Grand Challenge: Automatic Anomaly Detection over Sliding Windows

Tarek Zaarour
Insight Centre for Data Analytics, NUI, Galway
tarek.zaarour@insight-centre.org

Niki Pavlopoulou
Insight Centre for Data Analytics, NUI, Galway
niki.pavlopoulou@insight-centre.org

Souleiman Hasan
Lero - The Irish Software Research Centre, NUI, Galway
souleiman.hasan@lero.ie

Umair ul Hassan
Insight Centre for Data Analytics, NUI, Galway
umair.ulhassan@insight-centre.org

Edward Curry
Insight Centre for Data Analytics, NUI, Galway
edward.curry@insight-centre.org

ABSTRACT

With the advances in the Internet of Things and rapid generation of vast amounts of data, there is an ever growing need for leveraging and evaluating event-based systems as a basis for building real-time data analytics applications. The ability to detect, analyze, and respond to abnormal patterns of events in a timely manner is as challenging as it is important. For instance, distributed processing environment might affect the required order of events, time-consuming computations might fail to scale, or delays of alarms might lead to unpredicted system behavior. The ACM DEBS Grand Challenge 2017 focuses on real-time anomaly detection for manufacturing equipments based on the observation of a stream of measurements generated by embedded digital and analogue sensors. In this paper, we present our solution to the challenge leveraging the Apache Flink stream processing framework and anomaly ordering based on sliding windows, and evaluate the performance in terms of event latency and throughput.

CCS CONCEPTS

• **Computer systems organization → Distributed architectures**; • **Software and its engineering → Publish-subscribe / event-based architectures**; *Message oriented middleware*; • **Theory of computation →** *Parallel computing models*;

KEYWORDS

event-based processing, anomaly detection, event ordering, K-means, Markov chain model

ACM Reference format:
Tarek Zaarour, Niki Pavlopoulou, Souleiman Hasan, Umair ul Hassan, and Edward Curry. 2017. Grand Challenge: Automatic Anomaly Detection over Sliding Windows. In *Proceedings of DEBS '17, Barcelona, Spain, June 19-23, 2017,* 5 pages.
https://doi.org/10.1145/3093742.3095105

1 INTRODUCTION

Lately the Internet of Things has gained too much attention both from academia and industry. This contributes to the ever growing real-time data analytics applications, like anomaly detection. Real-time anomaly detection in event-based systems is used in many fields, like cyber-attack detection, environmental and traffic monitoring energy, and industry [10]. In a manufacturing environment, the ability to detect errors, problems or defections using data for equipment's sensors and be proactive so that appropriate actions can be taken. These actions allow for the detection and response to issues as they happen and sustain the normal behavior and satisfactory performance of the manufacturing process.

A manual approach to diagnosis of manufacturing anomalies is laborious and time-consuming. Therefore, a real-time computational analysis of a stream of events is crucial, yet challenging. Identifying the type and source of a defect could sometimes be infeasible, because the analysis of the data might be inaccurate or the response could be late for actions to be taken. Producing false alarms could result in an unexpected equipment behavior and unnecessary actions to be taken. The speed of the analysis should also be appropriate for actions to happen on time in case of an anomaly. Often delays might be introduced, because the analysis might not scale well, or the events might not come in order; therefore, the system can try to wait for a period of time and then reorder them for the final output.

This year's challenge [5] focuses on a real-time anomaly detection solution for streams of events that derive from sensors of fixed or dynamic machines. These events contain values of observed properties, such as temperature, pressure, etc., that are associated with specific timestamps. Our solution reads these events and temporarily stores them in window frames, where they are clustered with a mini-batch K-means algorithm. The state transitions between clusters are made for training a Markov chain model. The probability of a sequence of transitions within these window frames define the detection of an anomaly if it is lower than a given threshold.

For example, a molding machine may have sensors collecting indicators of its functions such as its temperature. It is normal for temperature to fluctuate within some range. However, if the temperature readings indicate a series with a very high and very low value within a window of continuous readings then a real-time analytics platform could detect that there is a malfunction that started in a specific timestamp within this range.

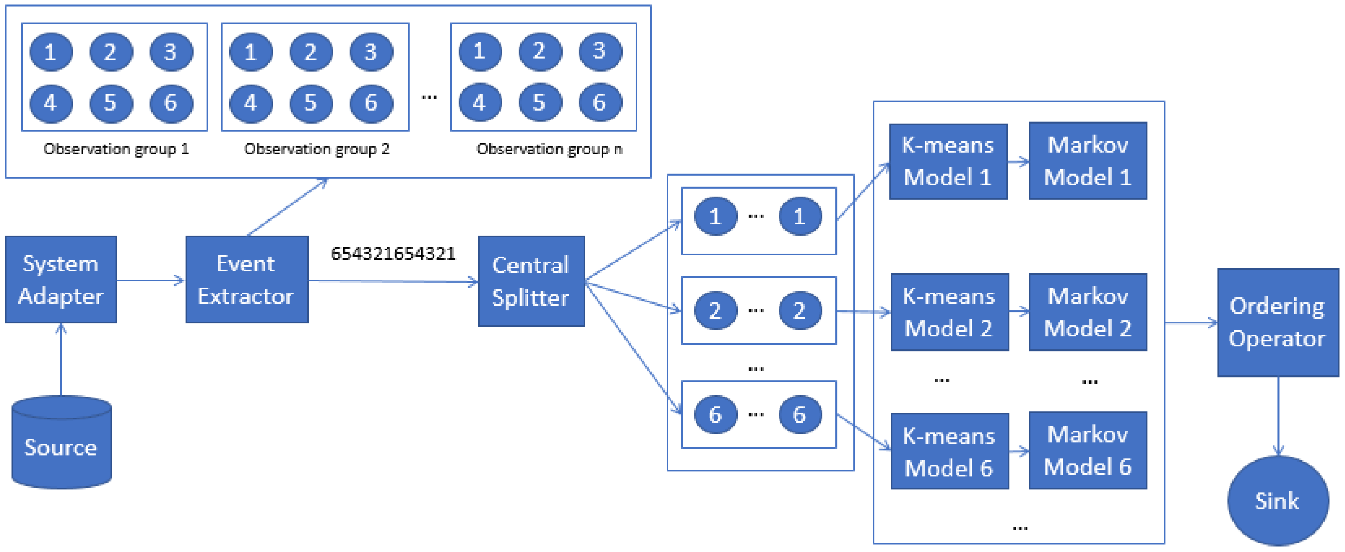

Figure 1: Solution architecture.

The remainder of this paper is structured as follows: Section 2 outlines the challenges associated with the problem and Section 3 presents our proposed approach. Section 4 discusses the experimental results and Section 5 concludes the paper.

2 CHALLENGES AND METHODOLOGY

The graph-structured data and event grouping strategy of the DEBS Grand Challenge 2017 poses various challenges. The first challenge is splitting graph-structured data points into separate timestamped events. The second challenge revolves around achieving high scalability and efficient allocation of computational tasks which requires parallelizing two operators that implement K-means clustering and a Markov Chain model respectively. Whereas, the third challenge is maintaining the order of events in a parallel processing environment.

- **Consistent Splitting:** The input stream consists of data points that represent what is referred to as observation groups. A data point or observation group read by the system at a certain point in time contains up to 120 measurements that are all associated with a single timestamp. The different measurements need to be windowed and processed independently. This requires consistent splitting while assuring that each generated event is bound to the timestamp of the observation group it belongs to.
- **Scalability and Task Allocation:** Achieving parallelization can be approached in two ways. 1) Splitting each independent windowed computation into multiple tasks. However, the size of the window poses a limitation because splitting, processing, and merging a relatively low workload can be more time consuming than serial processing. 2) Splitting the input stream into logical keyed streams based on the different types of measurements allows for windowed computations to be performed in parallel. This is a more efficient way, thus in our solution, an operator is split into several parallel instances and each instance performs an independent windowed computation.
- **Order of Anomalies:** Parallel processing might affect the order of processed events. To resolve this issue, we added an ordering task that buffers event notifications or anomalies over a sliding time window and sorts them based on their timestamp. Yet at a high level of parallelism, an event with an earlier timestamp than the events in the current window might appear in the next window slide; hence, we only forward the event holding the earliest timestamp in the current window.

3 ANOMALY DETECTION SOLUTION

Our solution follows a parallel stream processing paradigm as we have data coming at high injection rates requiring continuous time-consuming analytics computations. We first give an overview of our architecture, where we also go through operator implementation details, then we refer to the technology stack that is used for implementation.

3.1 Overview

The proposed solution shown in Figure 1 comprises six logical operators that apply complementary transformations on the incoming event stream resulting in a continuous workflow that follows the paradigm of distributed stateful stream processing. The following are the key components:

- **System Adapter:** As described in the challenge, a system adapter is in place to: (1) register the system consumer and producer with input and output queues; (2) make sure that the benchmarking execution environment that hosts all components is initialized and ready; and (3) send a message on the command queue to inform the benchmarking platform that it is ready to process a task or react to a termination message.

- **Event Extractor:** The event extractor is in charge of reading input events, extracting relevant pieces of information and its marshalling into an appropriate format. Each data point read by this operator represents an observation group that is associated with a specific timestamp and contains several observed properties. Moreover, the challenge provides a metadata file that contains the number of clusters for each stateful observed property generated by each individual machine. The event extractor: 1) splits each observation group into individual observed properties; 2) binds each observed property with the timestamp of the observation group it belongs to; and 3) adds relevant metadata before pushing the newly generated event downstream.
- **Central Splitter:** Every data point in the stream consists of various sensor readings that can be windowed and processed independently. Consequently, in order to scale the system to support data coming from more machines, this component partitions the stream logically based on the different observed properties. Therefore, allowing downstream windowed computations to be performed in parallel.
- **K-means Clustering Model Operator:** To leverage the large variability in observed properties, multiple instances of this operator can run concurrently on chunks of events assigned to overlapping sliding windows. Specifically, it implements K-means clustering by realizing the following functionality for each window of an observed property: (1) finding cluster centers; (2) assigning events to the closest cluster; (3) iterating until convergence or until it reaches a maximum number of allowed iterations; and (4) continuously producing sequences that represent the cluster membership of each event that belongs to the current window.
- **Markov Model Operator:** This component is in charge of training the Markov chain model where each point in the previously generated sequence represents a separate state. The main goal of performing this computation is to detect if the assignment of a point to a certain cluster is unlikely to happen, or in other words if the point is an outlier. For this to be detected the number of transitions is calculated between all states within each window and the combined transition probability of the last N transitions is calculated and checked against a given threshold T_d. In case of a deviation, an anomaly is produced and bound to the timestamp of the event that caused it.
- **Ordering Operator:** The stream is split into logical keyed streams that can be processed independently by assigning multiple tasks. Thus, allowing the distribution of both the K-means and Markov Model Operators across parallel workers can introduce out-of-order anomalies. This operator is in place to reorder the generated event notifications based on their timestamps. As shown in Figure 2, the ordering operator buffers all anomalies it receives over a sliding time window, it then sorts all assigned anomalies on each window and forwards the anomalies with the earliest timestamps.

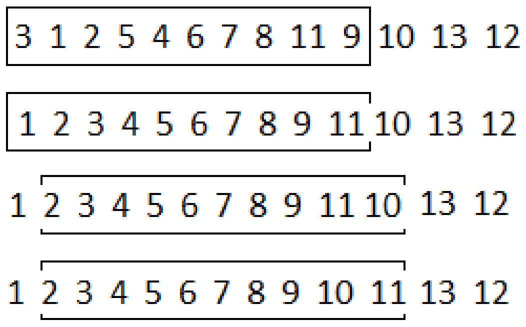

Figure 2: Sorting anomalies over sliding windows. The numbers represent the different timestamps and the blocks represent each window that slides by one timestamp each time.

3.2 Technology stack overview

We examined Apache Storm [12], Apache Spark [14] and Apache Flink [1] as our stream processing framework. We decided not to use Storm, because it did not support batching capabilities in case we needed them. Between Spark and Flink, we decided to use Flink, even though Spark is more mature, because it processes tasks with lower latency due to its pipeline execution. We also used Apache Jena [7], which is a framework for building Semantic Web and Linked Data applications, as an RDF processing framework. Docker [8], a platform for building, shipping and running distributed applications, was used for containerizing our software, since the evaluation platform consists of several containers. RabbitMQ [9], a message queueing system, is also used on the evaluation platform as a message bus. An overview of the different technologies is shown in Figure 3.

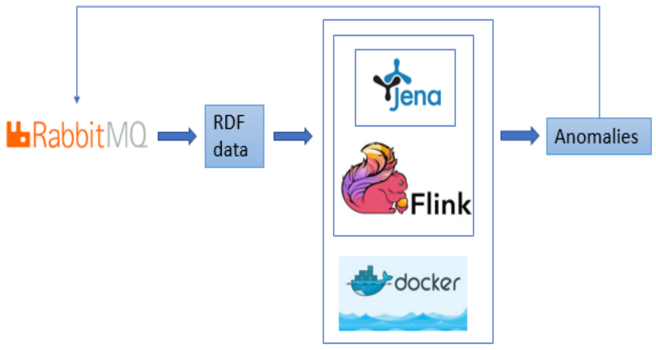

Figure 3: Technology stack.

4 EVALUATION

As part of the challenge, and in order to evaluate the performance of our system in comparison with solutions submitted by all participants, each team submitted their system to an automated evaluation

Table 1: Event throughput in Mbps

Number of messages	1000	10,000	20,000
Throughput(Mbps)	20.235	30.341	30.805

cluster comprising three 8-core nodes. Solutions were ranked according to system latency and throughput. System latency measures the average time difference between submitting an event notification to the input queue and writing the resulting anomaly to the output stream. The experiments show how our distributed system performs at relatively low and high injection rates. We compare the resulting latency and throughput to that of the sequential version, where parallelism is set to one, thus no additional event ordering phase is needed.

Figure 4 shows the mean latency results for around 30 experimental runs. As requested in the challenge description, we give the system latency as a function of different injection rates. In Figure 5 we show how system performance varies as we change the number of machines being monitored. As input stream, we did most of our experiments on 10,000 and 20,000 messages where a single message read at a certain point in time contains up to 120 different measurements (events). Table 1 depicts the performance results in terms of average throughput in Mbps as opposed to data volume.

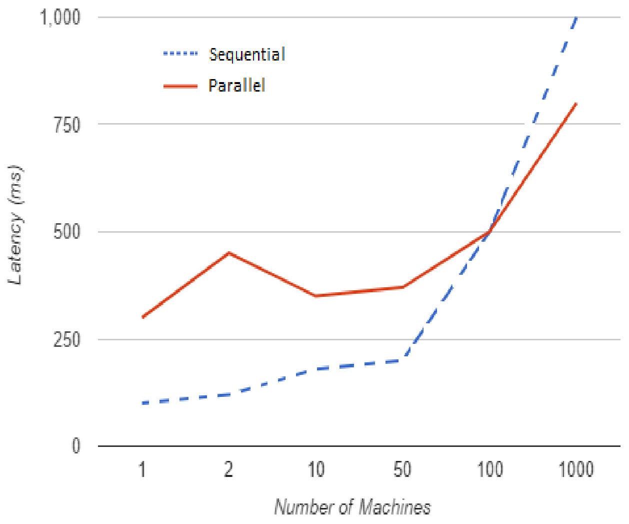

Figure 5: System latency as a function of number of machines.

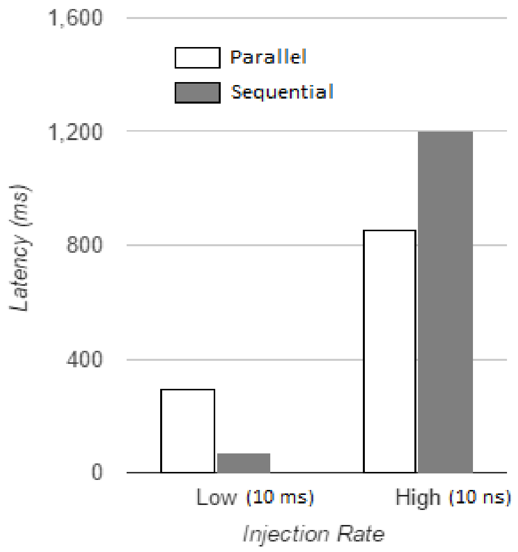

Figure 4: System latency as a function of injection rate.

Sequential Approach: Figure 4 shows that our sequential approach performs better at a low injection rate i.e., 10 ms. However, injection rate gets higher as more machines join, because each machine ingests data at the same rate concurrently. Therefore, the mean latency increases exponentially as the number of machines goes near 1000.

Parallelized Approach: The ordering operator increased the mean latency at low injection rates to 359 ms. As more machines join, injection rate increases causing an increase in overall system latency. Hence, the amount of latency eliminated by applying distributed processing becomes greater than the latency introduced by the sliding window used for sorting anomalies. This causes a noticeable decrease in overall latency from 1.2 seconds for the sequential approach to 850 ms for the distributed approach.

5 RELATED WORK

Researchers have previously tackled anomaly detection over event streams. Ye et al. [13] trained a Markov chain model from a normal sequence of cyber activity and compared it to a new sequence. The higher the probability of a new sequence, the more likely is the sequence of states derived from normal activity in the network. Goldberg et al. [4] used the augmented Markov model (AMM) for fault detection in the behavior of a group of mobile robots. Hahsler et al. [6] and Dunham et al. [2] extended the AMM [4] and combined it with the nearest neighbor algorithm for clustering to examine the spatiotemporal nature of continuously arriving data and the prediction of rare events. Fernandez et al. [3] presented a scalable stateful stream processing system for smart grids based on the SEEP stream processing platform. The system implemented parallel processing, stateful operators, and semantic load-shedding to reduce events to process downstream and periodic state-check pointing. Saleh et al. [11] addressed partitioning for scalable complex event processing over data streams. They presented a cost-based approach for partitioning dataflow graphs and patterns in complex event processing (CEP) queries. Our approach is based on the Markov chain model that was used in [2, 4, 6, 13]. The work in [2, 6] is very similar to this year's challenge, although nearest neighbor algorithm is used for clustering instead of K-means. [3, 11] gave us an idea of how to better approach windowing solution in streaming processes.

6 CONCLUSIONS

In this work, we describe a scalable solution for the DEBS Challenge 2017. The goal is to detect anomalies from RDF data generated by multiple sensors embedded in different machines. Our approach uses a mini-batch version of K-means and Markov chain models by leveraging the streaming capabilities of Apache Flink. It also reorders the anomalies according to their timestamp by using sliding windows. Our results showed how the parallel version coped with high injection rates and performed better than the sequential one in terms of latency and throughput. However, the latter performed better at relatively low injection rates, because there was no need for buffering and sorting out-of-order anomalies. The limitations we faced inspire several directions for future work. We plan to investigate better implementations for sorting out-of-order events in stream processing and to look further into logical stream partitioning in favor of having better task distribution and resource allocation.

ACKNOWLEDGMENTS

This work was supported in part by Science Foundation Ireland grants 13/RC/2094 and SFI/12/RC/2289 and in part with the European Union's Horizon 2020 research programmes Transforming Transport (TT) grant No 731932 and Big Data Value ecosystem (BDVe) grant No 732630.

REFERENCES

[1] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).

[2] Margaret H Dunham, Yu Meng, and Jie Huang. 2004. Extensible markov model. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on.* IEEE, 371–374.

[3] Raul Castro Fernandez, Matthias Weidlich, Peter Pietzuch, and Avigdor Gal. 2014. Scalable stateful stream processing for smart grids. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems.* ACM, 276–281.

[4] Dani Goldberg and Maja J Matarić. 1999. Coordinating mobile robot group behavior using a model of interaction dynamics. In *Proceedings of the third annual conference on Autonomous Agents.* ACM, 100–107.

[5] Vincenzo Gulisano, Zbigniew Jerzak, Roman Katerinenko, Martin Strohbach, and Holger Ziekow. 2017. The DEBS 2017 grand challenge. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS '17, Barcelona, Spain, June 19 - 23, 2017.*

[6] Michael Hahsler, Margaret H Dunham, et al. 2010. remm: Extensible markov model for data stream clustering in r. *Journal of Statistical Software* 35, 5 (2010), 1–31.

[7] Apache Jena. 2015. A free and open source Java framework for building Semantic Web and Linked Data applications. *Available online: jena. apache. org/(accessed on 28 April 2015)* (2015).

[8] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.

[9] Apache RabbitMQ. 2017. RabbitMQ - Messaging that just works. *URL: https://www.rabbitmq.com/* (2017).

[10] Sebnem Rusitschka and Edward Curry. 2016. Big Data in the Energy and Transport Sectors. In *New Horizons for a Data-Driven Economy.* Springer, 225–244.

[11] Omran Saleh, Heiko Betz, and Kai-Uwe Sattler. 2015. Partitioning for scalable complex event processing on data streams. In *New Trends in Database and Information Systems II.* Springer, 185–197.

[12] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. 2014. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data.* ACM, 147–156.

[13] Nong Ye, Yebin Zhang, and Connie M Borror. 2004. Robustness of the Markov-chain model for cyber-attack detection. *IEEE Transactions on Reliability* 53, 1 (2004), 116–123.

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. (2010).