D-011

異周期カウンタを用いたスライディングウィンドウ型ストリームに対する頻度クエリの精度向上

Improving the accuracy of frequency queries for Sliding Window streams via different frequency counter

山川竜太郎 *
Ryutaro Yamakawa
y2331158@gl.cc.uec.ac.jp

古賀 久志 *
Hisashi Koga
koga@sd.is.uec.ac.jp

1 はじめに

近年，IoT の普及に伴い時間と共に新データが追加されるストリームデータが注目されている．例えばネットワークの監視ログは時間と共にデータが追加されるストリームデータである．ストリームデータは時間が経過するとデータ量が膨大になるため，過去の全データを保持することが難しい．そこで**スケッチ**と呼ばれるサイズが小さい要約を保持し，スケッチを用いて様々な問い合わせ (クエリ) に対する近似解を返すことがよくなされる．Count-Min Sketch (以下，CMS) [1] はこうしたスケッチの代表例であり，指定された要素の近似頻度を返すための確率的データ構造である．CMS の一番の特色は，確率的なヒストグラムを複数個作って，頻度推定の精度低下を抑制したことである．しかしながら，CMS は新データが単に増え続ける状況を想定し，古いデータを廃棄するスライディングウィンドウ (以下ウィンドウモデル) に対応しない．そこで，CMS をウィンドウモデルに適応させた Sliding Sketch [2] が提案された．ウィンドウモデルに対応したスケッチは，他に [3] [4] が存在する．

本稿では，Sliding Sketch が確率的なヒストグラムを複数持つにもかかわらず，クエリ応答時には少数のヒストグラムしか使用せず，CMS の長所を維持していないことを指摘する．提案手法は，クエリ応答時に使われる確率的ヒストグラムの数を増やして近似頻度の精度を向上させる．

2 既存技術

毎時刻データが 1 つ到着するストリームデータを考える．時刻 t に到着するデータを e_t と表記する．e_t はアルファベット，すなわちカテゴリを表すラベルである．データのカテゴリ数を $|\Phi|$ とする．

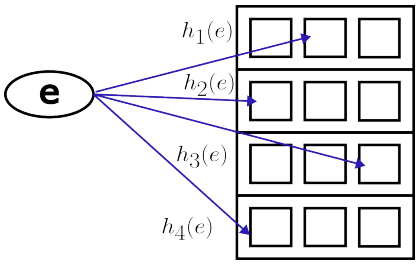

図 1: CMS の更新

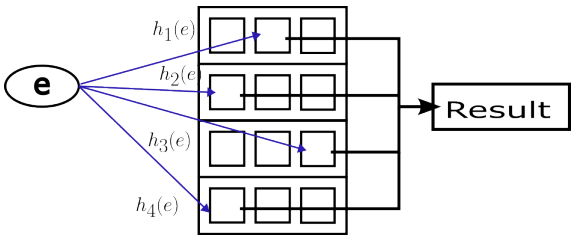

図 2: CMS のクエリ

2.1 Count Min Sketch

Count-Min Sketch (CMS) は確率的なデータ構造であり，サイズ n の配列を d 個持つ．これを $A_1, A_2, \cdots, A_d$ とする．また，互いに独立な d 個のハッシュ関数を $h_1, h_2, \cdots, h_d$ も持ち，ハッシュ関数 h_i は A_i に関連付けられている．ハッシュ関数の値域は 0 から $n-1$ までの整数であるが，n は $|\Phi|$ より十分小さい．個々の配列 A_i は近似ヒストグラムを記録するのに使用され，A_i の全要素は 0 に初期化される．

図 1 のように時刻 t に e_t が到着した時には，e_t に関する d 個のハッシュ値 $h_1(e_t), h_2(e_t), \cdots, h_d(e_t)$ を計算し，$1 \leq i \leq d$ に対して $A_i[h_i(e_t)]$ を 1 増やし，近似ヒストグラム A_i を更新する．$n \ll |\Phi|$ よりハッシュ値が衝突するため，A_i の 1 要素が複数アルファベットの頻度合計を記憶することになり，正確なヒストグラムにはならない．図 2 のように指定された要素 e の頻度を答えるクエリ (頻度クエリ) に対して，

*電気通信大学　大学院　情報・ネットワーク工学専攻

CMS は

$$\min_{1\le i\le d}(A_i[h_i(e)]). \quad (1)$$

を返す．ハッシュの衝突により $A_i[h_i(e)]$ は必ず e の真の頻度以上になるが，CMS は式 (1) により，d 個の中で最もハッシュ衝突の影響を受けていない要素値を返す．

CMS の空間計算量は要素数 n の配列が d 個あるので $O(nd)$ となる．n を大きくして空間計算量を大きくすると，n が $|\Phi|$ に近づくのでハッシュ衝突回数が減るので，近似頻度の精度が向上する．また，ハッシュ関数の数 d を増やして空間計算量を増やすと，ハッシュ衝突の影響を受けていないカウンタが出現する確率が高まるので，やはり近似精度が向上する．

2.2 Sliding Sketch

CMS は新データの到着により，データ量が経過時間に関して単調増加し続けるモデルを想定している．つまり，配列 A_i $(1 \le i \le d)$ の要素は単調増加する．このやり方は，ストリームデータの主要モデルであるスライディングウィンドウ（以下ウィンドウ）に対応しない．一般にストリームデータでは直近に到着した新しいデータほど価値が高い．ウィンドウは直近のデータを重視したストリームデータの表現であり，直近 W 個のデータのみを保持する．つまり，古いデータは観測対象から排除される．W をウィンドウサイズと呼ぶ．とくに時刻 t のウィンドウ SW_t は $\{e_{t-W+1}, e_{t-W+2}, \cdots, e_t\}$ という到着時間順に並んだ W 個の要素を持つ．

Sliding Sketch [2] は CMS をウィンドウモデルに拡張した手法である．CMS と同様に d 個の要素数 n の配列 A_i $(1 \le i \le d)$ を持つ．ただし，配列の 1 要素 $A_i[j]$ は B^{new} と B^{old} という 2 つのカウンタで構成される．B^{new} は W 時間の一定周期で 0 にリセットされるカウンタであり，B^{new} がリセットされる直前に，B^{new} の値は B^{old} に退避される．この結果，$A_i[j]$ は $(1+\alpha)$ サイクル分の頻度を管理し，古い 1 サイクル分が B^{old} に，新しい α サイクル分が B^{new} に保存される．ここで α は $0 \le \alpha \le 1$ を満たす変数である．

B^{new} がリセットされるタイミングは配列要素ごとに異なり，のべ nd 個の要素を周期 W のタイマで順番にスキャンしながら 0 にリセットする．例えば，時刻 T にある配列要素の B^{new} がリセットされると，次の配列要素の B^{new} は時刻 $T+\frac{W}{nd}$ にリセットされる．この性質から，配列 A_i の要素 $A_i[j]$ が $(1+\alpha)$ サイクル分の頻度を持つとき，配列 A_{i-1} の要素 $A_{i-1}[j]$ は $1+\alpha+\frac{1}{nd}\times n = (1+\alpha+\frac{1}{d})$ サイクル分の頻度を持つことになる．ただし，$\alpha+\frac{1}{d}>1$ の場合は，B^{new} から B^{old} への退避が発生するので，$\alpha+\frac{1}{d}$ サイクル分の頻度を持つ．図 3 は $d=10$ の例である．A_2 の要素は 1.4 サイクル以上 1.5 サイクル未満の頻度を記憶し，A_4 の要素は 1.5 サイクル以上 1.6 サイクル未満の頻度を記憶する．このように $d=10$ 個の配列が $\frac{1}{d}=\frac{1}{10}$ サイクル分だけ異なる頻度を持つ．

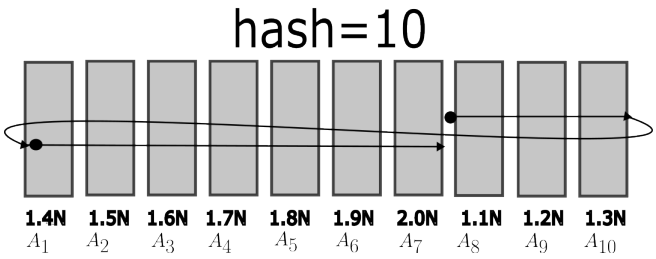

図 3: $d=10$ のときの配列が保持する頻度

時刻 t に e_t が到着した時には，e_t に関する d 個のハッシュ値 $h_1(e_t), h_2(e_t), \cdots, h_d(e_t)$ を計算し，$1 \le i \le d$ に対して $A_i[h_i(e_t)]$ の B^{new} カウンタを 1 増やす．指定された要素 e の頻度クエリに対しては CMS と同様に $\min_{1\le i\le d}(A_i[h_i(e)])$. を近似頻度として返す．ただし，$A_i[h_i(e)]$ は B^{new} カウンタと B^{old} カウンタとの合計値である．Sliding Sketch の頻度クエリ処理に対する戦略は以下のようにまとめられる．

1. $\forall 1 \le i \le d$ に対して $A_i[h_i(e)]$ が少なくとも 1 サイクル以上の頻度を持つので，頻度の under estimation が発生しないことを保証できる．
2. $1+\frac{1}{d}$ サイクル以下の頻度しか持たない $A_i[h_i(e)]$ が必ず 1 つ存在することを保証し，近似頻度が過大になることを抑制する．

3 異周期カウンタを利用した SS

ここでは我々の提案手法を説明する．まず，3.1 節で提案手法を着想するきっかけになった 従来手法 Sliding Sketch の欠点を指摘する．3.2 節で欠点を改良した提案手法を詳述する．

3.1 Sliding Sketch の欠点

本節では，Sliding Sketch のことを SS と省略して表記する．SS ではカウンタ B^{new} を 0 にリセットするタイミングを d 個の配列 $A_1, A_2, \cdots, A_d$ に対して $\frac{1}{d}$ ずつ時間差を付けることにより，常時どれか 1 つの配列は $1+\frac{1}{d}$ サイクル以下の頻度しか保持しないことを保証し，頻度クエリに対して過大な頻度を返さないように設計されている．言い換えると開始時刻が異なる複数期間に対する頻度を保持するために，複数個のハッシュ関数を用意している．

しかし SS は，ハッシュ衝突の影響が少ないハッシュ関数を選ぶために複数のハッシュ関数を用意するという CMS の元々のアイデアからはかけ離れてしまっている．具体的には，頻度クエリに対して $\min_{1\le i\le d}(A_i[h_i(e)])$ を返す際に，返り値に対応する

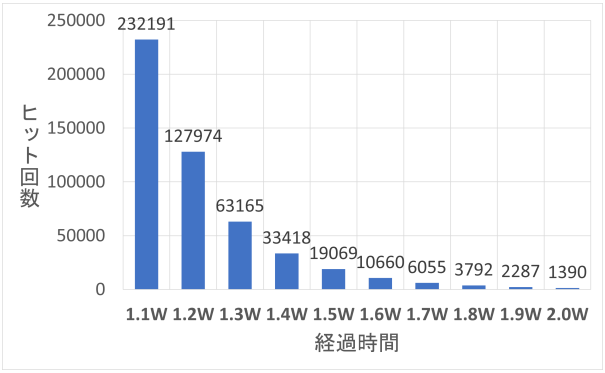

図 4: 頻度クエリを発行したとき，最小値を取得した配列要素の頻度保持期間

ハッシュ関数 (h_j とする) はハッシュ衝突の少なさよりも配列 A_j の頻度保持期間の短さを基準として選ばれる．図 4 は $d = 10$ で 50 万件の頻度クエリを発行したときに，$\mathrm{argmin}_{1\leq i\leq d}(A_i[h_i(e)])$ をとなる頻度保持期間を表したものである．半数の頻度クエリで頻度保持期間が 1.1 サイクル未満の配列が最小値として選ばれていることがわかる．頻度保持期間が 0.1 サイクル増える度に最小値になるケースは半減する．

本研究では SS を改良し，複数のハッシュ関数に対して頻度保持期間が同時にほぼ 1 サイクルとなる状況を作り，ハッシュ衝突の影響が少ないハッシュ関数を選択することを目指す．

3.2 提案手法 DCSS

複数個のハッシュ関数で頻度保持期間が同時に 1 サイクルになるようにするため，提案手法では B^{new} がリセットされる周期を配列すなわちハッシュ関数によって変える．具体的には，半数の配列をこれまでとおり周期 W で B^{new} を 0 にリセットし，残り半数を周期 $0.9W$ で B^{new} を 0 にリセットする．つまり，

- $A_1, A_2, \cdots, A_{\frac{d}{2}}$ は周期 W で B^{new} を 0 にリセットする．こちらの配列群をグループ G_1 とする．
- $A_{\frac{d+1}{2}+1}, A_{\frac{d+2}{2}}, \cdots, A_d$ は周期 $0.9W$ で B^{new} を 0 にリセットする．これをグループ G_2 とする．

上記のように 2 種類の周期を持つことから，提案手法を Dual Cycle Sliding Sketch (DCSS) と名付ける．

B^{new} がリセットされる直前に，B^{new} の値は B^{old} に退避される点は全配列で共通である．この結果，

- グループ G_1 に属する配列の要素は 1〜2 サイクル分の頻度を保持する．
- グループ G_2 に属する配列の要素は 0.9〜1.8 サイクル分の頻度を保持する．

さらに B^{new} がリセットされるタイミングは配列要素ごとに異なる．グループ G_1 とグループ G_2 のリセットタイミングは独立である．グループ G_1 では，周期 W で $\frac{nd}{2}$ 個の要素をリセットすることになるので，連続した要素の時間差は $\frac{2W}{nd}$ となる．グループ G_2 では，周期 $0.9W$ で $\frac{nd}{2}$ 個の要素をリセットすることになるので，連続した要素の時間差は $\frac{1.8W}{nd}$ となる．

時刻 t に e_t が到着した時の配列の更新処理は SS とまったく同じで，e_t に関する d 個のハッシュ値 $h_1(e_t), h_2(e_t), \cdots, h_d(e_t)$ を計算し，$1 \leq i \leq d$ に対して $A_i[h_i(e_t)]$ の B^{new} カウンタを 1 増やす．指定された要素 e の頻度クエリに対して，$\min_{1\leq i\leq d}(A_i[h_i(e)])$ を近似頻度として返す点も SS から不変である．

DCSS ではグループ G_2 に属する配列の要素は 0.9〜1.8 サイクル分の頻度情報を持つ．このことは SS よりも 1 サイクルに近い頻度情報を持つ確率が高いことを示唆する．一方，1 サイクル未満の頻度情報を持つことがあるため，頻度の under estimation が発生する可能性がある．しかし，under estimation が 0.1 サイクル分以上になることがない．スケッチに割り当てられるメモリサイズが少ない時にはハッシュ衝突による over estimation が見込まれるので，実際に under estimation に陥る可能性は低い．さらに B^{new} がリセットされる周期をハッシュ関数によって変え，2 個のハッシュ関数が頻度保持期間が同時にほぼ 1 サイクルとなる状況を作り，ハッシュ衝突の影響が少ないハッシュ関数を選ぶことを可能にした．

4 実験評価

本節では実験により提案手法 DCSS と従来手法 SS との性能比較実験を行う．先行研究の SS のソースコード [5] に同梱されているデータセットを用いる．データ数が 50 万のストリームデータであり，入力要素 e_t は 8byte のバイナリデータである．DCSS は SS のソースコード [5] を改良して実装した．すべてのプログラムを C++で実装した．

4.1 実験内容

50 万個のデータを含むデータストリームを DCSS と SS に与えて，それぞれのスケッチをデータ到着の度にオンラインで更新する．そして，ハッシュ関数つまり配列の数 $d = 10$ を固定し，スケッチに割り当てられるメモリサイズ $M = \{0.5, 1, 2, 4, 8, 16\}$MB の範囲で変更した．$d$ が固定なので，M を増やすと 1 配列あたりの要素数 n が増加する．1 サイクルの長さ $W = 50000$ とした．頻度クエリは毎時刻発行した．つまり，頻度クエリの総数は 500000 個である．

頻度クエリの精度は Average Relative Error (以下，ARE) で評価する．ARE は，複数の頻度クエリに対する正解頻度とスケッチを用いて返される近似頻度との相対誤差の平均値を取ったものであり，式 2 で表すことができる．e_i は i 回目の頻度クエリの対象となる要素を表す．f_i は要素 e_i の正解頻度であり，

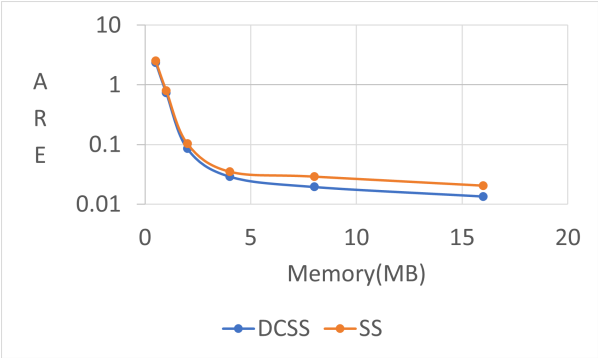

図 5: ARE

M	0.5	1	2	4	8
差	0.1623	0.066	0.0179	0.0061	0.0096

表 1: メモリサイズ M を変化させた時の DCSS と SS の ARE の差

$\hat{f}_i$ は，要素 e_i の近似頻度である．ψ は頻度クエリのセットであり，$|\psi|$ は頻度クエリの発行回数である．

$$\frac{1}{|\psi|}\sum_{e_i\in\psi}\frac{|f_i-\hat{f}_i|}{f_i} \tag{2}$$

4.2 実験結果と考察

スケッチのメモリサイズ M を変化させた時の ARE を図 5 に示す．このグラフでは横軸がメモリサイズとなっている．縦軸が ARE を対数目盛で示している．DCSS のどのメモリサイズでも ARE が低く，SS よりも近似精度を改善できた．

図 5 は対数グラフで表しているため，M が小さい時に DCSS と SS の差がないように見える．しかし，実際には ARE の差は M が小さいほど大きくなる．そのことを示すため，M が変化した時の DCSS の ARE が SS の ARE よりどれだけ小さくなったかを表 1 にまとめた．ARE の差はメモリサイズ M が少ないほど拡大していることが読み取れる．M が小さいと配列要素数 n が小さくなるためのハッシュ衝突が激しくなる．DCSS は M が小さい時に SS との差を広げられているが，これは周期の異なるカウンタを混在させて複数のカウンタがほぼ同時に 1 サイクル分の頻度を持てるようにしたことでハッシュ衝突の少ないハッシュ関数を選べることにしたのが原因であると考えている．つまり，SS の課題であったハッシュの衝突による精度の低下は改善できたと考えている．

一方で M が大きくハッシュ衝突が起きにくい状況でも DCSS は SS を上回る．これはリセットされる周期が $0.9W$ である配列において，配列要素が 1 サイクルに近い頻度情報を持つ確率が高いことが原因であると思われる．

M (MB)	0.5	1	2	4
回数	1	0	2328	2936

表 2: Under Estimation の回数

Under Estimation の回数提案手法では頻度クエリに対して Under Estimation が起きる可能性がある．Under Estimation の頻度をメモリサイズ M を $\{0.5, 1, 2, 4\}$MB で変化させて計測した．結果を表 2 に示す．．メモリサイズが 1MB 以下で小さい時にはハッシュ衝突によりスケッチから返される近似頻度が増加するため，under estimation は発生しない．M が 2MB 以上でハッシュ衝突が緩和する状況では under estimation が発生するが，500000 個のクエリを処理しているので，発生率は 0.6%未満である．

5 結論

本稿ではスライディングウィンドウモデルのストリームデータに対する頻度クエリを取り扱った．既存手法では複数のハッシュ関数を配列間で時間差を保持するために使用しており，ハッシュ衝突の影響が少ないハッシュ関数を選択しにくくなっていることを指摘した．この状況を改善するため，提案手法 DCSS では 0 リセットさせる周期が異なる 2 種類のカウンタを混在させ，同時に複数のハッシュ関数が 1 サイクル分の頻度を持つ状況を作り出し，ハッシュ衝突の影響が小さいハッシュ関数を選べるようにした．実験評価により，すべてのメモリサイズに対して DCSS が SS よりエラーを小さくできることを示した．今後の課題としてはより大規模な実データセットでの評価が挙げられる．

謝辞

本研究は JSPS 科研費 JP21K11901 の助成を受けたものである．

参考文献

[1] G.Cormode and S.Muthukrishnan. Approximating Data with the Count-Min Data Structure. IEEE Software (Volume: 29), Pages 64 - 69, 2011.

[2] Xiangyang Gou, Long He, Yinda Zhang, Ke Wang, Xilai Liu, Tong Yang, Yi Wang and Bin Cui. Sliding Sketches: A Framework using Time Zones for Data Stream Processing in Sliding Windows. KDD, 2020.

[3] Nicolò Rivetti, Yann Busnel and Achour Mostefaoui. Efficiently Summarizing Distributed Data Streams over Sliding Windows. IEEE 14th International Symposium on Network Computing and Applications, 2015.

[4] Zijun Hang, Yongjie Wang and Shuguang Huang. HSS: Faster and More Accurate Sliding Sketch by Using Half Fields. National University of Defense Technology. IEEE 23rd HPCC, 2021.

[5] “source code of sliding sketches and other sketches”, https://github.com/sliding-sketch/Sliding-Sketch.