An Implementation for "Approximate Medians and other Quantiles in One Pass and with Limited Memory" [MRL98] †

†Note: This is assignment 1 report for Statistical Techniques for Data Science and Robotics Course at Innopolis University for Spring 2022
Note: Unorganized quotes from the papers and sources is available at: Notions link

Note: The implementation is available at: GitHub repository link

Hany Hamed
Robotics Institute
Innopolis University
Russia
h.hamed@innopolis.university

***Index Terms*—Data profiling, approximate quantile, streaming model, Deterministic algorithm.**

I. INTRODUCTION

Quantile is an important statistic to compute, quantiles help to provide a description for the unknown data distribution [CZ20]. A Quantile is the value of an element at a certain position if the sequence was sorted. Therefore, quantiles are used for data profiling to find the metadata for the provided stream of data to understand more properties of the distribution of the data. However, computing the quantile is not an easy task to solve in case of large-scale data due to the memory constraint and the Computational complexity. Therefore, an approximation for quantile can be used with error guarantees, multiple methods has been described in the literature [CZ20] and they have different set of models (Streaming, and Distributed) and randomness in the algorithm (Deterministic, and Randomized).

I have chosen [MRL98] algorithm to implement it. This algorithm is a deterministic algorithm that means that it will generate the same quantile with the same dataset provided and the same parameters. Furthermore, it is based on a streaming model such that the data elements arrives one by one in a streaming fashion, and the algorithm is required to answer the quantile query with only one-pass scan [CZ20].

The rest of the paper will be organized as following: Section II describes the algorithm procedure and the implementation notes, and Section III explains the obtained results and the complexity analysis for the algorithm.

II. METHODOLOGY

The algorithm mainly consists of 3 main parts: **Buffers** to store data, **Main Operations** to deal with the data and generate the approximated quantile, and **Collapse Policy** which is the criteria for when to use the NEW/COLLAPSE operations, the collapse policy for [MRL98] is referred as "New Algorithm" in Section 3.4 Collapse Policy in [MRL98].

The main operations for the algorithm: NEW, COLLAPSE, and OUTPUT, furthermore, these operations have been the backbone for many other algorithms but with different collapse policy [CZ20].

[MRL98] is using buffers of data to store the streaming data with amount of b buffers that can store at most k elements. One of the disadvantages for this algorithm that it requires the prior knowledge of the length N of data stream.

Each buffer has the following property: empty to indicate if is it full or empty, weight to indicate the sum of weights of buffer collapsed to create this buffer.

A. *NEW*

This operation is called to fill the streamed data to the empty available buffers. It takes ek such that e is the number of empty buffers and k is the maximum number of elements can be stored in the buffer. This operation set the buffer as full and set its weight to 1. Note: if the elements is less than k then it is augmented with equal number of negative and positive infinity values (This infinity value is represented by the maximum possible value for the data multiplied and added to constants to act as infinity). This augmentation intuitively is done for the last batch from the streamed data as the length of the stream is not necessarily to be $N\%bk = 0$, there might be some number of elements at the end that are less than k elements. The augmented stream new length will be equal to βN such that $\beta \geq 1$

B. *COLLAPSE*

This operation is called to merge c buffers into one buffer such that $c \geq 2$ and free the c buffers. The weight of the resulted buffer is equal to the sum of the c buffers. COLLAPSE mainly repeats the elements of each buffer to create buffer's weight copies of that element and sort this list of elements

and take offset spaced elements from that list. The offset is computed based on the resulted buffer weight if it is odd or even, in case of even, there is 2 formulas to calculate the offset that the choice of performing the formula is alternating when executing this formula two time consecutively.

$$\text{odd\_offset} = \lfloor (W+1)/2 \rfloor$$

$$\text{even\_offset1} = \lfloor (W)/2 \rfloor$$

$$\text{even\_offset2} = \lfloor (W+2)/2 \rfloor$$

Such that W is the resulted buffer weight, $W = \sum_i^c W(X_i)$, $W(X_i) =$ weight of the buffer X_i

C. *OUTPUT*

This is called at the end of the algorithm to compute the approximate quantile value when there is no any available data in the stream to process, it takes c buffers as COLLAPSE and merge them as well into one buffer and compute the index for the quantile stored in the resulted buffer and return the value of that quantile. The index base on ϕ‘ is being computed as following:

$$\phi' = \frac{2\phi + \beta - 1}{2\beta}$$

such that ϕ is the quantile required to compute for original the stream of data, ϕ‘ is the quantile of the augmented stream of data as it has been proved the guarantee of error ϵ in [MRL98], and β has been explained previously in NEW operation subsection.

And the index (idx) formula is as following:

$$idx = \lceil phi_approx * self.k * W \rceil$$

D. *Collapse Policy*

The collapse policy is the criteria or the main algorithm that describes how the NEW and COLLAPSE operations are performed. The main collapse policy of [MRL98] is referenced as the ”New Algorithm” as [MRL98] propose a new algorithm that is better than the previous mentioned algorithms in their paper Munro and Paterson [MP80], and Alsabti, Ranka and Singh [ARS97]. The only difference between [MRL98] and [MP80] and [ARS97] is the collapse policy.

In the implemented algorithm [MRL98], the collapse policy is as following:

Each buffer has an extra property called level that is initialized by 0 for all the buffers. The algorithm proceed in calculating the number of empty buffers and find the minimum value of the levels among the buffers.

Then, if the number of empty buffers is equal to 1, perform NEW operation on that buffer with at most k elements from the stream and set its level to the minimum level. If the number of empty buffers is greater or equal to 2, perform NEW operation on these buffers with with at most ck elements from the stream and set its level to 1, such that c is the number of empty buffers. If there is no empty buffers (i.e. all of them are full), then we perform COLLAPSE operation on all these buffers and clear and empty all the buffers and store the resulted buffer in one of the newly emptied buffers and set its level to minimum level computed + 1. In [MRL98] has been proofed the guarantee of this algorithm for finding an approximate solution with ϵ of error.

Furthermore, the extra feature of the levels that it can represent the buffers history in a tree based graph.

III. Results

[MRL98] has been identified as an algorithm with $\mathcal{O}(\frac{1}{\epsilon}\log^2(\epsilon N))$ space complexity, this complexity is associated with the error guarantee, moreover the algorithm is only using $O(bk)$ space such that b is the number of buffers and k is the number of elements inside the buffers.

To generate data, I have implemented a Streamer and Generator classes that generate and stream the generated data by batches, the data is being generated randomly with specified ranges for values.

After running the algorithm with multiple parameters as it is mentioned in [MRL98], I have obtained the following results shown in Table I (Note: it is at the last page due to its width.

Moreover, I have generated plot from table I and it is shown in fig. 1 and fig. 2 shows the running time of the algorithm for different N of data and different parameters adapted from table I.

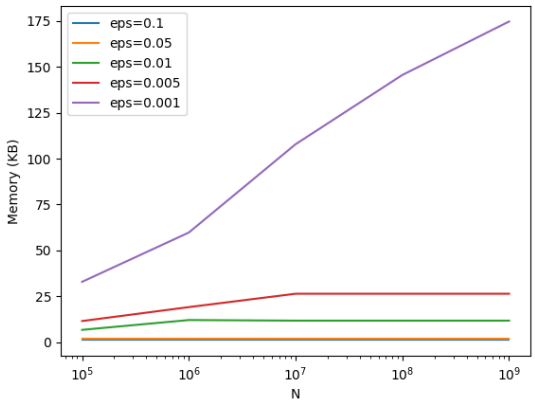

Fig. 1. Memory used with different parameters of the algorithm as it is shown in Table I

Moreover, I have verified and tested if the algorithm returned the correct quantile or not empirically using small data set (randomly generated, and sorted sequence of numbers), also I have observed with huge dataset with small range of data, we can verify if the approximate quantile is correct or not due to pigeonhole principle and that the probability of having all the elements in that small range in the huge dataset is increasing with the increase of the dataset and the decrease of the range of possible values for the elements in the datastream.

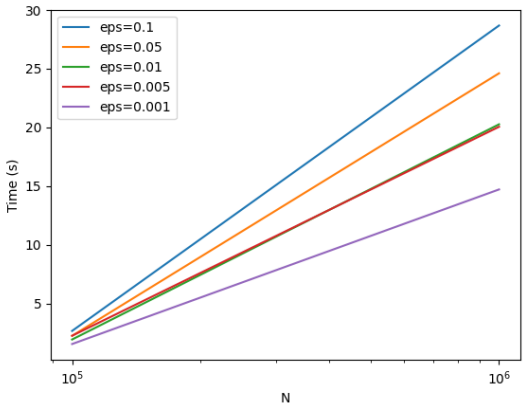

Fig. 2. Running time with different parameters of the algorithm as it is shown in Table I

IV. CONCLUSIONS

Quantile computation is a vital to understand the data distribution, however, it is hard to compute and requires special algorithms. This paper explains [MRL98] algorithm implementation and shows implementation details, furthermore, it verifies the complexity analysis for the algorithm.

REFERENCES

[MP80] J Ian Munro and Mike S Paterson. "Selection and sorting with limited storage". In: *Theoretical computer science* 12.3 (1980), pp. 315–323.

[ARS97] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. "A one-pass algorithm for accurately estimating quantiles for disk-resident data". In: (1997).

[MRL98] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. "Approximate medians and other quantiles in one pass and with limited memory". In: *ACM SIGMOD Record* 27.2 (1998), pp. 426–435.

[CZ20] Zhiwei Chen and Aoqian Zhang. "A survey of approximate quantile computation on large-scale data". In: *IEEE Access* 8 (2020), pp. 34585–34597.

TABLE I
TABLE FOR MEMORY CONSUMPTION FOR DIFFERENT PARAMETERS

	Number of buffers (b)					**Size of buffer (k)**					**Total Memory (KB)**				
eps, N	"1e5"	"1e6"	"1e7"	"1e8"	"1e9"	"1e5"	"1e6"	"1e7"	"1e8"	"1e9"	"1e5"	"1e6"	"1e7"	"1e8"	"1e9"
0.1	5	5	5	5	5	31	31	31	31	31	1.152344	1.152344	1.152344	1.152344	1.152344
0.05	5	5	5	5	5	76	76	76	76	76	2.03125	2.03125	2.03125	2.03125	2.03125
0.01	7	12	6	6	6	217	229	472	472	472	6.699219	12.046875	11.71875	11.71875	11.71875
0.005	3	8	7	7	7	953	583	937	937	937	11.496094	19.09375	26.386719	26.386719	26.386719
0.001	3	5	5	9	9	2778	3031	5495	4114	4943	32.882812	59.746094	107.871094	145.617188	174.761719