結果比較（census）

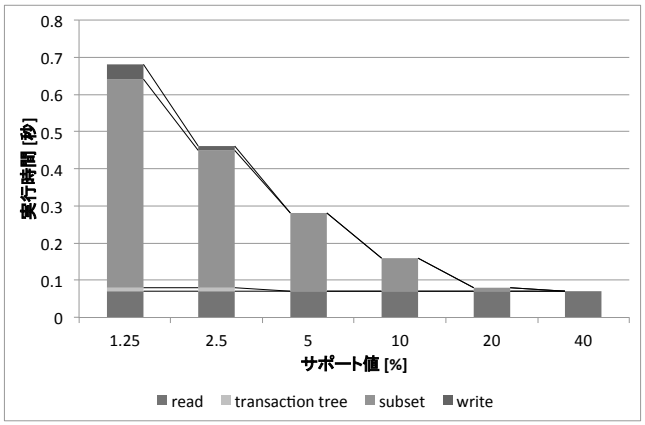

図 7 Apriori アルゴリズム実行時間の内訳（census）

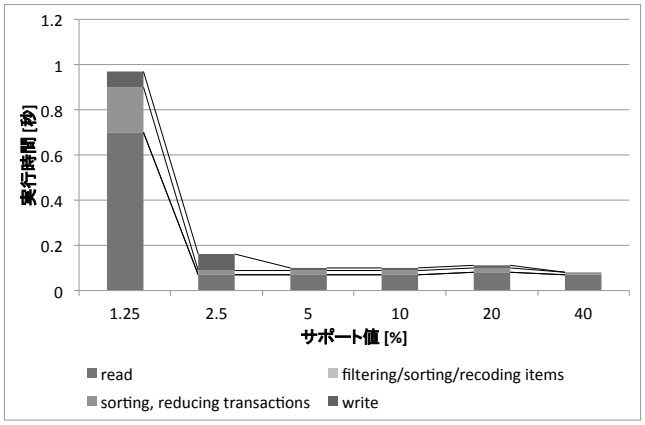

図 8 FP-growth アルゴリズムの実行時間の内訳（census）

アルゴリズムの FP-tree 生成コストを、Apriori アルゴリズムの頻出パターン探索コストが上回ったのが、サポート値 1.25%の場合であり、サポート値が 2.5%以上の場合には Apriori アルゴリズムの方が有利であったと考えられる。しかし、実行時間内訳を見ると、FP-growth アルゴリズムはサポート値 2.5%以上では入出力部分以外の実行時間がほぼ一定であるのに対し、Apriori アルゴリズムではサポート値にほぼ反比例していることが分かる。

次に、papertitle について、サポート値を 1.25 から 40 まで変化させながら、それぞれのアルゴリズムで実行した場合の発見セット数を表 4 に示す。また、それぞれのアルゴリズムでの実行時間の比較を図 9 に、Apriori アルゴリズムの実行時間の内訳を図 10 に、FP-growth アルゴリズムの実行時間の内訳を図 11 にそれぞれ示す。

表 4 papertitle データ使用時の発見セット数

サポート値	発見セット数
1.25	49
2.5	21
5.0	8
10.0	3
20.0	0
40.0	0

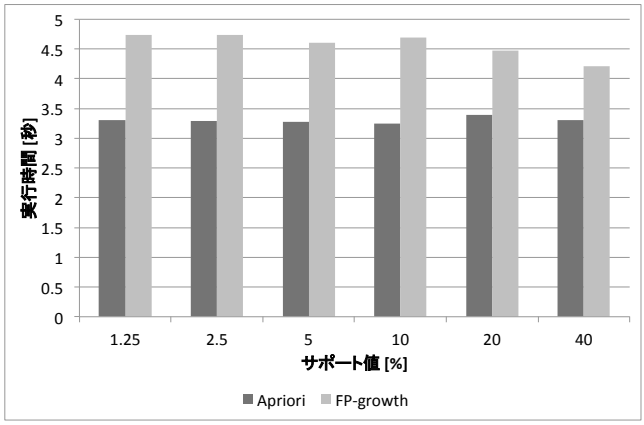

図 9 Apriori アルゴリズムと FP-growth アルゴリズムの実行結果比較（papertitle）

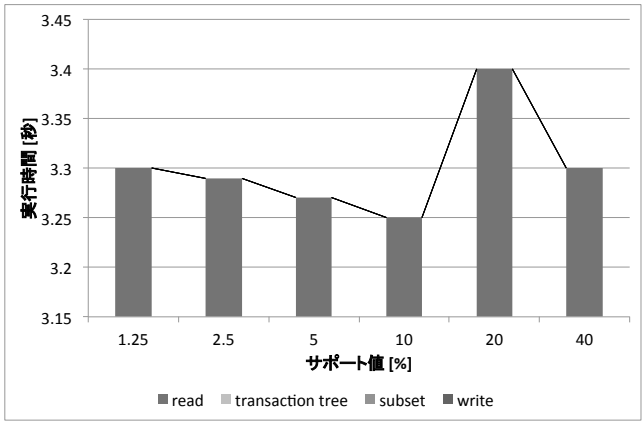

図 10 Apriori アルゴリズム実行時間の内訳（papertitle）

papertitle については、発見パターンが census よりも非常に少ない。その結果、アルゴリズム特性から予想可能な通り、全般において Apriori アルゴリズムが FP-growth アルゴリズムよりも優勢である。また、サポート値の変化に対して発見パターン数の変化が大きくないことから、実行時間についても、双方のアルゴリズムでほぼ横ばいである。

最後に、shoppers について、サポート値を 1.25 から 20 まで変化させながら、それぞれのアルゴリズムで実行した場合の発見セット数を表 5 に示す。また、それぞれのアル

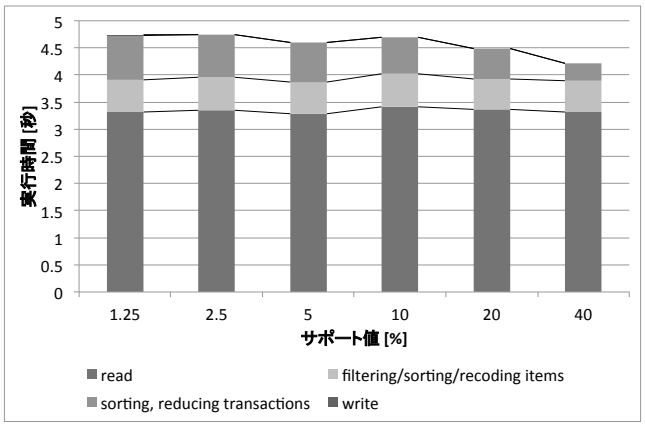

図 11 FP-growth アルゴリズムの実行時間の内訳（papertitle）

表 5 shoppers データ使用時の発見セット数

サポート値	発見セット数
1.25	991
2.5	161
5.0	26
10.0	1
20.0	0

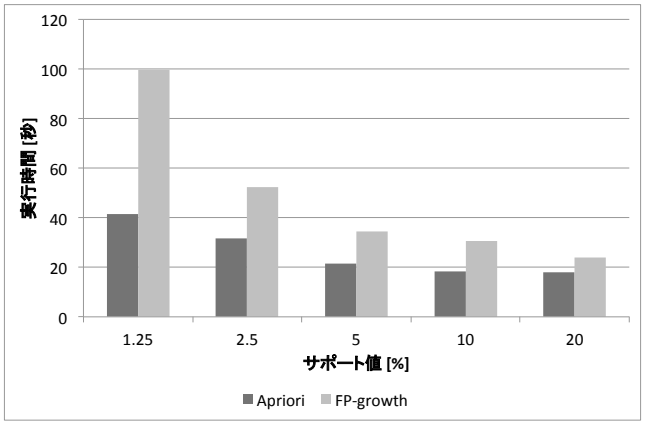

図 12 Apriori アルゴリズムと FP-growth アルゴリズムの実行結果比較（shoppers）

ゴリズムでの実行時間の比較を図 12 に、Apriori アルゴリズムの実行時間の内訳を図 13 に、FP-growth アルゴリズムの実行時間の内訳を図 14 にそれぞれ示す。なお、shoppers については、サポート値 40%の場合について、Apriori アルゴリズム、FP-growth アルゴリズム共に、本処理に入る前に候補アイテムがゼロとなってしまったため、有効な実行結果を得ることができなかった。

shoppers についても、全般に Apriori アルゴリズムの方が FP-growth アルゴリズムよりも優勢である。しかし、実行結果の内訳を見ると、Apriori アルゴリズムは入出力部分以外の実行時間がサポート値にゆるやかに反比例している傾向が見える。一方で、FP-growth アルゴリズムでは、サポート値 10%以下では、ほぼ横ばいの傾向にあり、このままサポート値を小さくすることで FP-growth アルゴリズムが優勢になるケースが生じる可能性がある。しか

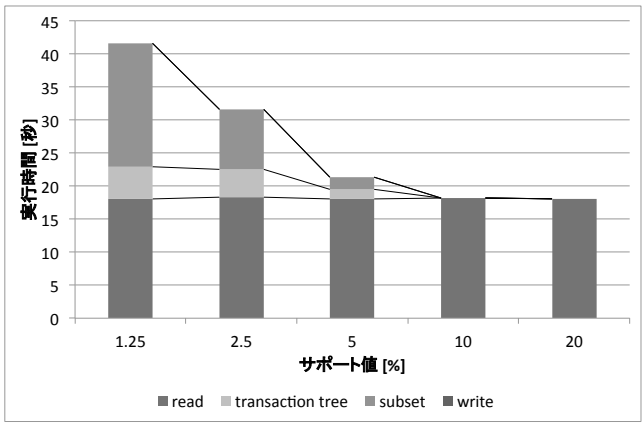

図 13 Apriori アルゴリズム実行時間の内訳（shoppers）

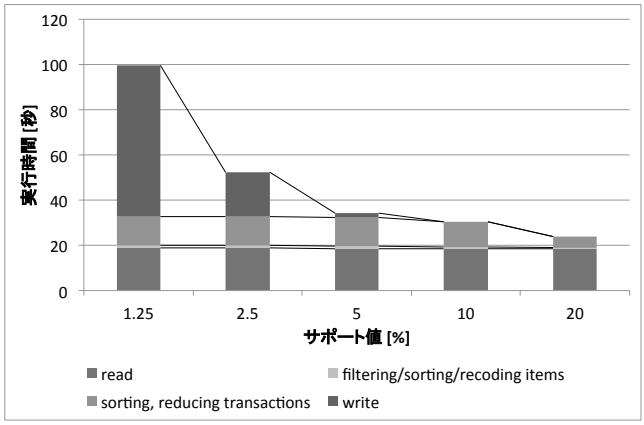

図 14 FP-growth アルゴリズムの実行時間の内訳（shoppers）

し、census における FP-growth アルゴリズムのサポート値 1.25%と 2.5%の場合のように、FP-growth アルゴリズムにおいても入出力部分以外の実行時間が急激に増加する可能性もある。

まとめると、入出力部分以外の実行時間について、Apriori アルゴリズムはサポート値に反比例する傾向が見られることが多く、FP-growth アルゴリズムはほぼ横ばい、あるいは段階的に増加する傾向が見られることが分かった。しかし、サポート値と実行時間変化の関係性は直接的ではなく、入力データから取り出される頻出パターンの候補の数や、サポート値を超える出現回数を持つアイテム種数などをパラメータとすることが予想できる。

4. 関連研究

プログラム内部のデータ依存関係や制御依存関係、各部分の実行コストなどを模式的に有効グラフで表現し、プログラムをモデル化するタスクグラフに関する研究や、ランダムなタスクグラフの生成手法に関する研究は数多くある。しかし、実際のプログラムからタスクグラフを構築する例は非常に少なく、それらのタスクグラフも高速フーリエ変換などの、HPC コミュニティで長年研究されてきた数値計算アプリケーションを対象にした研究ばかりであり、ス

トリーム解析処理のようなアプリケーションを対象にした研究は少ない。

Task Graphs for Free (TGFF) は、ランダムタスクグラフジェネレータであり [10], [11]、ユーザの好みに応じたパラメータ設定でランダムなタスクグラフを生成することができる。GGen もまた、Corderiro らが提案するランダムタスクグラフジェネレータであり [12]、代表的なランダムタスクグラフ生成アルゴリズムに基づいて、タスクグラフを生成する。GGen は、生成したランダムタスクグラフを最長パスやエッジの数などのパラメータで分析するためのツールも備えている。Task graph generator は、ランダムタスクグラフジェネレータであるが、同プロジェクトでは高速フーリエ変換、ガウスの消去法、LU 分解など、数値計算の分野で頻繁に使われる手法について、実際のプログラム実装から生成したタスクグラフも公開している [13]。また、TGFF よりもタスク形状に関する制約が少なく、スター型やリング型のタスクグラフが生成できることも特徴である。Tobita らは、Standard Task Graph Set (STG) を公開している [14], [15]。さらには STG を用いて代表的なスケジューリングアルゴリズムの評価を行ない、最適解の公開も行なっている。公開されているタスクグラフの多くはランダムタスクグラフであるが、中にはロボット制御プログラムや疎行列解法、SPEC fpppp など、実際のプログラム実装から構築したタスクグラフも含んでいる。

以上のように、ランダムタスクグラフを中心として、一部の数値計算アプリケーションを主な対象として、タスクグラフ研究は多く行なわれている。しかし、入力データの特性とそれと連動したアプリケーション挙動の変化をモデル化した研究はない。また、ランダムタスクグラフの利用には注意が必要であることが Corderiro によって指摘されており [12]、実情に即したタスクグラフの重要性は認識されている。

5. おわりに

本稿では、ストリーム解析アプリケーションの挙動をモデル化する上で不可欠な、入力データとアプリケーション挙動の関係性を明確にするために、頻出パターンマイニングの代表的手法である Apriori アルゴリズム、および FP-growth アルゴリズムを取り上げ、異なる特性の入力データを用いて実行し、その挙動の変化について考察した。今後は、さらに多くのストリーム解析手法や特徴的な入力データを用いて同様の実験を行ない、入力データのモデル化を行なっていく予定である。

参考文献

[1] Raicu, I., Foster, I., Zhao, Y., Little, P., Moretti, C. M., Chaudhary, A. and Thain, D.: The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems, *Proc. the 18th ACM International Symposium on High Performance Distributed Computing (HPDC'09)*, Munich, Germany, pp. 207–216 (2009).
[2] Dongarra, J., Bunch, J., Moler, C. and Stewart, G. W.: *LINPACK Users Guide*, SIAM (1979).
[3] SPEC: Standard Performance Evaluation Corporation.
[4] Agrawal, R. and Srikant, R.: Fast algorithms for mining association rules in large databases, *Proc. the 20th International Conference on Very Large Data Bases (VLDB1994)*, Santiago de Chile, Chile, pp. 487–499 (1994).
[5] Han, J., Pei, H. and Yin, Y.: Mining frequent patterns without candidate generation, *Proc. the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD2000)*, Dallas, USA, pp. 1–12 (2000).
[6] Borgelt, C.: Apriori - Association Rule Induction / Frequent Item Set Mining, http://www.borgelt.net/apriori.html.
[7] Borgelt, C.: FP-growth - Frequent Item Set Mining, http://www.borgelt.net/fpgrowth.html.
[8] kaggle: KDD Cup 2013 - Author-Paper Identification Challenge (Track 1), https://www.kaggle.com/c/kdd-cup-2013-author-paper-identification-challenge.
[9] kaggle: Acquire Valued Shoppers Challenge, https://www.kaggle.com/c/acquire-valued-shoppers-challenge.
[10] Dick, R. P., Rhodes, D. L. and Wolf, W.: Task graphs for free, *Proc. International Workshop on Hardware/Software Codesign*, Seattle, USA, pp. 97–101 (1997).
[11] Dick, R.: TGFF, http://ziyang.eecs.umich.edu/ dickrp/tgff.
[12] Corderiro, D., Mounie, G., Perarnau, S., Trystram, D., Vincent, J. M. and Wagner, F.: Random graph generation for scheduling simulations, *Proc. the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools'10)*, Torremolinos, Spain (2010).
[13] task graph generator project: TGG, Task graph generator, http://taskgraphgen.sourceforge.net.
[14] Kasahara Lab.: STG, Standard task graph set, http://www.kasahara.elec.waseda.ac.jp/schedule/index.html.
[15] Tobita, T. and Kasahara, H.: A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *Journal of Scheduling*, Vol. 5, pp. 379–394 (2002).