ICIC Express Letters ICIC International ©2023 ISSN 1881-803X
Volume **17**, Number **9**, September **2023** pp. **997–1003**

APACHE STORM-BASED DISTRIBUTED SAMPLING LIBRARY FOR DATA STREAMS

Hyojong Moon, Myeong-Seon Gil and Yang-Sae Moon*

Department of Computer Science and Engineering
Kangwon National University
1 Gangwondaehakgil, Chuncheon-si, Gangwon-do 24341, Korea
{ moonhyojong; gils }@kangwon.ac.kr; *Corresponding author: ysmoon@kangwon.ac.kr

Received November 2022; accepted January 2023

Abstract. *Recently, large data streams are rapidly generated in various fields such as Internet of Things (IoT) and social network services (SNS). Analyzing all such data streams is very inefficient, and we generally pre-process the data to reduce its huge value. In this paper, we first present a distributed processing-based sampling library for effective use of large data streams. We then apply and implement the proposed library to Apache Storm, a real-time distributed processing system. To validate the efficiency of the sampling library, we extract samples from real data streams and compare the graphs of the extracted samples with the original population. Experimental results show that the sample data using the proposed library presents a similar pattern reflecting the characteristics of the original data well. We believe that our library is an excellent result for easily sampling large data streams in a distributed environment.*

Keywords: Data stream, Sampling, Distributed processing, Apache Storm

1. **Introduction.** A data stream [1] is a set of data items that are constantly generated at high speed. Because these data streams are rapidly generated and accumulate continuously, we need a high processing cost to utilize the data. Accordingly, many interests in a sampling technique that can effectively reduce the amount of data streams are increasing.

Sampling [2-4] is a statistical technique that extracts a sample representing a population (original data). Typical sampling algorithms include Systematic [5], Reservoir [6], and Binary Bernoulli [7] sampling. Along with these sampling techniques, a computational environment essential for data stream processing is a distributed processing system. In distributed processing, a number of servers connected by a network communicate with each other and process multiple tasks simultaneously. Specifically, Apache Storm [8], Apache Spark [9], and Apache Flink [10] are distributed processing systems applicable to the stream environment, and in this paper we present a real-time sampling library based on Storm[1].

The proposed Storm-based sampling library consists of four detailed steps. First, when the original data stream is input through the *Source-In* step, the parameters required for each sampling are defined in the *Parameter-Setting* step. Next, the original data is actually sampled in the *Sampling-Execution* step, and the result is converted and output in the data stream format in the *Sample-Out* step. These steps of our sampling library are described in detail in Section 3.

The proposed real-time sampling library is evaluated through various experiments using real data. In the experiment, we evaluate whether each algorithm in the sampling library has sampled well the data representing the population. As a result of the experiment, we

DOI: 10.24507/icicel.17.09.997

[1]We provide the library in the github (https://github.com/dke-knu/i2am/tree/master/i2am-app/SamplingAlgorithm).

confirmed that the samples extracted from each sampling algorithm and the population showed very similar patterns even in a distributed environment. Based on these results, we believed that the proposed sampling library is an excellent research that allows users to more easily use sampling techniques through a real-time distributed processing system.

Contributions of the paper can be summarized as follows. First, we propose a new sampling library that supports stream purification in a distributed environment. Second, we design and implement each sampling algorithm in Apache Storm using its components of spouts and bolts. Third, through actual data-based experiments, we show that the proposed library performs sampling well while reflecting the characteristics of the original data.

The rest of the paper is organized as follows. Section 2 describes the related work on Apache Storm and sampling algorithms. Section 3 presents the proposed library for stream data sampling in a distributed environment. Section 4 shows the results of experimental evaluation. Finally, Section 5 concludes the paper.

2. **Related Work.** Apache Storm [8,11-13] is an open-source distributed framework developed by Twitter. The definition of the work required for the data input-processing-output process in Storm is called a topology. This topology consists of a combination of multiple spouts and bolts. Figure 1 shows the operating structure of the Storm based on spouts and bolts, and this structure becomes a topology. In the topology, the spout receives data from the data stream source, converts the data into tuples which are data types used by Storm, and transfers the tuples to the next bolt. The bolt receives data from the spout or the previous bolt, processes the task defined by users, and passes the results to the next bolt or the final output. For the detailed operating process of Apache Storm, refer to [8].

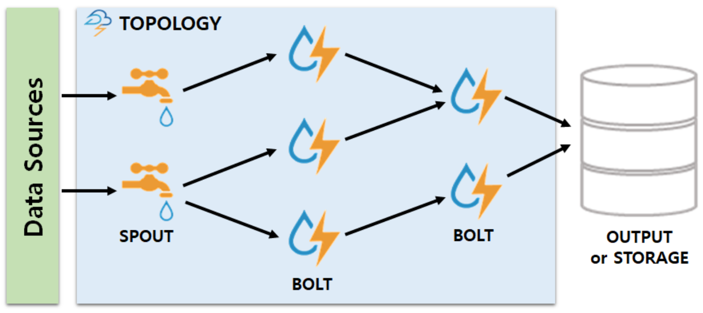


FIGURE 1. The operating structure of Apache Storm

Sampling is a statistical technique for selecting data that well reflects the characteristics of a given population. Figure 2 shows sampling algorithms that are frequently used in specific areas such as statistics and dynamic streams. As shown in the figure, the sampling techniques can be divided into sampling specialized for batch processing of the entire population, sampling specialized for the data stream environment, and sampling used for multimedia data. Among the sampling algorithms, clustering and stratified sampling specialized for batch processing classify the entire data into relevant groups and sample the data for each group. In addition, sampling algorithms specialized for data streams that focus on processing speed include Reservoir [6], Binary Bernoulli [7], and KSample [14]. In this paper, we focus on sampling algorithms suitable for the data stream environment from the existing algorithms and compose them into a sampling library. For example, since statistics-based algorithms often require the entire population for sampling, we choose the systematic sampling that does not use the entire population. Finally, we have selected seven algorithms: statistically-based Systematic [5], stream-based Reservoir, Hash [15], Priority [16], Binary Bernoulli, KSample, and UC KSample [18]. In particular, Binary

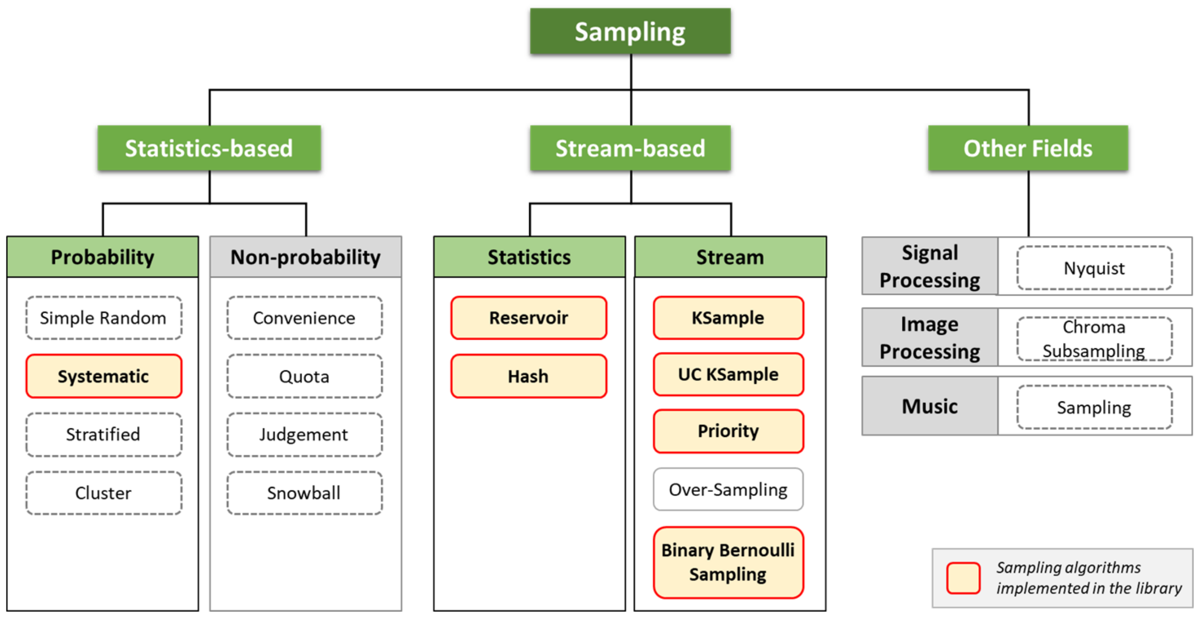


Figure 2. Sampling algorithms for each field

Bernoulli and UC KSample are new algorithms that improve sampling performance in a stream environment. However, most of the selected algorithms do not support the distributed stream environment. To solve this problem, in this paper, we design and implement each algorithm with Apache Storm, and propose a new library that integrates these algorithms. We explain the algorithms and our library in more detail in Section 3.

3. Storm-Based Sampling Library.

Before explaining the proposed sampling library, we briefly describe each sampling algorithm implemented in the library. We target seven sampling algorithms in this paper, and the description of each algorithm is as follows.

- *Systematic Sampling*: Select the k-th data item as a sample from the input population.
- *Reservoir Sampling*: It is based on simple random sampling, but it has a fixed number of samples and performs random sampling so that the sample size does not increase even when the population size increases.
- *Priority Sampling*: It is a stream sampling algorithm similar to Reservoir, and a sample extraction method that applies an additional variable called data frequency to simple random sampling.
- *Hash Sampling*: It is an effective technique for sampling text data. In Hash Sampling, a specific data field is converted to a number using a hash function, and if the value is less than a specified value, a sample is selected.
- *Binary Bernoulli Sampling*: As a method of sampling data streams with the same probability in a multi-source environment, we use an improved algorithm [17] that solves the existing Binary Bernoulli bottleneck and network delay problem.
- *KSample*: It selects a sample while maintaining a sampling rate in the data stream environment in which the population continuously increases.
- *UC KSample* [18]: It is a recent algorithm that guarantees the reliability of samples by improving the problem that the existing KSample does not guarantee uniformity reliability.

In particular, we optimize each algorithm to operate in Storm's distributed processing environment, and use the improved algorithms for high sampling accuracy. For example, in the case of Hash Sampling, in which the sampling result varies depending on the hash function, three hash functions with different characteristics are provided to improve the sampling accuracy. In addition, the Binary Bernoulli and KSample algorithms, which are stream-based sampling techniques, construct a library using the latest advanced algorithms.

To integrate these sampling algorithms into a library that can be used to process data streams, we use Apache Storm. The proposed sampling library consists of four stages, and the overall operation structure of these stages is shown in Figure 3. First, in the Source-In step, the sampling library converts the data stream received through the spout into a tuple, and transmits it to the next bolt. Next, the Parameter-Setting step defines necessary parameters before starting sampling, and transmits the data to the next step. The transmitted data is sampled through the sampling algorithm bolt selected by the user in the Sampling-Execution step. The extracted sample is output in the form of a data stream in the Sample-Out step, or stored in a storage device set by the user.

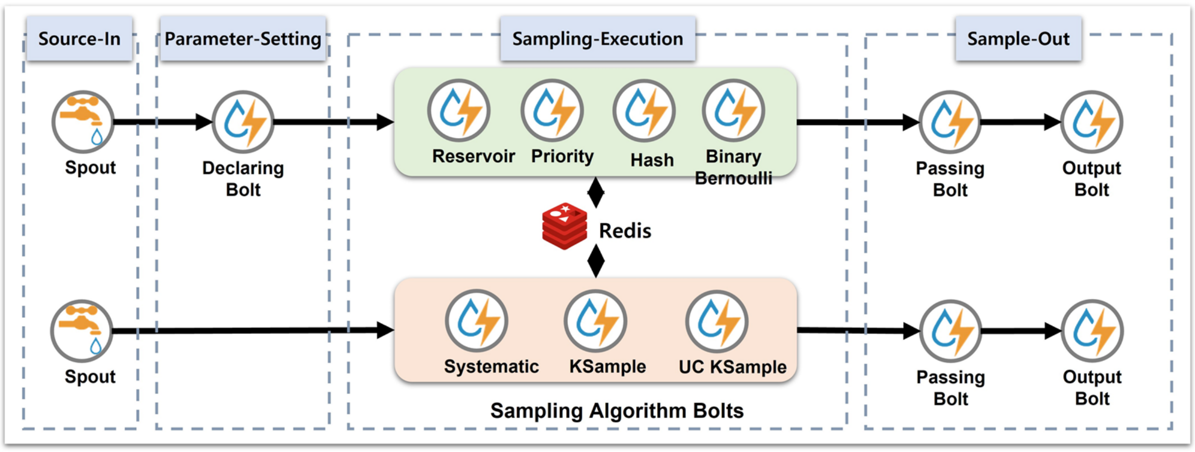


FIGURE 3. Overall structure of Storm-based sampling library

We describe in detail the Storm components of each step. Declaring bolt manages the parameters required for the sampling algorithms. This bolt collects and defines the necessary parameters for each sampling algorithm, and delivers the defined parameters to the sampling algorithm bolt. As shown in Figure 3, in this paper we adopt Redis [19] as an in-memory DBMS for parameter sharing of each algorithm. In the case of the Systematic, KSample, and UC KSample algorithms, because the algorithm bolt reads the user-specified values from Redis without additional parameter settings, the spout delivers data directly to the sampling algorithm bolt without going through the Declaring bolt.

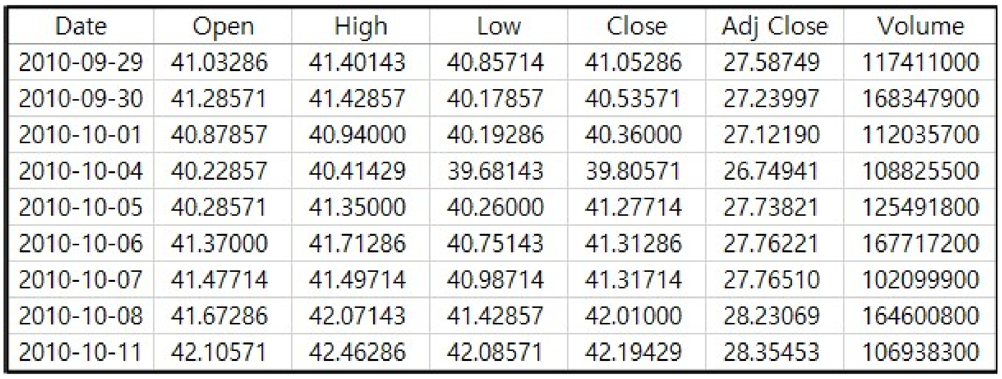
The sampling algorithm bolt is a component that samples an input data stream through a selected sampling algorithm. As mentioned above, seven algorithms of Reservoir, Priority, Hash, Binary Bernoulli, Systematic, KSample, and UC KSample are processed by each corresponding bolt. Here, the Reservoir, Priority, Hash, and Binary Bernoulli samplings use a fixed sample size (number), and they extract a sample of a user-specified size from a window designated for processing the stream. On the other hand, KSample and UC KSample use a fixed sampling rate, and they extract samples as much as the sample rate set by the user. When sampling is completed in the sampling algorithm bolt, the bolt sends the sample to the passing bolt in the Sample-Out step.

The passing bolt converts the extracted list structure sample into a data stream and delivers it to the output bolt. Since the samples received from the sampling algorithm bolt are used as a list structure for internal operation, the process of converting back to the data stream should be necessary. Therefore, the passing bolt converts the sample into a data stream, and the output bolt serves to output the received data stream to a storage location.

As explained so far, we designed a new sampling library by analyzing existing data stream-based sampling algorithms applicable to the data stream environment. We also improved and optimized the algorithms for distributed processing systems. If we need to add a new sampling technique, we can support the function easily in the proposed sampling library by implementing the algorithm as a new Storm bolt.

4. Performance Evaluation.

In the experiment, we evaluate whether the proposed sampling library accurately samples the original data even in the Storm-based distributed environment. The hardware platform consists of one master node (Xeon E5-2630V3 2.4GHz, 8core) and eight slave nodes (Xeon E5-2630V3 2.4GHz, 6core), each of which is equipped with 32GB RAM and 256GB SSD. The software platform of nine nodes is CentOS Linux 7.2.1511. The data used in the experiment is the closing price data of Apple's stock from 2010 to 2018. Figure 4 shows an example of stock data used in the experiments.



Date	Open	High	Low	Close	Adj Close	Volume
2010-09-29	41.03286	41.40143	40.85714	41.05286	27.58749	117411000
2010-09-30	41.28571	41.42857	40.17857	40.53571	27.23997	168347900
2010-10-01	40.87857	40.94000	40.19286	40.36000	27.12190	112035700
2010-10-04	40.22857	40.41429	39.68143	39.80571	26.74941	108825500
2010-10-05	40.28571	41.35000	40.26000	41.27714	27.73821	125491800
2010-10-06	41.37000	41.71286	40.75143	41.31286	27.76221	167717200
2010-10-07	41.47714	41.49714	40.98714	41.31714	27.76510	102099900
2010-10-08	41.67286	42.07143	41.42857	42.01000	28.23069	164600800
2010-10-11	42.10571	42.46286	42.08571	42.19429	28.35453	106938300

FIGURE 4. An example of Apple's stock data

The parameters of each algorithm are shown in Table 1. All parameters in the table are user-defined. Since the proposed library uses stream data, real-time input data is collected and sampled in a fixed size buffer called window. Among the parameters, WindowSize means the size of this fixed buffer, and SampleSize means the number of data to be sampled in the window. In this paper, we perform experiments by fixing the WindowSize and SampleSize to 2000 and 500, respectively.

TABLE 1. Parameters for each sampling algorithm

Algorithm name	Parameter (Value used in the experiment)
Reservoir	WindowSize (2000), SampleSize (500), SampleKey (id)
Priority	WindowSize (2000), SampleSize (500), SampleKey (id)
Hash	WindowSize (2000), SampleSize (500), SampleKey (id)
Systematic	Interval (5)
Binary Bernoulli	WindowSize (2000), SampleSize (500), SampleKey (1st-id), PreSampleKey (2nd-id)
KSample	SamplingRate (10)
UC KSample	SamplingRate (10), UCLowerBound (0.7)

Figure 5 shows the original data (population) and the sampled data extracted by each sampling. The graph on the top-left is the original Apple's stock data, and the other graphs are sampled data extracted by seven sampling algorithms. Comparing the population shown with the sample extracted, we can intuitively confirm that the trend and shape of the original and sampled data are very similar. As a result of analyzing the distribution between the actual original and sampled data based on JSD (Jensen-Shannon Divergence),

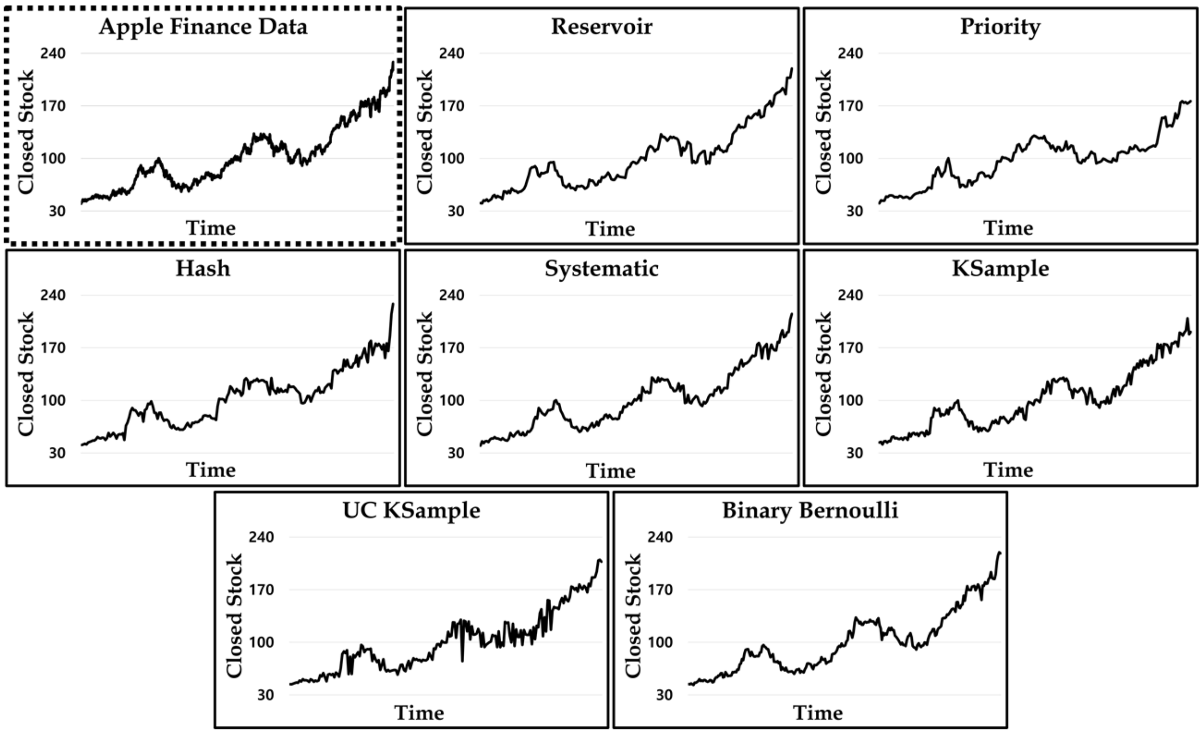


FIGURE 5. Population and sample graph for each sampling algorithm

the error rate range for each algorithm is only 0.1% to 1.6%. This indicates that each algorithm in the sampling library proposed performs relatively accurate sampling on data streams even in the top of the Storm's distributed environment.

5. Conclusions.

Since the amount of data streams is constantly increasing, we need an efficient pre-processing method that can reduce the processing cost of such data streams. In this paper, we designed an integrated library of sampling algorithms that can be applied to the stream environment to efficiently process these data streams, and implemented them in Apache Storm, a real-time distributed processing system. As a result of the experiments with the proposed sampling library using real data, we confirmed that it shows similar patterns when comparing the trends and shapes of the original data and the sampled data. From these results, we can see that the proposed sampling library performs accurate sampling even in the distributed environment. Using the Storm-based sampling library proposed in this paper, we believe that Naïve users without expert knowledge of sampling algorithms can more easily purify data streams in the distributed environment. In the future, we plan to design and implement an integrated purification library by improving the performance speed and function of the proposed library and building a filtering library that effectively purifies the data streams according to the user's intention.

Acknowledgment. This work was partly supported by Institute of Information & communications Technology Planning & evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00859, Development of a Distributed Graph DBMS for Intelligent Processing of Big Graphs) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2022R1A2C1003067).

REFERENCES

[1] J. Leskovec, A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*, Cambridge University Press, 2014.

[2] G. Cormode and N. Duffield, Sampling for big data: A tutorial, *Proc. of the 20th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, New York, NY, DOI: 10.1145/2623330.2630811, 2014.

[3] M. S. Mahmud, J. Z. Huang, S. Salloum, T. Z. Emara and K. Sadatdiynov, A survey of data partitioning and sampling methods to support big data analysis, *Big Data Mining and Analytics*, vol.3, no.2, pp.85-101, 2020.

[4] R. Jayaram and D. Woodruff, Perfect L_p sampling in a data stream, *SIAM Journal on Computing*, vol.50, no.2, pp.382-439, 2021.

[5] D. R. Bellhouse and J. N. Rao, Systematic sampling in the presence of a trend, *Biometrika*, vol.62, pp.694-697, 1975.

[6] J. S. Vitter, Random sampling with a reservoir, *ACM Transactions on Mathematical Software*, vol.11, pp.37-57, 1985.

[7] G. Cormode, S. Muthukrishnan, K. Yi and Q. Zhang, Optimal sampling from distributed streams, *Proc. of the 21st ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, Indianapolis, IN, pp.77-86, 2010.

[8] *Apache Storm*, http://storm.apache.org/, 2023.

[9] *Apache Spark*, http://spark.apache.org/, 2023.

[10] *Apache Flink*, http://flink.apache.org/, 2023.

[11] A. Muhammad and M. Aleem, BAN-Storm: A bandwidth-aware scheduling mechanism for stream jobs, *Journal of Grid Computing*, vol.19, no.3, pp.1-16, 2021.

[12] S. Son and Y.-S. Moon, Locality/fairness-aware job scheduling in distributed stream processing engines, *Electronics*, vol.9, no.11, Article No. 1857, 2020.

[13] S. Son, H. Im and Y.-S. Moon, Stochastic distributed data stream partitioning using task locality: Design, implementation, and optimization, *The Journal of Supercomputing*, vol.77, no.10, pp.11353-11389, 2021.

[14] T. R. Kepe, E. C. Almeida and T. Cerqueus, KSample: Dynamic sampling over unbounded data streams, *Journal of Information and Data Management*, vol.6, no.32, pp.32-47, 2015.

[15] T. Zseby, M. Molina, N. Duffield, S. Niccolini and F. Raspall, *Sampling and Filtering Techniques for IP Packet Selection RFC 5475*, Technical Report, IETF, 2009.

[16] E. Cohen, Stream sampling for frequency cap statistics, *Proc. of the 21st ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, New York, NY, pp.159-168, 2015.

[17] W. Cho, M.-S. Gil, M.-J. Choi and Y.-S. Moon, Storm-based distributed sampling system for multi-source stream environment, *International Journal of Distributed Sensor Networks*, vol.14, no.11, DOI: 1550147718812698, 2018.

[18] H. Kim, M.-S. Gil, Y.-S. Moon and M.-J. Choi, Variable size sampling to support high uniformity confidence in sensor data streams, *International Journal of Distributed Sensor Networks*, vol.14, no.4, DOI: 1550147718773999, 2018.

[19] *Redis*, http://redis.io/, 2023.