

An Approximate Counting for Big Textual Data Streams

Rudy Raymond¹ and Teruo Koyanagi² and Takayuki Osogami³

1 INTRODUCTION

Key-value data structures are ubiquitous in many information retrieval, document analysis and applications, ranging from simple keyword search to complex pattern matching and counting *tf-idf* (term frequency – inverse document frequency) scores. This phenomenon is accelerated with the necessity to cope with big textual data analyzed with a distributed computing platform.

This paper addresses the task of allocating key-value itemsets to data structures staged by their frequencies while counting significant itemsets that appear frequently in the data stream. We wish to place itemsets with higher frequencies in faster data structures and to keep track of their frequencies. Approximate counting of objects in data streams is one of basic and popular subjects in the artificial intelligent community, see e.g., [9, 10] that cleverly combine Bloom filters with Morris' approximate counting [7].

Our contribution is a simple extension of a popular stream algorithm called *Lossy Counting* [6, 2], or, LC for such frequency tracking. Where in LC items are grouped into two stages, in our extension they are divided into multiple stages according to their frequencies over the stream: few high-frequency items in high stage, and the rest in middle or low stages. We call our algorithm the *Multi-Stage Lossy Counting* (MSLC). The algorithm identifies items in high stage and keeps track of their frequencies. It stores items in middle and low stages without their frequencies (their approximation can be inferred from their stages), and therefore string compression can be applied. We show non-trivial theoretical proofs that the simple extension of LC into multiple stages can still result in good approximate counting. We also confirm by experiments that it is effective for processing big text data streams.

2 PROBLEM SETTING

We are given a datastream $(e_1, e_2, \dots, e_N, \dots)$ whose length is not known in advance. Here, N denotes the number of items in the data-stream that has been observed. Each item e_i corresponds to a key in the key-value dataset.

The parameters of MSLC are an error parameter, ϵ , and a support threshold, s , such that $0 < \epsilon \ll s < 1$. These parameters are the same as those in the original lossy counting [6] and its variants. The data stream is divided into substreams, or buckets, that each contains $1/\epsilon$ items. Buckets are numbered from 1 up to b_{curr} , and b denotes the number of bucket such that $1 \leq b \leq b_{curr}$.

\mathbf{H} denotes the set of items that expectedly have high frequency (i.e., it contains all items with approximate frequencies $\geq sN$) in

the stream. Unique to MSLC are \mathbf{M} , that contains items having *middle* frequency, and a parameter θ , that determines the range width of frequency of items stored after seeing a bucket of $1/\epsilon$ of items.

If the average frequency of an item $e \in \mathbf{H}$ per bucket is more than θ , then it stays in \mathbf{H} . For each item $e \in \mathbf{H}$, its frequency since it is inserted to \mathbf{H} , denoted by f_e , and its approximated frequency before the insertion, denoted by Δ_e , are maintained. Namely, the elements of \mathbf{H} are the list of (e, f_e, Δ_e) tuples.

\mathbf{M} contains the subset \mathbf{M}_b for $1 \leq b \leq b_{curr}$, where each \mathbf{M}_b is expected to hold items whose frequencies range from $(b-1)\theta$ to $b\theta$.

3 MULTI-STAGE LOSSY COUNTING (MSLC)

We first describe theoretical properties of MSLC: the number of items in the high stage \mathbf{H} is bounded above by a constant and independent of the length of the stream (Theorem 1), and those in the middle stages \mathbf{M}_b is about the same as in LC (Theorem 2). Moreover, its accuracy is comparable to LC as shown in Lemma 2.

For ease of explanation, we first describe a *basic* MSLC that uses ϵN number of \mathbf{M}_b but only uses at most $1/(\epsilon\theta)$ space for \mathbf{H} as in Algorithm 1.

When seeing an item e of the bucket b_{curr} , the algorithm keeps its approximate count in \mathbf{H} in the form of (e, f, Δ) , where f and Δ are adjusted according to whether it is already in \mathbf{H} , or is found in \mathbf{M} . At the end of processing the bucket, \mathbf{H} is cleaned so that it only contains items whose approximate frequencies, $(f + \Delta)$, are more than $\epsilon N\theta$. Clearly, there are at most $1/(\epsilon\theta)$ of such items. We call such \mathbf{H} to be *compact*.

Next, when deleting an item (e, f, Δ) from \mathbf{H} , we place it into an \mathbf{M}_b that can approximate its true frequency. Namely, \mathbf{M}_b is such that $(b-1)\theta < f + \Delta \leq b\theta$. Notice that all items in each \mathbf{M}_b do not need separate counters, and compressing textual key-value pairs is possible. We call such \mathbf{M}_b to be *compact*. Grouping items in the same \mathbf{M}_b stems from the observation that if (e, f, Δ) and (e', f', Δ') in \mathbf{H} satisfy $f + \Delta \approx f' + \Delta'$ within some error guarantee, then keeping separate counters for e and e' is unnecessary.

We present the analysis of basic MSLC by showing a series of technical lemmas and theorems whose proofs are omitted. We first show the compactness of \mathbf{M}_b 's and \mathbf{H} as below, that leads to bounding the size of \mathbf{H} .

Lemma 1. *For any $b \leq b_{curr}$, if $e \in \mathbf{M}_b$, then*

$$\|\mu_b - t_e\| \leq \frac{\theta b_{curr}}{2}, \text{ and } t_e > \frac{\mu_b}{2} - \frac{\theta}{4}.$$

Lemma 2. *If $(h, f_h, \Delta_h) \in \mathbf{H}$, then*

$$f_h + \Delta_h - \frac{\epsilon N\theta}{2} \leq t_e \leq f_h + \Delta_h + \frac{\epsilon N\theta}{2}.$$

Theorem 1. *At the end of each bucket, the size of \mathbf{H} is at most $\frac{10}{\epsilon\theta}$.*

¹ IBM Research - Tokyo, email: raymond@jp.ibm.com

² Ecrebo Ltd., email: teruok@gmail.com

³ IBM Research - Tokyo, email: osogami@jp.ibm.com

Algorithm 1 Basic Multi-Stage Lossy Counting(Basic MSLC)

Input: $0 < s, \epsilon < 1$ such that $\epsilon \ll s$, and $\theta > 0$

Ensure: Compact \mathbf{H} and Compact \mathbf{M} 's.

1: **Initialization**

2: $w = \lceil 1/\epsilon \rceil$

3: $\mathbf{H} = \{\}$

4: $\mathbf{M}_b = \{\}$

5: $b_{curr} = 1$

6: $j = 0$

7: **for all** e in the stream **do**

8: **if** $(e, f_e, \Delta_e) \in \mathbf{H}$ **then**

9: $\mathbf{H} \leftarrow (e, f_e + 1, \Delta_e)$

10: **else if** $e \in \mathbf{M}_b$ for $b < b_{curr}$ **then**

11: $\mathbf{H} \leftarrow (e, 1, \mu_b)$

12: remove e from \mathbf{M}_b

13: **else**

14: $\mathbf{H} \leftarrow (e, 1, 0)$

15: **end if**

16: $j = j + 1$

17: **if** $j == w$ **then**

18: **for all** $(h, f_h, \Delta_h) \in \mathbf{H}$ **do**

19: **if** $\theta(b-1) < (f_h + \Delta_h) \leq \theta b$ for some $b \leq b_{curr}$ **then**

20: $\mathbf{M}_b \leftarrow h$

21: remove h from \mathbf{H}

22: **end if**

23: **end for**

24: $j = 0$

25: $b_{curr} = b_{curr} + 1$

26: **end if**

27: **end for**

Algorithm 1 requires storing all items in \mathbf{H} or \mathbf{M}_b . However, by slightly modifying the algorithm and ignoring \mathbf{M}_b for $b < \epsilon N/4$, it can be shown that the compactness of \mathbf{H} and \mathbf{M}_b 's still hold.

When discarding \mathbf{M}_b for $b \leq \epsilon N/4$, we lose the information on the range of true frequencies of their elements. Accordingly, we can adjust Line 14 of Algorithm 1 to insert $(e, 1, (b_{curr}/4 - 1/2)\theta)$ into \mathbf{H} when e is not found in \mathbf{M}_b for $b_{curr}/4 < b \leq b_{curr}$. Even though the true frequencies of items discarded are lost, the first part of Lemma 1 still holds. The second part of Lemma 1 does not hold anymore. However, Theorem 1 still holds because for each $h \in \mathbf{H}$ such that $\Delta_h > \frac{\epsilon\theta N}{2}$ implies that h was a part of \mathbf{M}_{b^*} for $b^* > \epsilon N/2$ prior to its insertion. Therefore, it must hold that $t_h \geq ((\epsilon N + 1)/2 - \epsilon N/4)\theta/2 > \epsilon N\theta/8$, and accordingly the number of such h is at most $8/(\epsilon\theta)$. The number of undiscarded elements in \mathbf{M} is as below.

Theorem 2. *The total number of elements in \mathbf{M}_b 's for $b > \frac{\epsilon N}{4}$ is $\frac{8 \log(\epsilon N)}{\epsilon\theta}$.*

4 EXPERIMENT

We show MSLC is effective for processing text streams. In the experiment, we merged all \mathbf{M}_b 's with $b \geq \epsilon N/4$ into a single \mathbf{M} , and others with $b < \epsilon N/4$ into a single \mathbf{L} , i.e., a three-stage LC. We generated streams of keywords from a NHTSA dataset, a car complaint data publicly available from National Highway Traffic Safety Administration, which is an artificial datastream of 256,113,080 entries including 6,561,453 unique keys. We randomly shuffled the order of these articles, parsed them to obtain their keywords as keys. The metrics we measured were 1) memory size for storing items, and 2) the

average number of items retrievable from stages, called throughput. All algorithms were implemented in Java.

The size of \mathbf{H} was used as a metrics of memory consumption for LC, while the total size of \mathbf{H}, \mathbf{M} was used for MSLC. In the experiment, the parameters s and ϵ were 0.00004 and 0.0004.

Method	Size(MB)	Throughput(per sec)
LC+DA TRIE	177	207,364
LC+LOUDS	67	26,093
LC+Incremental TRIE	132	146,364
MSLC	89	201,613

Table 1. Size and Throughput Comparison

Table 1 shows the memory consumption of MSLC against various LCs whose \mathbf{H} was each implemented with *Double Array (DA) TRIE* (denoted as LC+DA TRIE), *LOUDS* (denoted as LC+LOUDS), and *Incremental TRIE* (denoted as LC+Incremental TRIE). TRIE [3], or prefix tree, is an ordered tree data structure that is efficient for storing strings. Double Array (DA) TRIE [1] has a fast search capability, while LOUDS [5] can compactly store a tree of n nodes using only at most $2n + o(n)$ bits. The incremental TRIE [8, 4] is an online implementation of TRIE. For the MSLC, \mathbf{H} was implemented by a simple Hash Map, \mathbf{M} by DA TRIE, and \mathbf{L} by LOUDS. The table shows that the size of MSLC is about 50% smaller than LC+DA TRIE, and only 33% bigger than LC+LOUDS, but its throughput is almost similar ($\approx 97\%$) to the best one. This confirms the effectiveness of MSLC for allocating patterns according to their frequencies.

We also evaluated the false positives of MSLC to find all items with support frequencies more than sN for various s , and compared it with those of LC. We found that the false positives of MSLC for various θ are higher than LC: they did not converge and oscillated as the datastream increased. The false positive rates of MSLC were higher for larger θ due to the implementation of \mathbf{M}_b 's that utilizes the Bloom Filter. There have been many literatures where Bloom Filters were the key in approximate counting, such as, [9]. It is an interesting future work to balance the false positive rates of the Bloom Filter and MSLC.

ACKNOWLEDGEMENTS

A part of this research was supported by JST, CREST.

REFERENCES

- [1] Jun-ichi Aoe, ‘An efficient digital search algorithm by using a double-array structure’, *IEEE Transactions on Software Engineering*, **15**, 1066–1077, (1989).
- [2] Graham Cormode and Marios Hadjieleftheriou, ‘Finding frequent items in data streams’, *PVLDB*, **1**(2), 1530–1541, (2008).
- [3] Edward Fredkin, ‘Trie memory’, *Communications of the ACM*, **3**, 490–499, (1960).
- [4] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, and Setsuo Arikawa, ‘On-line construction of symmetric compact directed acyclic word graphs’, in *SPIRE*, pp. 96–110. IEEE Computer Society, (2001).
- [5] Guy Joseph Jacobson, *Succinct static data structures*, Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1988.
- [6] Gurmeet Singh Manku and Rajeev Motwani, ‘Approximate frequency counts over data streams’, in *VLDB*, pp. 346–357, (2002).
- [7] Robert Morris, ‘Counting large numbers of events in small registers’, *Communications of the ACM*, **21**(10), 840–842, (1978).
- [8] Kunihiko Sadakane, ‘Dynamic succinct ordinal trees’, *IEICE technical report. Theoretical foundations of Computing*, **109**(9), 37–41, (2009).
- [9] David Talbot, ‘Succinct approximate counting of skewed data’, in *IJCAI*, pp. 1243–1248, (2009).
- [10] Benjamin Van Durme and Ashwin Lall, ‘Probabilistic counting with randomized storage’, in *IJCAI*, pp. 1574–1579, (2009).