PROUD: PaRallel OUtlier Detection for streams

Theodoros Toliopoulos, Christos Bellas, Anastasios Gounaris, Apostolos Papadopoulos
Aristotle University of Thessaloniki, Greece
(tatoliop,chribell,gounaria,papadopo)@csd.auth.gr

ABSTRACT

We introduce PROUD, standing for PaRallel OUtlier Detection for streams, which is an extensible engine for continuous multi-parameter parallel distance-based outlier (or anomaly) detection tailored to big data streams. PROUD is built on top of Flink. It defines a simple API for data ingestion. It supports a variety of parallel techniques, including novel ones, for continuous outlier detection that can be easily configured. In addition, it graphically reports metrics of interest and stores main results into a permanent store to enable future analysis. It can be easily extended to support additional techniques. Finally, it is publicly provided in open-source.

CCS CONCEPTS

• **Information systems → Data stream mining**; • **Computing methodologies → Anomaly detection**; *Massively parallel algorithms.*

ACM Reference Format:
Theodoros Toliopoulos, Christos Bellas, Anastasios Gounaris, Apostolos Papadopoulos . 2020. PROUD: PaRallel OUtlier Detection for streams. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3318464.3384688

1 INTRODUCTION

Distance-based outliers [9] are defined as the objects that do not have more than k neighbors within a distance up to R (according to a specified distance metric). This definition has been useful and valid in several application domains [11]. In a streaming setting, instead of having a static set of objects, distance-based outlier detection is performed every time the contents of a window change due to window sliding. The size W of a window and the size S of the slide can be expressed in number of objects or in time units [15]. In a single-parameter setting, the user specifies a single pair of R and k values, while W and S are fixed. In multi-parameter queries, the user can ask for multiple combinations of R, k, W and S values.

Although there are tools and implementations of the main state-of-the-art algorithms for non-parallel streaming outlier detection [8, 15], there is no open-source engine for massively parallel platforms. This work fills this gap. More specifically, we introduce PROUD, standing for PaRallel OUtlier Detection for streams, which is an extensible engine built on top of Flink that supports a variety of parallel techniques, including novel ones, for continuous distance-based outlier-detection. It comes with a user friendly interface for configuration, it graphically reports metrics of interest and stores main results into a permanent store to enable future analysis. Additionally, PROUD can be easily extended to support additional partitioning and outlier detection techniques. It can be downloaded in open source from https://github.com/tatoliop/PROUD-PaRallel-OUtlier-Detection-for-streams.

2 SYSTEM DESCRIPTION

PROUD's core functionality regarding continuous anomaly detection is implemented in Flink [1], but the engine also leverages two additional state-of-the-art open source frameworks for big data processing, namely Kafka [2] and InfluxDB [6] (see Fig. 1). The former is used for handling the input streams, while InfluxDB stores the results of the outlier detection task and performance-related metadata. In a nutshell, PROUD is a high-throughput distributed outlier detection engine that supports distance-based outlier detection algorithms and different partitioning techniques, while providing custom metrics for each individual task of the engine. Below, we briefly describe its main functionality, the algorithms already implemented and the extensibility points.

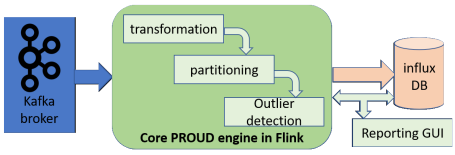


Figure 1: PROUD's architecture

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3384688

2.1 Main Functionality

There are four main processing phases after reading input data from a stream handler, such as Kafka brokers:

1. **Data Transformation.** Input data are continuously read from a stream handling platform and transformed into the reference data model of PROUD.
2. **Data Partitioning.** A user-defined partitioning technique is applied to split the data points into separate logical partitions according to their values, while arranging them into sliding time windows based on their timestamps.
3. **Outlier Detection.** For each logical partition, the selected distance-based outlier detection algorithm runs to process the local data points and output the resulting outliers; the algorithms are implemented in such a manner that neither false positives nor false negatives are produced despite processing each logical partition independently in parallel.
4. **Storage and Reporting.** The results from all of the partitions are transferred to a data sink for storage, reporting and/or further processing

More specifically, in the initial stage, the engine reads the input stream and transforms it into a reference data model containing three items: (i) the value read in the form of a list of real numbers, (ii) the timestamp of arrival and (iii) an id. Due to the distributed nature of Flink, the data ingestion and transformation is performed by multiple partitions in parallel to attain scalability. The reference data model is independent of the outlier detection algorithm being used and can be extended with additional auxiliary fields according to each algorithm's requirements.

Following the data transformation, the partitioning technique is responsible for assigning each data point to a specific logical partition for the actual outlier detection processing. Currently, there are two main options offered to users: (i) a grid-based one that assumes a euclidean space split into grid cells according to the parameters provided by the user; and (ii) a tree-based ones that can be applied to arbitrary metric spaces and relies on the provision of a sample to construct the partitioning tree. The implementation rationale is the same as in [14]. Both partitioning techniques allow for limited overlaps between the datasets in the logical partitions with a view to enabling each partition to be processed independently and produce exact results, as explained in [14] and other similar works. Both these options are value-based ones, i.e., the partition assignment depends on the list of values so that, in principle, points close to each other are assigned to the same partition. PROUD includes also a naive processing partitioning and processing approach for baseline purposes, according to which each point is replicated to all partitions.

After each point is assigned to a partition, it is further transformed into a different data class, based on the outlier detection algorithm. Also, depending on the time of their arrival, the points are arranged into a sliding time-based window. The window size and slide size are also provided by the user that starts the outlier detection job. Next the outlier detection part comes into the foreground as soon as the window slides. The detection algorithm that is chosen by the user works independently on each partition's data. There is no communication between each partition and the algorithm needs only the partition's own data points in order to process them correctly and output the exact local outliers. Each outlier detection algorithm has a distinct state class that is used to store data points and their meta-data for each window. Whenever the window slides, the new data points are inserted into the state to be processed by the algorithm and the expired data points are removed from the state. We briefly mention the algorithms currently implemented in the following section.

The final step of the engine is to output the resulting outliers of each logical partition into a data sink. The local outliers of each partition are grouped together for the complete result. The outlier detection engine supports both single-query and multi-query distance-based outlier detection algorithms. If, during the initialization phase, the user has chosen more than one queries, the resulting outliers of each logical partition are also grouped by the query they refer to. Finally, Flink supports custom performance metrics at multiple levels of granularity from the job itself to a specific task. The PROUD engine also encapsulates three custom metrics for an outlier detection task running on each logical partition. These metrics measure the number of window slides, the total processing time and the average processing time for each logical partition; they are reported on the Flink's UI and can also be forwarded to a selected reporter/storage.

2.2 Algorithms Supported

At the moment of writing, PROUD encapsulates a re-engineering of the state-of-the art set of algorithms, and two novel techniques, as specified below.

Single-parameter techniques. These fall into three categories: (i) PROUD provides a re-engineered version of all algorithms presented in [13], which is the first work on massively parallel streaming distance-based outlier detection. (ii) According to the experimental survey, in [15], the best performing technique was the one in [10] followed by [7]. The parallel flavor of the former, which leverages the notion of micro-clusters, is already described in [13]; PROUD also encapsulates a parallel flavor of the notion of window slicing from [7], an initial version of which is mentioned in [14]. (iii) Recently, [16] has been proposed for non-parallel

distance-based oulier detection; PROUD includes a novel advanced algorithm that combines the key elements of [10] and [16] in a parallel manner. Briefly, the novel technique, before dissolving a micro-cluster due to point lifetime expiration, which is an expensive process, checks whether a similar cluster could be created due to the arrival of new objects. **Multi-parameter techniques.** These techniques consist of the re-engineering of the set of solutions described in [12] to fit into the PROUD engine. Details are omitted due to lack of space.

2.3 Extensibility Points

The PROUD engine is designed in order to be easily extended in many ways ranging from the data transfer frameworks to the outlier detection algorithms, partitioning techniques and custom metrics.

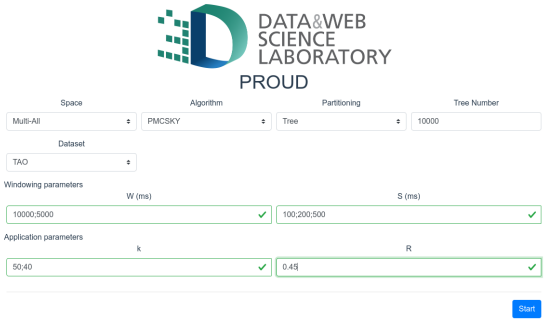
(1) The core of the engine are the distance-based outlier detection algorithms. Section 2.2 provides information about the implemented algorithms that are already available for usage. Inserting new algorithms that work alongside the already existing ones is easy through creating a class that incorporates the algorithm. This new class extends Flink's base `ProcessWindowFunction` class, and there is no need for further dependencies between algorithms and partitioning techniques. Furthermore each algorithm to be implemented can have its own state, internal variables and data model. If a specific data model is needed, e.g., for custom meta-data, the `Data_basis` class can be extended without affecting the input stream or the partitioning techniques.

(2) New partitioning techniques and custom metrics can also be implemented. The new partitioning techniques are independent classes that do not need to extend any Flink or engine-specific classes. Their only restriction is to ingest the reference data model and output the data point along with the partitions that it needs to be transferred to, i.e., to conform to a simple API. Furthermore, due to Flink's features, one can implement any number of different custom metrics. E.g., a new custom metric can be inserted during the partitioning task to measure the replication rate.

(3) Finally, one of the extensibility features that the engine provides through the Flink framework are the data sources and sinks. The Flink community provides libraries for many different data transfer and storage frameworks and changing the source or the sink that the engine uses is easy. Even in the cases where an official library is not available, the user can extend the `RichParallelSourceFunction` class in order to ingest data in a distributed manner from a custom source. This also holds true for data sinks.

3 DEMO DESCRIPTION

During the demonstration of the engine, a complete workflow of outlier detection setup, running and reporting will




```
> select * from outliers limit 5
name: outliers
time                Outliers R    S   W     k
----                -------- -    -   -     -
1578572736000000000 54       0.35 500 10000 50
1578572736500000000 183      0.35 500 10000 50
1578572737000000000 321      0.35 500 10000 50
1578572737500000000 439      0.35 500 10000 50
1578572738000000000 552      0.35 500 10000 50
>
```

Figure 2: Sceenshots of the UI (top) and InfluxDB outlier info (bottom)

be provided. The setup is performed in 4 steps. Step 1: to emulate a real stream, a custom generator that works in Flink is implemented and supports the generation of two different streams based on real-world datasets: Stock and TAO. The datasets are generated by emulating a bell distribution within the same range of values as in the original datasets. The rate at which data points are created is approximately 1 data point per millisecond. The generated stream is transferred through Kafka to the main PROUD engine. Step 2: Through a user interface implemented in Javascript, the user can choose the parameter space (e.g. single-parameter or multi-parameter). Based on the previous results, a list of appropriate algorithms are present for the user to choose. Step 3: The user configures the partitioning technique. Step 4: The user inputs the parameters for the job (e.g. window size and queries). Figure 2(top) shows the UI.

While the job is processing the data points, three custom measurements are reported regarding outlier detection: the number of window slides, the total processing time of the outlier detection task and the average processing time for each partition. These measurements along with Flink's default metrics, such as the heap memory used per taskmanager and the CPU time used by the JVM, are reported both to Flink's UI and stored to InfluxDB as time series. For better visualization, the open-source analytics and monitoring framework Grafana [5] is chosen. In the demonstration, Grafana is setup to present the latest results from the outlier detection algorithm as well as the average processing time of the task (see Figure 3). Another part of the demonstration includes the storage of the engine's results (see Figure 2(bottom)), which allows for arbitrary further analysis.

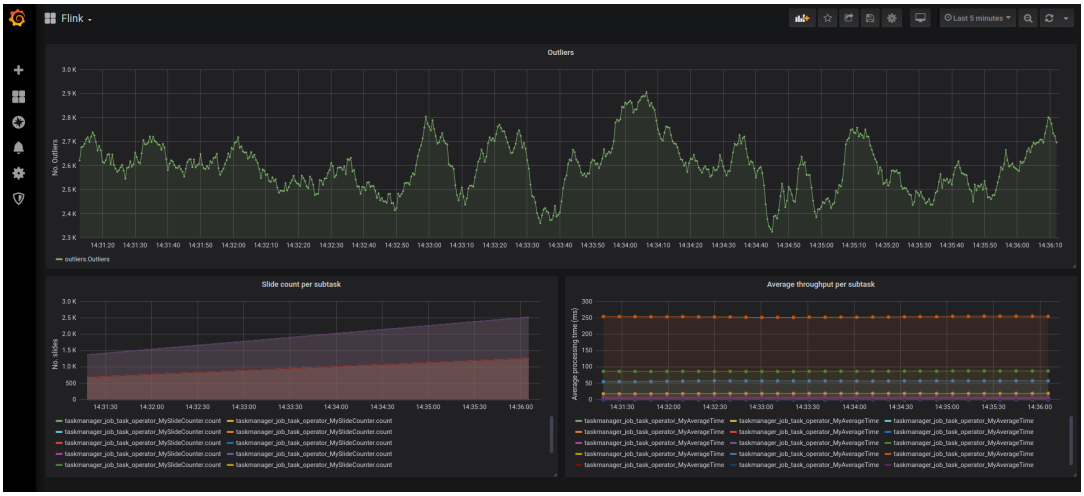


Figure 3: Grafana scheenshot showing the number of outliers (top), the number of slides per partition (bottom left) and the processing time per partition (bottom right).

Deploying PROUD and reproducing the demo. The complete demo platform and scenario is available for reproduction through Docker containers and a Docker-compose file [4]. The Docker-compose file sets up all framework components as well as the dependencies between them. The user, through a bash script, with just one command can directly build the source code, build the Flink image with the compiled jar file and start up all services including Flink, InfluxDB, Grafana, Kafka and Zookeeper[3], which is required by Kafka. Afterwords, another command connects directly to the Flink container in order to start the local server with the user interface. After the outlier detection job has started through the UI the results become visible from Grafana's UI. As a final note, the Docker-compose file is set to run on a single multi-core machine for the purposes of the demonstration; minimal changes are needed to transfer it to a cluster using either the Docker Swarm service or Kubernetes.

Summary. This work presents PROUD, an extensible framework for parallel streaming distance-based outlier detection in Flink. PROUD already includes novel techniques that have not appeared in the literature yet along with published state-of-the-art. It can be extended in several ways; here we mention three of them: (i) insert support for other types of data than numeric, for which distance metrics exist, e.g., strings; (ii) make PROUD partitioning adaptive; and (iii) insert techniques tailored for high dimensions and support also density-based and/or approximate outlier detection.

Acknowledgement. This research work has been supported by the European Commission under the Horizon 2020 Programme, through funding of the RAINBOW project (Grant 871403).

REFERENCES

[1] [n.d.]. Apache Flink. https://flink.apache.org/.
[2] [n.d.]. Apache Kafka. https://kafka.apache.org/.
[3] [n.d.]. Apache Zookeeper. https://zookeeper.apache.org/.
[4] [n.d.]. Docker. https://www.docker.com/.
[5] [n.d.]. Grafana. https://grafana.com/.
[6] [n.d.]. InfluxDB. https://www.influxdata.com/.
[7] Lei Cao, Di Yang, Qingyang Wang, Yanwei Yu, Jiayuan Wang, and Elke A Rundensteiner. 2014. Scalable distance-based outlier detection over high-volume data streams. In *ICDE*. 76–87.
[8] Dimitrios Georgiadis, Maria Kontaki, Anastasios Gounaris, Apostolos N. Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2013. Continuous outlier detection in data streams: an extensible framework and state-of-the-art algorithms. In *SIGMOD*. 1061–1064.
[9] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. 2000. Distance-based Outliers: Algorithms and Applications. *The VLDB Journal* 8, 3-4 (2000).
[10] Maria Kontaki, Anastasios Gounaris, Apostolos N Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2016. Efficient and flexible algorithms for monitoring distance-based outliers over data streams. *Information systems* 55 (2016), 37–53.
[11] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. 2006. Online Outlier Detection in Sensor Data Using Non-Parametric Models.. In *VLDB*. 187–198.
[12] Theodoros Toliopoulos and Anastasios Gounaris. 2019. Multi-parameter streaming outlier detection. In *WI*. 208–216.
[13] Theodoros Toliopoulos, Anastasios Gounaris, Kostas Tsichlas, Apostolos Papadopoulos, and Sandra Sampaio. 2018. Parallel Continuous Outlier Mining in Streaming Data. In *DSAA*. IEEE, 227–236.
[14] Theodoros Toliopoulos, Anastasios Gounaris, Kostas Tsichlas, Apostolos Papadopoulos, and Sandra Sampaio. 2019. Continuous Outlier Mining of Streaming Data in Flink. *CoRR* abs/1902.07901 (2019).
[15] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2016. Distance-based outlier detection in data streams. *PVLDB* 9, 12 (2016), 1089–1100.
[16] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. 2019. NETS: Extremely Fast Outlier Detection from a Data Stream via Set-Based Processing. *PVLDB* 12, 11 (2019), 1303–1315.