Doctoral Symposium: Pre-processing and Data Validation in IoT Data Streams

Philsy Baban
Databases and Information Systems Group
TU Ilmenau, Germany
philsy.baban@tu-ilmenau.de

ABSTRACT

In the last few years, distributed stream processing engines have been on the rise due to their crucial impacts on real-time data processing with guaranteed low latency in several application domains such as financial markets, surveillance systems, manufacturing, smart cities, etc. Stream processing engines are run-time libraries to process data streams without knowing the lower level streaming mechanics. Apache Storm, Apache Flink, Apache Spark, Kafka Streams and Hazelcast Jet are some of the popular stream processing engines. Nowadays, critical systems like energy systems, are interconnected and automated. As a result, these systems are vulnerable to cyber-attacks. In real-world applications, the sensing values come from sensor devices contains missing values, redundant data, data outliers, manipulated data, data failures, etc. Therefore, our system must be resilient to these conditions. In this paper, we present an approach to check if there is any above mentioned conditions by pre-processing data streams using a stream processing engine like Apache Flink which will be updated as a library in future. Then, the pre-processed streams are forwarded to other stream processing engines like Apache Kafka for real stream processing. As a result, data validation, data consistency and integrity for a resilient system can be accomplished before initiating the actual stream processing.

CCS CONCEPTS

• **Information systems** → **Data cleaning**; **Integrity checking**; *Stream management*.

KEYWORDS

Stream processing, data pre-processing, data validation, resiliency

ACM Reference Format:
Philsy Baban. 2020. Doctoral Symposium: Pre-processing and Data Validation in IoT Data Streams. In *The 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20), July 13–17, 2020, Virtual Event, QC, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3401025.3406443

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DEBS '20, July 13–17, 2020, Virtual Event, QC, Canada
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8028-7/20/07…$15.00
https://doi.org/10.1145/3401025.3406443

1 INTRODUCTION

In the Big Data era, data comes from different sources such as sensors, devices, customer databases, click-stream logs, transactional applications, mobile applications, social media, etc. mostly in real-time. And this enormous amount of data is analyzed depending on the use case. The results or conclusions gathered during the data analysis will be useful only when the data received from the data sources are valid and consistent. Thus to ensure the data quality, data must be validated before initiating any other operations. Also, the storage of unlimited data is difficult. Therefore, this unbounded data is processed in real-time and retain only the useful bits using a stream processing engine. Stream processing engines are run-time libraries to process streaming data without dealing with lower-level streaming mechanics. Stream processing can either be stateless or stateful. In stateless processing, data streams can be processed independently while in stateful processing, the information gathered from the processed records are also used.

In this paper, we discuss our approach based on the energy systems domain. The integration of new IoT technologies to the existing Power Grid results in a reliable and more flexible Smart Grid that enables the two-way flow of electricity and data. Due to this adaptation, electric meters are now smart meters with sensors that can send data records about the energy consumption or energy generation on a defined time interval. As a result, the direct inspection of energy meters is no longer required. At the same time, the quality of data is reduced due to the presence of imperfections, redundancies or inconsistencies caused by sensor failures, anomalous situations, exogenous factors, cyber attacks like data manipulation, etc [7]. The data record may contain zero or negative values as the amount of energy consumed, the amount of energy generated greater than the total generator capacity, missing data, continuous arrival of the same records, etc. are some of the examples of imperfections in data streams. Therefore the incoming records must be validated before processing the data records. For this, data pre-processing is the solution. Otherwise, the low quality data provide incorrect results.

Data pre-processing [10] refers to the processing of data streams that contain missing data, noisy data, inconsistent and redundant data. Data transformation and discretization are the widely used data pre-processing algorithms. Feature selection is one of the main categories in data transformation: it selects only the relevant features and non-redundant attributes. Data discretization is defined as a process of dividing the domain of the variables into a finite set of non-overlapping intervals with minimal loss of information.

In this field of research, we propose an approach to perform data validation as a part of pre-processing to validate the streaming data received from the data sources like smart meters, to enhance the

quality of data. The remaining part of this paper is organized as follows: Section 2 overviews the related work in the area of data pre-processing; Section 3 introduces our data pre-processing approach; Section 4 provides the preliminary results; Section 5 concludes the paper and proposes some future work.

2 STATE OF THE ART

Data pre-processing [10] is one of the main tasks in data mining as it enhances the quality of data. Apache Spark, Apache Flink, Apache Kafka, Apache Samsa, and Apache Storm are some of the main stream processing engines. Apache Storm [9] is the oldest real-time stream processing engine that is suitable for non-complicated streaming use cases. Even though it has very low latency and high throughput, it does not support state management and advanced features like aggregations, sessions, etc. In Apache Kafka [6], Kafka Streams API is the client library used to process data stored in Kafka clusters. It is fault-tolerant and supports stateful processing. Though Kafka Streams API is a very lightweight library, it is not possible to use Streams API without Kafka. In Apache Samsa [9], a publish/subscribe task is used. It uses Apache Kafka messaging system to offer buffering, fault tolerance, and state storage. Samza is good at maintaining states and provides high throughput. In SPEs, it is possible to add user-defined functions to perform stream processing operations. Since most of the SPE do not have a library of operators to perform data pre-processing, user-defined functions are used to implement the pre-processing tasks.

At the same time, Apache Spark [12] and Apache Flink [5] are hybrid processing systems that support batch and stream processing, where Spark is designed for large-scale in-memory data processing, and Flink focuses on distributed streams and batch data processing. Both SPEs provide libraries known as BigDaPSpark and BigDaPFlink for data pre-processing. BigDaPSpark [8][11] is a library focused on the static data pre-processing built on top of Apache Spark to support feature selection, data reduction to obtain a reduced set of original data, noise filtering, missing values imputation, discretization and imbalanced learning for balancing the dataset. Noise filtering contains two sub-libraries, one for noise removal and the other for noise filtering. BigDaPFlink [8][2] is a library for real-time data pre-processing for Apache Flink. It includes six data pre-processing algorithms based on the discretization and feature selection problems.

The BigDaPSpark and BigDaPFlink libraries mainly concentrate on the feature selection and discretization, and not on data validation. Therefore, our aim is to validate the incoming data as apart of pre-processing.

3 PROPOSED APPROACH

Nowadays, traditional energy systems are transformed into a distributed intelligent energy systems called smart grids where energy meters are now sensors that can send data continuously. As a result, huge amount of data is generated where human supervision is not possible. Thus, the probability of finding inconsistencies and imperfections in streaming data is high. Consequently, low quality of data makes false predictions or incorrect results that can affect the entire functioning of the system.

Following are the common data quality problems,

- *Missing data*: The incoming data stream records may contain missing data fields. For example, if the customer identifier or meter identifier is missing in the record received from the smart meter, then the processing of this record is a wastage of time and resource. Therefore the incoming data must be complete. Otherwise, missing data generates inconsistencies that have a critical impact on the stream processing step.
- *Data outliers*: The data stream records may contain data outliers due to some errors or data manipulation. For example, if the meter reading of the energy generated is greater than the maximum generator capacity, then there are outliers. This has to be identified beforehand.
- *Inconsistent data*: The data records from the energy meters may contain inconsistent data. For example, the amount of energy consumed must not be less than or equal to zero. The reason for inconsistencies may be due to sensor failures, cyber-attacks, etc., and processing of this data results in an inefficient system that helps to bypass security features [3].
- *Irrelevant data*: Data records received from sensors contains data that is not needed for further processing. For example, if the data records consist of data fields such as *device id*, *device id type*, *measured time*, *profile*, *reading reason* and *measured value*. Here, *device id type* is an alternate id to identify the device, *profile* is to specify the profiles as a meter can have multiple profiles and *reading reason* is a status data to show whether it is a periodic reading or a manual reading. In this record, *device id type* and *reading reason* is not needed in the next step of processing. So we should eliminate these data fields. Otherwise, this irrelevant data results in wastage of memory.
- *Duplicate data*: Stream data contains duplicate records. For example, the same data record is received multiple times for processing. This is mainly due to some attacks or failures. Consequently, the processing of duplicate records reduces the performance of an SPE and provides incorrect results.

Thus, to improve the quality of streaming data, data streams must be validated. Since pre-processing of streaming data increases the quality of data, data validation should also be included as a task in data pre-processing. Even though data validation tools are already available, they are not designed in combination with an SPE. As we mentioned in section 2, data pre-processing is performed either using user-defined functions or using libraries. In general, feature selection and discretization are the main two tasks in data pre-processing. But, if we do not have valid data, then the pre-processing is useless. Therefore, data validation in streaming data is significantly important. So, we must validate streaming data along with other tasks in pre-processing. In addition, the storage of data that continuously generated from thousands of data sources like sensors is no longer possible. These unbounded data should be processed either in batch or in real-time. For critical systems like energy systems, the information must be up to date to monitor and discover if there are any problems that must be solved immediately. Therefore, we use real-time stream processing.

Figure 1 shows the proposed architecture of the solution. In the figure, data sources like smart meters generate continuous data streams. Since the storage of data streams are not possible, data

streams are forwarded to a stream processing engine for real-time processing. Data validation and stream processing operations are performed using stream processing engines. In our approach, we use two stream processing engines, one for the data validator and the other for actual stream processing. This is because data validation is not specific to a particular use case. That is, for almost all types of streaming data, the data records must be validated. Presently, there is no SPE that provides a library to perform data validation. Therefore, here we use a lightweight SPE that can be easily integrated to any other SPE and can perform stateful processing where order based processing is required for validation. In a data validator, we check whether the incoming data has any of the conditions such as missing data, inconsistent data, data outliers, irrelevant data, and redundant data. If the data streams do not match any of the conditions, it is then forwarded for actual stream processing. Finally, processed data is either stored in a database or used for monitoring or report generation.

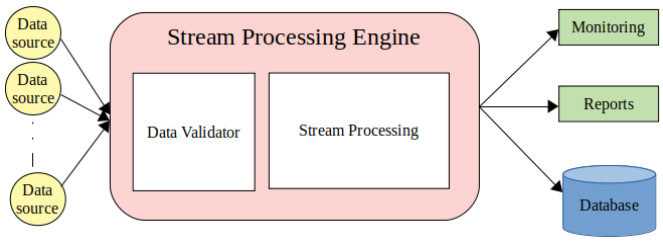

Figure 1: Proposed system architecture

In an SPE, stream processing can be either stateful or stateless. Thus, operators can be mainly classified as stateful and stateless operators. In stateless stream processing, computations are performed on the current inputs without considering any previous state. Filter, map, flatmap, etc., are some of the stateless operators. A stateless operator can be easily parallelized as it processes each record independent of other records.

In contrast, stateful stream processing requires an additional input called state for processing. Window, joins, aggregations, etc., are some of the operators that require state for computations. The stateful stream processing can be done using any of the methods such as window, machine learning, pattern recognition, and stream-stream joins. In stream-stream joins [13], an input stream is buffered as a streaming state for the future input streams. So, if the future stream match with the past stream, then the system can generate joined results. For example, stream-stream joins are used for advertising the matching ads and impressions when a user visits a website. In pattern matching over event streams [1], incoming events are compared against the complex patterns for a defined time period. In the real world, pattern matching is used to recognize suspicious financial transactions, clickstream analysis, etc. In the machine learning approach, machine learning algorithms are used to identify the models based on the datasets over a time period. For example, to monitor the trend of the incoming stream, the corresponding model is identified based on the dataset using a machine learning algorithm. Finally, window operators help us to control how to group records for further processing. This can be based on time or sessions. For example, a window operator can be used to count the number of customers that purchased some goods or services from an E-commerce website, where each customer can have multiple transactions.

Based on the type of operations that should be implemented on the data validator, data quality problems can be classified into two as stateless and stateful.

(1) Stateless processing
Following are the data quality problems and their solutions that requires stateless stream processing.
- Missing data: In this step, we search all the data fields in the incoming record that have data or not. If any of the mandatory fields like unique identifiers are missing, then the tuple is deleted. If the data is missing in the fields where the data can be fetched during actual processing is maintained by setting a status and forwards to the next step of data validator.
- Data outliers: Here, we check if the tuple contains the values that are outside the range of expected data. If it is not, then we can say that there are data outliers. The cause of data outliers is usually due to errors or attacks. Also, in this section, we cross-check the data fields to identify the manipulated data. If any data outliers are identified, then this tuple is deleted.
- Inconsistent data: In this step, we check all the fields in the incoming tuple is valid or not. That is, the data fields do not contain inconsistent values such as negative numbers, string at the place of integers, etc. All the identified tuples with inconsistent data will be deleted in this step.
- Irrelevant data: For stream processing, we do not have to maintain all the data fields as it results in memory wastage. Therefore, we filter all the irrelevant data that is not required and forwards the data records with selected fields to the next step of stream processing.

(2) Stateful processing
For the identification and removal of duplicate data from the data streams, we require stateful processing. That is, here we maintain a state, to check whether if the data record is already processed or not. If the same record arrives in the SPE multiple times, then this can be due to some attacks or failures. Duplicate records in the streaming data will affect the performance of SPE during actual stream processing. Therefore, these records will be deleted during this step.

In the data validator, we add a status field to the data record to identify the records that has problems. If the data record does not match any of the above-mentioned conditions, then the status is set as "VAL", to indicate that the data is valid. Otherwise, the pre-processing status varies based on resiliency conditions such as "MIS","OUT","INC" and "DUP". For feature selection, we consider it as valid tuple. Also, using pre-processing status, the system can be evaluated.

4 PRELIMINARY RESULTS

In the previous section, we discussed about using two stream processing engines, one for data validation and the other for actual

stream processing. Nowadays, several stream processing frameworks are available and each has its own features. Therefore, depending on the use case, we choose the framework. Since Apache Flink [9] is a true SPE based on the concept of streams and transformation and can provide low latency with high throughput, the data validator is implemented on Apache Flink. And the actual stream processing is performed using Kafka Streams as it provides a set of features such as fault tolerance, scalability, stateless and stateful processing, supports Kafka Connect to connect to other applications and databases. In Kafka, records are published in topics. From these topics, consumers can subscribe records. Here, we store pre-processed data streams on topics based on the pre-processing status. As a result, the performance of Kafka will be increased.

Apache Flink provides different data sources and sink operators. In our system, a user-defined source function is used to generate data streams from a dataset in CSV format. The dataset provided by Ausgrid[4] is used as the data source. In this dataset, the data has been collected from 300 randomly selected solar customers on a domestic tariff for the period starting from 1 July 2010 to 30 June 2013. In this dataset, each record contains 7 data fields such as *Customer ID* to store the customer data and the value ranges from 1 to 300, *Postcode* to store the location of the customer, *Generator Capacity* to record the solar panel capacity of each customer, *Consumption Category* is a two-letter code which is used to show whether meter value is consumption or generation, *Date* is in DDMMMYYYY format, *time* is of the format *0:30, ...,00:00* where meter reading is recorded in every 30 minutes and *Row Quality* shows whether the actual electricity consumption or generation is recorded by the meter or if it is an estimate value.

For data validator, the following three operators are used to handle the data quality problems,

- *filter*: Presently, the data quality problems such as missing data, inconsistent data, data outliers, and irrelevant data are identified and deleted from the incoming data records are performed using this operator.
- *window*: Stateful operations such as identification and removal of duplicate records are performed using this operator. Flink provides four types of window assigners such as tumbling windows, sliding windows, session windows and global windows. Tumbling window is used in our approach.
- *map*: Map operator is used for the data transformation. In our dataset, each record contains six data fields. Once all the stateless and stateful operations are performed on the data record, the six data fields in the record are combined to a single string separated using delimiters. This transformation is performed in order to make the sending of data to Kafka easier.

A built-in sink connector for Kafka in Flink is used for sending data to Kafka. Currently, all the validated tuples are stored in a single topic inside Kafka.

5 CONCLUSIONS AND FUTURE WORK

In the age of digitization, the existing and upcoming IoT applications increases the human comfort and efficiency. At the same time, they face wide attacking surfaces. In energy system domain, the system must protect itself against cyber attacks for the efficient generation, transmission, distribution and real-time monitoring of electric energy in the smart grid. In order to make the system resilient to situations like missing values, redundant data, data outliers, manipulated data, data failures, etc., we perform validation of data streams. Data validation is performed as a task in data pre-processing. By data pre-processing, the quality of data is increased. At the same time, data validation, data consistency and data integrity for a resilient system can be achieved. Also, it reduces unnecessary work, thereby increase the performance of the SPE during actual stream processing.

In this paper, we added basic pre-processing scenarios. In the future, it will be better to perform data analysis to study energy consumption and generation patterns. During this analysis, we can identify different possibilities, or we can identify the current trends of energy generation or consumption that may help in adding more data pre-processing or actual processing scenarios to increase the resiliency of the system. Presently, the data validator that we discussed above is based on energy management domain. As the next step, other domains will also considered and add the necessary operations in the data validator. Then, the validator will be updated as a library that can be used in different stream processing applications.

ACKNOWLEDGMENTS

This work is funded by the BMBF (Bundesministeriums für Bildung and Forschung) under grant 01IS18074A.

REFERENCES

[1] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient pattern matching over event streams. https://doi.org/10.1145/1376616.1376634

[2] Alejandro Alcalde-Barros, Diego García-Gil, Salvador García, and Francisco Herrera. 2019. DPASF: a flink library for streaming data preprocessing. *Big Data Analytics* 4, 1 (2019), 4.

[3] Malik Nadeem Anwar, Mohammad Nazir, and Khurram Mustafa. 2017. Security threats taxonomy: Smart-home perspective. In *2017 3rd International Conference on Advances in Computing, Communication & Automation (ICACCA)(Fall)*. IEEE, 1–4.

[4] Ausgrid. 2020. Ausgrid - Solar home electricity data. *URL https://www.ausgrid.com.au/Industry/Our-Research/Data-to-share/Solar-home-electricity-data* (2020).

[5] Apache Flink. 2020. Stateful Computations over Data Streams. *URL https://flink.apache.org/* (2020).

[6] Apache Software Foundation. 2017. Apache Kafka: A distributed streaming platform. *URL https://kafka.apache.org/* (2017).

[7] Salvador García, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez, and Francisco Herrera. 2016. Big data preprocessing: methods and prospects. *Big Data Analytics* 1, 1 (2016), 9.

[8] Diego Garcıa-Gil, Alejandro Alcalde-Barros, Julián Luengo, Salvador Garcıa, and Francisco Herrera. 2019. Big Data Preprocessing as the Bridge between Big Data and Smart Data: BigDaPSpark and BigDaPFlink Libraries. (2019).

[9] Vairaprakash Gurusamy, Subbu Kannan, and K Nandhini. 2017. The Real Time Big Data Processing Framework: Advantages and Limitations. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING* 5 (12 2017), 305–312. https://doi.org/10.26438/ijcse/v5i12.305312

[10] Julián Luengo. 2020. *Big Data Preprocessing: Enabling Smart Data*. Springer Nature.

[11] Weiwei Shi, Yongxin Zhu, Tian Huang, Gehao Sheng, Yong Lian, Guoxing Wang, and Yufeng Chen. 2017. An Integrated Data Preprocessing Framework Based on Apache Spark for Fault Diagnosis of Power Grid Equipment. *Journal of Signal Processing Systems* 86, 2-3 (2017), 221–236.

[12] Apache Spark. 2020. Apache Spark: Lightning-fast cluster computing. *URL http://spark. apache. org* (2020).

[13] Apache Spark. 2020. Structured Streaming Programming Guide. *URL https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html* (2020).