ICIC Express Letters
Part B: Applications
Volume **10**, Number **1**, January **2019**

ICIC International ©2019 ISSN 2185-2766
pp. 55–61

APACHE STORM CONFIGURATION PLATFORM FOR DYNAMIC SAMPLING AND FILTERING OF DATA STREAMS

Youngkuk Kim, Siwoon Son and Yang-Sae Moon*

Department of Computer Science
Kangwon National University
1 Gangwondaehakgil, Chuncheon-si, Gangwon-do 24341, Korea
{ ygkim; ssw5176 }@kangwon.ac.kr; *Corresponding author: ysmoon@kangwon.ac.kr

Received July 2018; accepted October 2018

Abstract. *With the launch of the big data era, in recent years real-time data streams are being used in various fields. It is not practical to collect and process the whole of these big data streams. Thus, there is a need for a sampling method for extracting a good sample and/or a filtering method for extracting necessary data from the entire data stream. Apache Storm is a real-time distributed parallel processing framework for processing large data streams. However, Storm needs to modify the source code, redistribute it, and restart the process when changing the structure or algorithm of the input data. In this paper, we describe the problems of sampling and filtering methods in Storm environment, and define the requirements to solve them. In addition, we design a novel plan model consisting of the input, processing, and output modules in the data stream. Our proposed plan manager has features that can dynamically create, execute, and monitor the plan visually through the Web UI (Web User Interface).*

Keywords: Big data, Data stream, Apache Storm, Sampling, Filtering, Distributed processing

1. Introduction.

Recently, a large volume of data streams is quickly being generated in SNS (Social Network Service), finance, sensors, and IoT (Internet of Things) applications. A data stream is the data that is constantly generated continuously, and it is very inefficient to store and process the entire data stream. Therefore, it is effective to use a sampling method that extracts a sample that reflects characteristics and patterns of the original data stream well, and a filtering method that extracts only the data that satisfies the given condition. However, as the generation rate and amount of data stream increase, there is a practical limitation to sampling and filtering in real time on a single server.

Velocity, one of the characteristics of big data [1], means that data is generated so fast that it cannot be processed by existing legacy systems. Due to this characteristic of big data, distributed processing of data streams is being actively researched. Apache Storm [2] and Apache Spark Streaming [3] are representative examples of distributed processing frameworks for real-time data streams. In particular, Apache Storm is a distributed parallel framework that allows large data streams to be processed and analyzed in real time on distributed servers. These distributed processing systems focus on processing and analyzing data streams, so they often use a separate queueing system for I/O of data streams. Apache Kafka [4] is a distributed message queueing system that specializes in real-time log processing, and it is widely used as an input or pipeline for a data stream processing framework [5].

In this paper, we propose a plan manager that can dynamically process sampling and filtering methods on top of Storm. Since Storm defines the collecting and processing features of data streams together in a single module, there is a problem that, if the data

DOI: 10.24507/icicelb.10.01.55

structure or processing algorithm needs to be changed, the source codes of the program should be modified and redistributed. This problem may cause some data streams to be lost until the process is restarted. To solve this problem, the plan manager separates the data collection module from the processing module, and we can increase the availability of Storm. Thus, even if the processing module is restarted by changing the algorithm, the data collection module continuously gathers data streams. Our work can be summarized as follows. First, we derive the problems that can arise when we simply develop the sampling and filtering methods in Storm or existing environment. These problems are non-economic, unavailability, and functional decentralization. To solve these problems, we define the requirements for flexible integration of data collection and processing, and the integration of external frameworks such as Kafka and databases with Storm. We then propose a plan manager, a data stream refinement system that satisfies these requirements and dynamically configures sampling and filtering features. Plan manager provides a Web UI, which enables users to create, modify, delete and, monitor plans visually and dynamically.

The rest of the paper is organized as follows. In Section 2, as the related work, we describe sampling and filtering algorithms first. We also explain open source distributed computing frameworks: Apache Storm, Apache Kafka, and StreamFlow. Section 3 derives the problem caused by the simple development of sampling and filtering with existing environment of Storm. Section 4 designs a plan manager, the proposed data stream refinement platform. Finally, Section 5 concludes the paper and discusses future work.

2. **Related Work.**

2.1. **Sampling and filtering methods.**

Sampling is a statistical technique that extracts some of the data representing the population. Such sampling can be classified into batch-based and data stream-based methods. Examples of batch-based sampling [6,7] include systematic sampling, stratified sampling, and cluster sampling. On the other hand, examples of data stream-based sampling [6,7] are reservoir sampling, Hash sampling, K sampling, and priority sampling. Also, binary Bernoulli sampling is specialized in a distributed stream environment. In a data stream environment, it is effective to exploit the sampling method because it is inefficient to process and store the entire data due to the memory limitation.

Filtering is a technique that extracts only the data that satisfies a given condition in the data. Typically, query filtering is used to select only data that satisfies the user's query. Also, Bloom filtering is used to probabilistically calculate whether data belongs to a set, and Kalman filtering to estimate the original data from data that contains errors such as sensor data.

2.2. **Open source distributed computing framework.**

Apache Storm [2] is a representative distributed processing system for processing data streams in real time. Storm represents a series of tasks from the input to the output of data as a topology, and the topology consists of a number of spouts and bolts. Figure 1 shows an example architecture of Storm topology. First, the spout receives data from the source of the data stream, converts it into a tuple, which is the data type used by Storm, and sends it to the bolt. Next, the bolt processes the tuple received from the spout or bolt and transfers it to the next bolt, or sends the result to the output or storage. Storm has a problem that if the structure of the input data changes, the source code needs to be modified. In other words, when the source code is changed, it is necessary to newly construct and redistribute the topology, which incurs a serious problem that the service is interrupted.

Apache Kafka [4] is a distributed messaging system specialized for processing a large volume of log data generated in real time. Kafka is designed as a distributed architecture, which makes the system easy to decentralize, and server replication is fast. In addition,

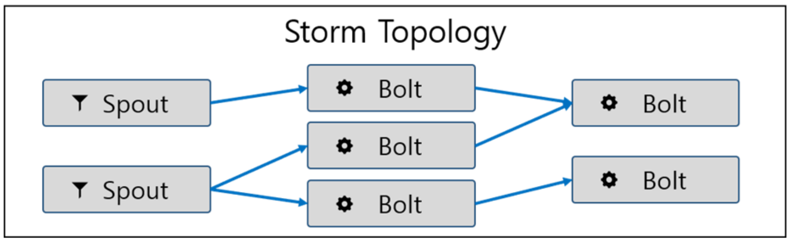


Figure 1. Topology architecture of Apache Storm

by storing log data on a disk rather than a memory, it is possible to guarantee high data stability compared to an in-memory based message queue, and to perform both a message queue role and a log collector role.

StreamFlow [9] provides a responsive Web interface as an open source tool for easily creating and monitoring Storm topologies. In other words, it provides a drag-and-drop topology builder to create a new topology. It also provides a dashboard for topology state and performance monitoring and log analysis. However, the new version has not been released since 2015, and not all features have been implemented.

3. Problem and Requirement Analysis.

In this section, we derive the problems of existing data collection and processing techniques and present the requirements to solve them. First, existing collection and processing techniques may cause the following problems.

- *Non-economic problem* arises when examining the entire data stream. Processing all the continuously generated data streams is very difficult in terms of storage capacity limitation and low throughput. Therefore, there is a strong need for a preprocessing process that extracts only meaningful or user-required data.
- *Unavailability problem* occurs when the collecting and processing of data streams are defined together. This means that a service interruption may occur when a change in the data structure or processing algorithm is required. These changes require the source code of the program to be modified, and the entire process running on the system must be stopped and restarted. In this case, a continuously generated data stream can be lost until the process is normally restarted.
- *Functional decentralization problem* is that big data platforms have been developed for different purposes, so we need to integrate multiple platforms into one system as needed. The data streams must be managed as a single system, from input to processing to output. On the other hand, the streaming platforms such as Storm and Spark are specialized for processing data streams, and they can lead to data loss if not used with a separate platform for input and output. Therefore, a separate platform with a queuing function must be used together.

In this paper, we define the following three requirements in order to solve the above three problems. First, *data stream refinement* for solving the non-economic problem extracts only necessary data from a large volume of data streams by applying sampling and filtering methods. Second, *flexibility* to address the unavailability problem allows the data stream collection, processing, and storage modules to operate independently to each other. Third, *integration* for solving the functional decentralization makes it possible to use the platforms required for the collection, processing, and storage of data streams as a single system. In other words, Storm, Kafka, and database systems can be managed in one integrated system, and Web UI is introduced for this integration. In this paper, we propose a plan manager, a sampling and filtering system for data stream refinement based on Apache Storm, which satisfies these three requirements.

4. Design of Plan Manager.

In this section, we design a plan manager, a proposed data stream refinement system. Figure 2 shows the structure of the plan manager. The plan manager consists of three modules.

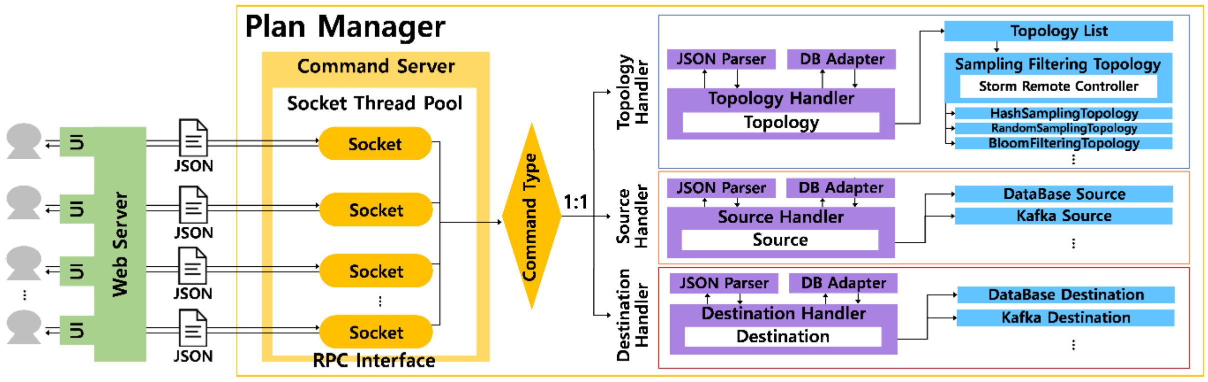


FIGURE 2. Detailed structure of the plan manager

The operation steps of the proposed plan manager are as follows. First, Web UI receives the source information of data streams from the user. Through this step, the schema of the data streams is received and stored in the database. Second, Web UI receives destination information for the processed data from the user and stores it in the database. Third, a plan is constructed by receiving sampling and/or filtering algorithms with appropriate parameters to be applied to the data streams, and the server stores it in the database. Fourth, when the user executes the plan, the server sends the command to the Storm cluster. Storm reads the data schema and sampling and filtering parameters from the database. Fifth, when the data input is executed in the Web UI, the server reads the data and inputs it to Kafka. The Storm topology reads the data that are input to Kafka to perform sampling and/or filtering algorithms. Finally, the processed data stream is output to the destination defined above.

In this section, first, Section 4.1 describes a plan that is a logical flow for refining a user's data stream. Next, Sections 4.2 to 4.4 describe each module in detail.

4.1. Plan.

A plan is a model for managing the input, processing, and output operations of a data stream as one. The plan consists of three modules as shown in Figure 3. First, a source stores and manages the information and schema of the input data stream and reads the data stream. Next, a topology is a Storm-based sampling and filtering algorithm that extracts parts of the data stream. Table 1 shows the sampling and filtering methods provided by the plan manager. Finally, a destination manages the information of the repository that outputs the data stream extracted by the topology, and works to export the data stream. The plan consists of one source and one destination, and one or more topologies. Each module operates independently and transmits data streams through Kafka. With this structure, even if a specific module is interrupted, it does not affect other modules.

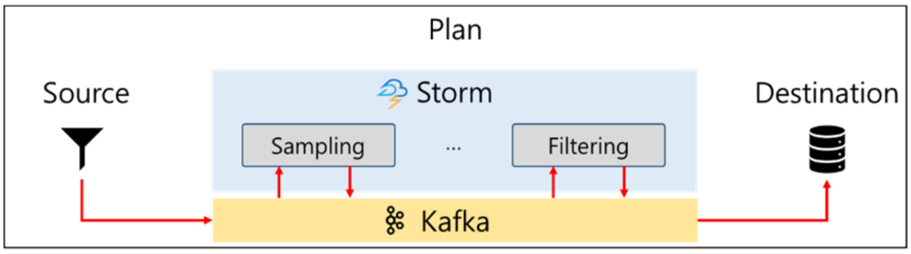


FIGURE 3. Generic structure of a plan

Table 1. Sampling and filtering methods provided by the plan manager

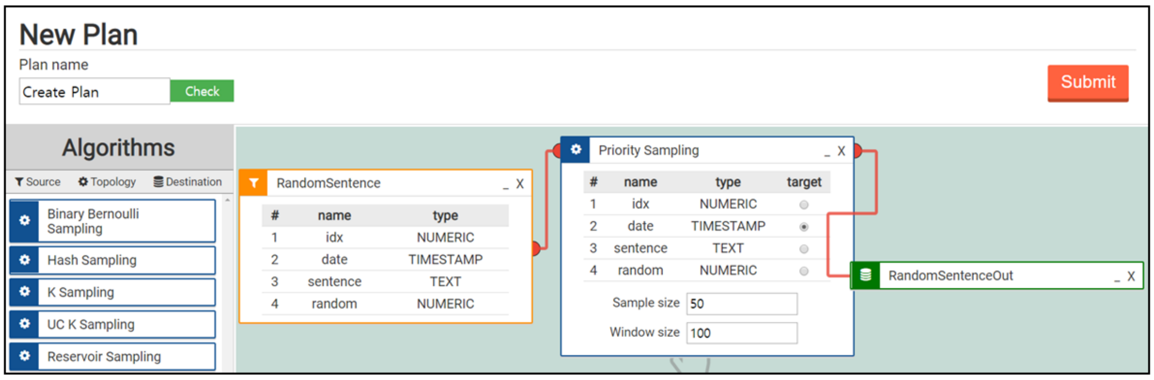
Sampling methods	Filtering methods
■ Hash sampling	■ Bloom filtering
■ K sampling	■ Kalman filtering
■ Priority sampling	■ Query filtering
■ Reservoir sampling	
■ Systematic sampling	
■ Binary Bernoulli sampling	

4.2. **Web user interface module.** Web UI provides users with the ability to visually manage their plans. First, in the step of user authentication, the user creates an account and logs in. In this step, our system identifies the user through personal information such as ID, name, and password. After logging in, the user is able to check each information in the plan, source, and destination menus.

In the source menu, the user can see the names, creation and modification dates, and execution status of the sources. In the source creation page, the user inputs the name of the source. Next, the user specifies the type of the source, which includes Kafka, database, and customizing source. Of these, the customizing source allows the user to input data streams directly into Kafka. Finally, the user needs to specify the schema of the data stream. In this step, the user inputs the name and data type of the data column. We support TEXT, NUMERIC, and DATE as data types at this stage.

In the destination menu, the user can see the names, the creation and modification dates, and the execution status of the destinations. Destination creation is similar to source creation. Since the schema of the message does not change after processing, the step of defining data schema is omitted.

In the plan menu, the user can see the name, the creation and modification date, and the execution status of the plan. In the plan creation page, the user inputs the name of the plan. And it shows the list of source and destination, sampling and filtering topology created by the user. The user selects elements to be used by doing drag and drop, and constructs a plan via connection between elements. In the case of sampling, additional parameters for sampling should be given as input parameters. In the case of filtering, the user should give an appropriate query. Figure 4 shows an example of creating a plan in Web UI.



RandomSentence

#	name	type
1	idx	NUMERIC
2	date	TIMESTAMP
3	sentence	TEXT
4	random	NUMERIC

Priority Sampling

#	name	type	target
1	idx	NUMERIC	○
2	date	TIMESTAMP	●
3	sentence	TEXT	○
4	random	NUMERIC	○

Sample size 50

Window size 100

Figure 4. Example screenshot of creating a plan Web UI

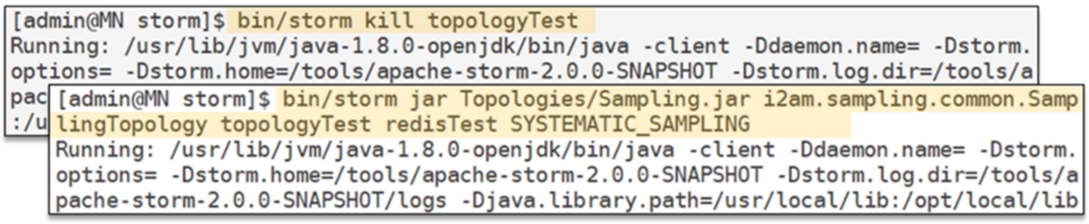
4.3. **RPC interface module.** The RPC interface operates as a server and receives JSON type commands from the Web UI. The interface manages sockets using a thread pool. A thread pool is a technique for allocating threads in advance, which prevents performance degradation due to thread creation and collection operations occurring each time. When a request is entered in the Web UI, the thread pool creates a socket, classifies the command type, and calls the handler that matches the result.

4.4. **Handler module.** The topology handler manages the plan information and refers to the source, topology, and destination information contained in the plan. It also manages the sampling and filtering topology with their parameters, and sends topology active and deactive commands to the Storm cluster. When the plan is activated, Storm executes the topology according to the received command. The topology then reads the source topic information from the database, takes the data stream, and applies a sampling or filtering algorithm. It then outputs the processed data stream to Kafka.

The source handler manages the information of the source that holds or generates the data stream. It is also responsible for creating, modifying, deleting, and executing the source. When the user activates the source, it creates a thread, reads the data stream from the source, and enters it into Kafka.

The destination handler manages the information of the destination, which is the output of the data stream. It is also responsible for creating, modifying, deleting, and executing the destination. When the user activates the destination, it creates and executes a thread that outputs the data stream stored in Kafka to an external destination such as Kafka or a database.

4.5. **Simulation results.** As shown in Figure 5, the existing Storm needs to create a topology for processing the data stream and deploy the work through a specific Storm cluster console. The commands on the console are very inconvenient and easy to mistake. On the other hand, if the proposed system is used, source creation, destination creation, and plan creation work can be performed through the Web UI as shown in Figure 4. After creating the source, destination, and plan, the user can simply start collecting and processing the data streams. Using plan manager makes it much easier to use and manage data streams than existing console-based operations.



```
[admin@MN storm]$ bin/storm kill topologyTest
Running: /usr/lib/jvm/java-1.8.0-openjdk/bin/java -client -Ddaemon.name= -Dstorm.
options= -Dstorm.home=/tools/apache-storm-2.0.0-SNAPSHOT -Dstorm.log.dir=/tools/a
pac
:/u
```

```
[admin@MN storm]$ bin/storm jar Topologies/Sampling.jar i2am.sampling.common.Samp
lingTopology topologyTest redisTest SYSTEMATIC_SAMPLING
Running: /usr/lib/jvm/java-1.8.0-openjdk/bin/java -client -Ddaemon.name= -Dstorm.
options= -Dstorm.home=/tools/apache-storm-2.0.0-SNAPSHOT -Dstorm.log.dir=/tools/a
pache-storm-2.0.0-SNAPSHOT/logs -Djava.library.path=/usr/local/lib:/opt/local/lib
```

Figure 5. Example of console-based Storm operation

The proposed system can solve the non-economic, unavailability, and functional decentralization problems described in Section 3. First, we solve the non-economic problem by providing sampling and filtering techniques that can be applied to data streams. Second, we solve the non-availability problem by applying the plan model to operating the data collection, processing, and output modules independently. Third, we solve the functional decentralization problem by integrating and managing Storm and Kafka into a single unified system.

5. **Conclusions.** In this paper, we proposed and designed the plan manager, a sampling and filtering system for Storm-based data stream refinement. We also derived the problems of non-economic, unavailability, and functional decentralization that occur when processing the data stream, and defined the requirements to solve this problem. Plan manager was able to provide Storm-based sampling and filtering algorithms to collect and refine data streams. In addition, we used Kafka to construct the plan by grouping the source that were data sources, the topology based on Storm-based algorithms, and

the destination that stored results. Plan manager provided a Web UI to visually create, modify and monitor these plans.

As the future work, we will actually implement the plan manager according to the design. We then verify the practicality of collecting through the plan manager and showing whether the refined data stream well represents the population. In addition, we will analyze the problem to occur in the implemented plan manager and improve it.

Acknowledgment. This work was partly supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2016-0-00179, Development of an Intelligent Sampling and Filtering Techniques for Purifying Data Streams) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2017R1A2B4008991).

REFERENCES

[1] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh and A. Byers, *Big Data: The Next Frontier for Innovation, Competition, and Productivity*, Technical Report, McKinsey Global Institute, 2011.

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

[3] *Apache Spark Streaming*, https://spark.apache.org/streaming/.

[4] *Apache Kafka*, http://kafka.apache.org/.

[5] *Storm and Kafka Together: A Real-Time Data Refinery*, https://hortonworks.com/blog/storm-kafka-together-real-time-data-refinery/.

[6] P. J. Haas, Data-stream sampling: Basic techniques and results, *Proc. of Data Stream Management*, Berlin, Germany, pp.13-44, 2016.

[7] R. E. Sibai et al., Sampling algorithms in data stream environments, *Proc. of Int'l Conf. on Digital Economy*, Carthage, Tunisia, pp.29-36, 2016.

[8] R. Cheng et al., Filtering data streams for entity-based continuous queries, *IEEE Trans. Knowledge and Data Engineering*, vol.22, no.2, pp.234-248, 2010.

[9] *Streamflow*, https://github.com/lmco/streamflow/.