Count-Min Sketch

An Implementation and Refinement

Algorithmic Techniques for Massive Data
(CS-5390)

Instructed By

Dr. Subrahmanyam K

Authors

Joseph K J
cs17mtech01001@iith.ac.in

Gautam Muduganti
cs17resch01003@iith.ac.in

Abstract

Count-Min sketch is a sublinear space data structure that can be used to summarize the frequency of items in a data stream. Its uses constant amount of space that does not vary with the size of the stream. It guarantees an ε-approximation of the actual frequency statistic of any element in the stream with very good confidence bound.

In this project work, we focus on the following

- A Java implementation for the Count-Min Sketch
- Integration with various data streams which include inhouse data streams and cloud based data streams from a data source called PubNub
- Conservative Count-Min which builds on top on the standard Count-Min sketch which limits the space used by the hash functions

Keywords: Count-Min Sketch, Sublinear Algorithm, Conservative Count-Min, PubNub, Java

1. Introduction

Count-Min Sketch uses an efficient way to keep track of the number of items that passed by in the stream, that scales well the number of items in the stream. The size of the data structure is solely dependent on accuracy guarantees of the frequency count. The accuracy guarantee is specified by two user specified values, ε and δ, meaning that the error in answering the frequency query is within a factor of ε with a probability δ.

1.1. Data Structure

The Count-Min Sketch is an array of counters of width w and depth d. The width and depth is computed using the ε and δ values provided by the user. Each entry in initialised to zero. The number of rows in the array correspond to the number of hash function that we use and the number of column in the number of buckets to which each item is hashed into. It is also possible that the user might specify the width and depth and the algorithm provides a guarantee on the ε and δ values based on that.

1.2. Update Procedure

When we see an element i in a stream, i is hashed with all the d hash functions. The counter at each of the hash location is incremented by the count c passed into the update procedure. For adding an element, d updates are made in the data structure.

$$\forall 1 \leq j \leq d : count\left[j, h_j(i_t)\right] \leftarrow count\left[j, h_j(i_t)\right] + c_t$$

1.3. Update Procedure

When querying for an element i, we hash i with each of the d hash functions. Then we take the minimum of all the values that are at the hashed location.

$$\min_{1 \leq j \leq d} count[h, h_j(i)]$$

2. Conservative Count-Min

Count-Min Sketch with conservative update is a modification of the vanilla Count-Min Sketch. It helps to avoid unnecessarily updating counter values and hence reduce the overestimation error. The query procedure for Conservative Count-Min is the same as Count-Min. However, to update an item i, with frequency c, we first run the query for i. This will give the frequency f of i in the current data structure. The count is updated only if the count the count at the index is greater than $f + c$.

Count is update according to the formula:

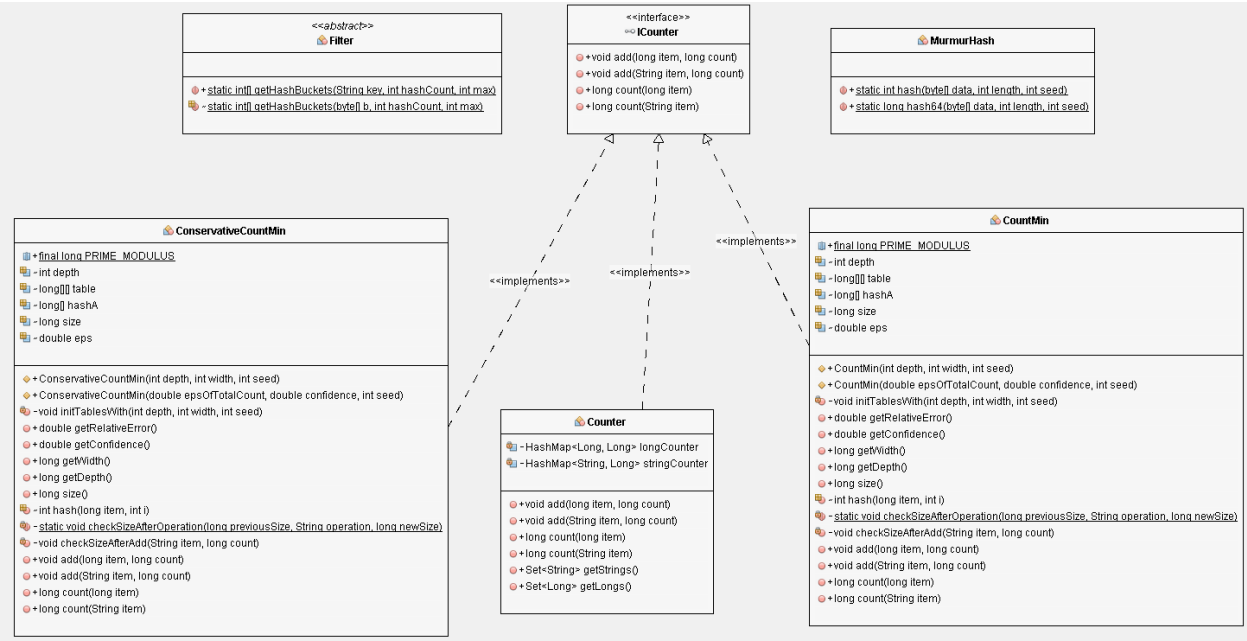
$$count[k, h(x)] \leftarrow \max\{count[k, h(x)], f + c\}$$

The intuition is that since the point query returns the minimum of all the d values, we update a counter only if it is necessary. This will reduce overestimation and limit the size of each of the buckets.

3. Implementation

We implemented a Count-Min Sketch which can support both numeric and string streams. The implementation is based on the paper *Approximating Data with the Count-Min Data Structure* [2]. The source code is available at - https://github.com/cs17resch01003/pubnub_countmin.

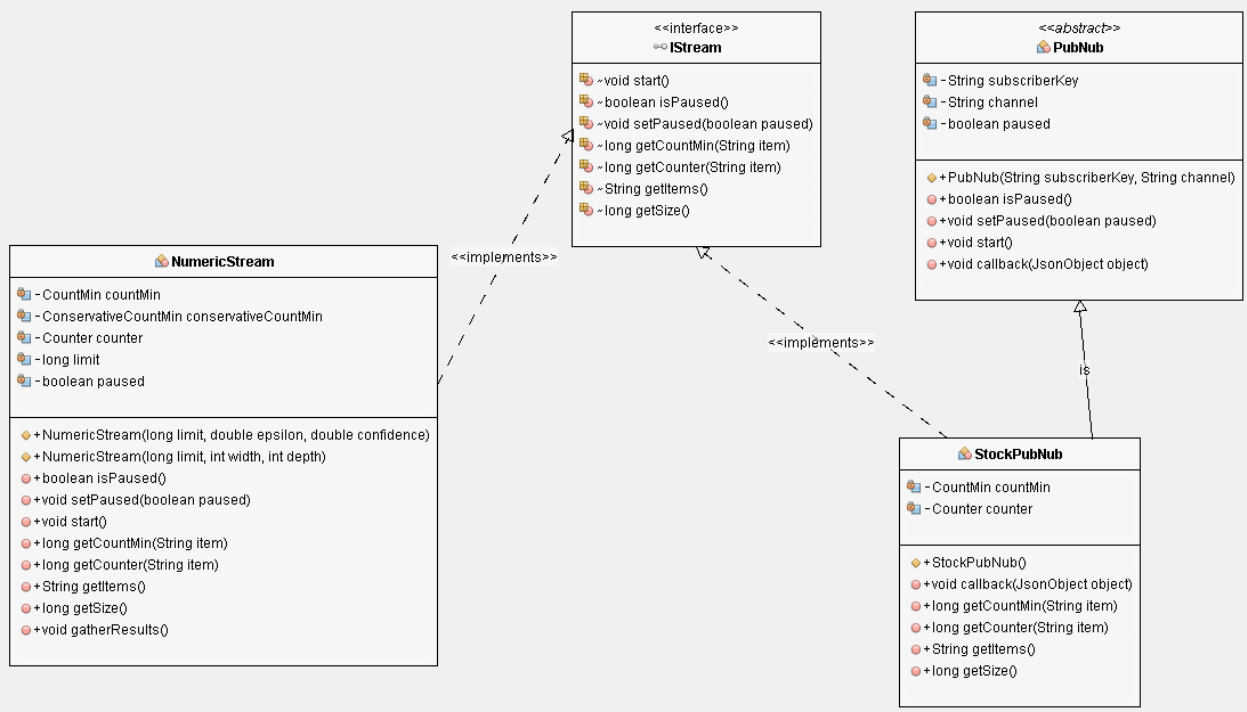
3.1. Class Diagram



The counters implement the ICounter interface and we have three different counters -

- Counter - The standard counter which does exact counting
- CountMin - The counter which implements the Count-Min Sketch
- ConservativeCountMin - The counter which implements the Conservative Count-Min Sketch

MurmurHash is a standard algorithm which is used to hash Strings.



All the streams implement the IStream interface and there is a standard abstract class PubNub which all implementations going to PubNub inherit from. We have two streams implemented as a part of this project -

- NumericStream - A stream which generates random numbers with a given limit
- StockPubNub - A publicly available stream which constantly sends the price movements of various commodities

Each of these streams internally uses the three counters discussed above to count the frequency of elements.

4. Experimental Evaluation

As a part of the experimental evaluation, we run the algorithm with different widths and depths and calculated the mathematical value of ε and δ and the actual values of ε and δ. All experiment was done on the numeric stream with at-least 5 million items coming through the stream and the number of distinct elements which the stream was formed was varied through the experiments. Here are the values that were considered for the depth, width and distinct elements -

- Depths: 10, 20, 50, 100, 250, 500, 1000, 2500, 5000, 10000
- Widths: 1, 2, 5, 10, 12, 15, 18, 20, 25
- Distinct Elements: 100, 1000, 10000, 100000

Sample Results:

distinct elments	width	depth	blocks	epsilon	actual epsilon	epsilon diff	error entities	confidence	actual confidenc	confidence diff
100000	1000	1	1000	0.0020	0.0024920	-0.000492	100000	0.500000	0.9800000	0.480000
100000	1000	2	2000	0.0020	0.0009900	0.001010	0	0.750000	1.0000000	0.250000
100000	1000	5	5000	0.0020	0.0009660	0.001034	0	0.968750	1.0000000	0.031250
100000	1000	10	10000	0.0020	0.0008850	0.001115	0	0.999023	1.0000000	0.000977
100000	1000	12	12000	0.0020	0.0008830	0.001117	0	0.999756	1.0000000	0.000244
100000	1000	15	15000	0.0020	0.0008820	0.001118	0	0.999969	1.0000000	0.000031
100000	1000	18	18000	0.0020	0.0008810	0.001119	0	0.999996	1.0000000	0.000004
100000	1000	20	20000	0.0020	0.0008800	0.001120	0	0.999999	1.0000000	0.000001
100000	1000	25	25000	0.0020	0.0008690	0.001131	0	1.000000	1.0000000	0.000000
100000	2500	1	2500	0.0008	0.0009910	-0.000191	100000	0.500000	0.9800000	0.480000
100000	2500	2	5000	0.0008	0.0003910	0.000409	0	0.750000	1.0000000	0.250000
100000	2500	5	12500	0.0008	0.0003750	0.000425	0	0.968750	1.0000000	0.031250
100000	2500	10	25000	0.0008	0.0003440	0.000456	0	0.999023	1.0000000	0.000977
100000	2500	12	30000	0.0008	0.0003400	0.000460	0	0.999756	1.0000000	0.000244
100000	2500	15	37500	0.0008	0.0003360	0.000464	0	0.999969	1.0000000	0.000031
100000	2500	18	45000	0.0008	0.0003360	0.000464	0	0.999996	1.0000000	0.000004
100000	2500	20	50000	0.0008	0.0003270	0.000473	0	0.999999	1.0000000	0.000001
100000	2500	25	62500	0.0008	0.0003250	0.000475	0	1.000000	1.0000000	0.000000

The complete set of results has 360 different experiments: Experimental Results.xls

As it can be seen from the experiment results, the actual values of both ε and δ are within the expected values thereby proving that the implementation works.

5. Summary

As a part of this project, we implemented the Count-Min and Conservative Count-Min sketches and verified their working through experimental evaluation. We can see that the values of ε and δ are significantly impacted with the number of hash functions and the width of each of them. We noticed that increasing the hash functions from 1 to 2 increases the confidence from 0.5 to 0.75 and from 2 to 3 brings it to about 0.968. For all practical purposes at-least 3 hash functions would be required for a proper working of the algorithm.

6. References

[1] Sketch Algorithms for Estimating Point Queries in NLP
Graham Cormode, Amit Goyal, Hal Daume III
Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, July 2012

[2] Approximating Data with the Count-Min Data Structure
Graham Cormode and S. Muthukrishnan
IEEE Software, published by the IEEE Computer Society