EasyFlinkCEP: Big Event Data Analytics for Everyone

Nikos Giatrakos
Athena Research Center
Technical University of Crete
ngiatrakos@athenarc.gr

Eleni Kougioumtzi
Athens University of
Economics and Business
elkoum15@gmail.com

Antonios Kontaxakis
Athena Research Center
Technical University of Crete
akontaxakis@athenarc.gr

Antonios Deligiannakis
Athena Research Center
Technical University of Crete
adeli@athenarc.gr

Yannis Kotidis
Athens University of
Economics and Business
kotidis@aueb.gr

ABSTRACT

FlinkCEP is the Complex Event Processing (CEP) API of the Flink Big Data platform. The high expressive power of the language of FlinkCEP comes at the cost of cumbersome parameterization of the queried patterns, acting as a barrier for FlinkCEP's adoption. Moreover, properly configuring a FlinkCEP program to run over a computer cluster requires advanced skills on modern hardware administration which non-expert programmers do not possess. In this work (i) we build a novel, logical CEP operator that receives CEP pattern queries in the form of extended regular expressions and seamlessly re-writes them to FlinkCEP programs, (ii) we build a CEP Optimizer that automatically decides good job configurations for these FlinkCEP programs. We also present an experimental evaluation which demonstrates the significant benefits of our approach.

CCS CONCEPTS

• **Information systems** → **Query optimization**; • **Computer systems organization** → **Distributed architectures**.

KEYWORDS

Big Data; FlinkCEP; Complex Event Processing; Optimizer

ACM Reference Format:
Nikos Giatrakos, Eleni Kougioumtzi, Antonios Kontaxakis, Antonios Deligiannakis, and Yannis Kotidis. 2021. EasyFlinkCEP: Big Event Data Analytics for Everyone. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3459637.3482094

1 INTRODUCTION

Complex Event Processing (CEP) enables analysts to express business rules as patterns and directly query incoming streams of data to detect these patterns [11, 12, 14]. For instance, in the maritime domain, CEP is utilized to detect illegal activities at sea based on vessel position streams [4, 5]. Or, in the financial sector, sequences of increasing, decreasing trend events on stock trade time series express price swing patterns [17, 18].

In the CEP context, rapid streams of data are processed online to timely report detected Complex business Events (CEs). CEs consist of simpler events in the form of singleton, looping, sequencing or other patterns with the use of quantifiers. CEP queries additionally include (a) contiguity conditions (a.k.a. selection strategies [12]) to determine whether we are allowed to ignore events of the incoming streams that are irrelevant to the monitored pattern or not, (b) event consumption policies, to denote how many matches an event may be assigned to, and (c) windowing operations [12, 14, 19].

To handle the volume and velocity of Big Streaming Data, Big Data platforms such as Apache Flink [9] have emerged. They focus on scaling-out the computation to a number of machines in a computer cluster/cloud, working in parallel on portions of the streams, to speed up continuous analytic outcomes. Flink includes a native API, namely FlinkCEP [10], for CEP analytics. FlinkCEP provides a CEP language of high expressive power [7, 15] and also allows for parallel processing to ensure rapid delivery of CEP analytics. Nonetheless, there are certain barriers in FlinkCEP's adoption.

First, FlinkCEP requires business analysts, who know how to define business rules (i.e., the CEs to be monitored) but are usually non-expert programmers, to write code in Scala or Java. Second, pattern expression and parameterization involves cumbersome notations which makes the whole code writing process error-prone [12]. Third, submitting a FlinkCEP program to get executed as a job on a Flink cluster requires advanced skills on properly configuring and optimizing the usage of the available hardware resources which, more often than not, business event analysts do not possess.

To overcome the barrier posed by coding directly in FlinkCEP, we build a novel, logical CEP operator that receives as input CEP pattern queries in the form of extended regular expressions [1, 2, 6, 15, 22, 25] and seamlessly re-writes them to FlinkCEP programs. To offload domain experts from computer cluster administration decisions, we build a CEP Optimizer that automatically decides good configurations for executing the transformed FlinkCEP code on a computer cluster. We implement a prototype, namely EasyFlinkCEP, in the open-source platform of the INFORE project [1] and present experiments demonstrating the benefits of our approach.

This work has received funding from the EU Horizon 2020 research and innovation program INFORE under grant agreement No 825070.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482094

[1] https://infore-project.eu/, https://bitbucket.org/infore_research_project/, https://github.com/eleniKougiou/Flink-cep-automation

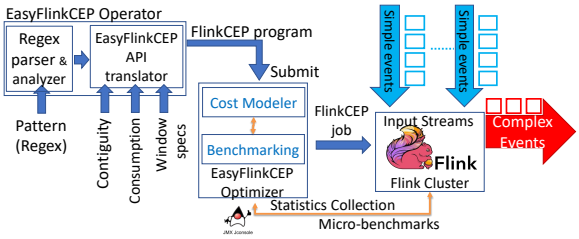


Figure 1: EasyFlinkCEP Operator to FlinkCEP Program

Listing 1: Java code in FlinkCEP for detecting *ab+* patterns

```
1   // ------ Create pattern "a b+" ------
2   Pattern<Event, ?> pattern = Pattern.<Event>begin("start", skipStrategy).where(new
    SimpleCondition<>() {
3       @Override
4       public boolean filter (Event value){
5           return value.getName().startsWith("a");}
6   }).followedBy("next").where(new SimpleCondition<>() { //followedBy for Relaxed Contiguity
7       @Override
8       public boolean filter (Event value){
9           return value.getName().startsWith("b");}
10  }).oneOrMore().consecutive(); // oneOrMore().consecutive() for b+
11  PatternStream<Event> patternStream = CEP.pattern(input, pattern);
```

2 THE EasyFlinkCEP OPERATOR

The EasyFlinkCEP Operator [29] is a logical operator allowing the business event analyst to be oblivious of FlinkCEP's language and, instead of coding, to focus on rapidly defining CEP queries. This is done by simply specifying (P1) a regular expression (Regex) denoting the CEs to be detected, (P2) the contiguity condition, (P3) the consumption policy and (P4) window specifications [19, 24].

FlinkCEP [10] supports the following contiguity conditions (i) Strict Contiguity: where matching events appear strictly one after the other in the incoming stream(s), (ii) Relaxed Contiguity: where non-matching events appearing in-between the matching ones are ignored, and (iii) Non-Deterministic Relaxed Contiguity: that allows non-deterministic actions between matching events. For event consumption, basic options include: (i) NO_SKIP: produce all matches (ii) SKIP_TO_NEXT: discard partial matches that started with the same CE (iii) SKIP_PAST_LAST_EVENT: discard partial matches after CE match started but before it ends. Two more options are SKIP_TO_FIRST[p] and SKIP_TO_LAST[p] that are similar to SKIP_TO_NEXT and SKIP_PAST_LAST_EVENT, respectively, but use a pattern p to dictate the start (resp. last) event in the CE [10].

By integrating the EasyFlinkCEP operator on top of the streaming extension of a popular data science platform, RapidMiner Studio [26], users can draw workflows by simply dragging and dropping operators on a canvas. The parameters (P1)-(P4) are specified using graphical components. The first step after prescribing (P1)-(P4) is to press a submit button. The queried pattern is then handled by the EasyFlinkCEP Parser and Regex Analyzer (Figure 1) which processes and translates the Regex into a FlinkCEP program. Regex, i.e., (P1) may include disjunction, negation, quantifiers and the wild char. For example $a\{2,5\}b?(c|d)$ denotes that we seek between 2 and 5 appearances of event a, followed by zero or one event b, followed by events c or d. The final FlinkCEP program is created by the EasyFlinkCEP API Translator (Figure 1) which also adds to the created code the user specifications for (P2)-(P4).

Our experience shows that the EasyFlinkCEP Operator can reduce the time required to define and deploy new CEP analytics workflows from day(s) to minutes. To understand why, compare EasyFlinkCEP's simple Regex *ab+* and graphical definition of (P2)-(P4), with Listing 1 which provides part of the FlinkCEP code in Java querying for an equivalent pattern.

EasyFlinkCEP creates code ready to get executed as a FlinkCEP job in a computer cluster. But before job submission, the application has to specify the computer cluster resources, i.e., parallelism, of that FlinkCEP job. In our architecture (Figure 1) the EasyFlinkCEP Optimizer automates such cluster configuration decisions.

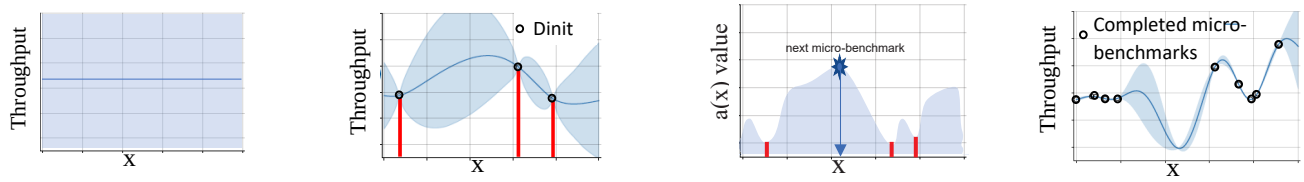
3 THE EasyFlinkCEP OPTIMIZER

Challenges. FlinkCEP operators internally function on automata [7, 12, 16]. Briefly, these automata have simple events (such as a, b in Listing 1) as their states and the occurrence of involved simple events triggers transitions from one state to another. Whenever an automaton reaches a final state, we say that a full match has been completed and a CE is outputted. But for an automaton that is in a non-final state, FlinkCEP has to keep and process all the partial matches that are valid based on the chosen contiguity, consumption and window specifications, since these may produce full matches for future streaming tuples. Therefore, FlinkCEP may have to process simultaneously multiple partial matches with each streaming tuple, which obviously affects CEP performance in terms of throughput, i.e. number of tuples being processed per time unit. Some relaxed contiguity strategies may produce a number of partial matches that grows exponentially to the number of input events [3, 31]. On the other hand, these partial matches may be reduced based on the specified window and consumption policy [12, 21]. Event consumption policies may also act as a bottleneck for a CEP operator, for example, halting its parallel execution to check if a given event has been consumed by other parallel instances of a pattern [12, 20].

Hence, it is hard to come up with an analytic formula to describe the expected performance of a CEP query and, therefore, the EasyFlinkCEP Optimizer treats a Regex as a black-box function attempting to learn its behavior–performance in terms of throughput for various parallelism degrees, contiguity, consumption and window specs. To do that, it encompasses the Benchmarking and the Cost Modeler Submodules (Figure 1).

The EasyFlinkCEP Benchmarking Submodule. When a previously unseen Regex, interpreted into a FlinkCEP program, is submitted, the Benchmarking Submodule of the EasyFlinkCEP Optimizer (Figure 1) performs a number of micro-benchmarking iterations, before actually deploying the CEP query in production. These micro-benchmarks focus only on the interpreted Regex and each runs a series of smaller scale FlinkCEP jobs, of limited duration each, testing the Regex under different sets of (Regex, contiguity, consumption, window, parallelism) specifications. Note that the submitted FlinkCEP program has fixed such specifications (excluding parallelism which is to be prescribed by the optimizer), while the Benchmarking Submodule runs jobs with different (contiguity, consumption, window) parameters. The aim is to learn Regex's expected throughput under different such specs and build a performance model that can be used any future time the same Regex is submitted, even if the exact specs are altered.

Micro-benchmarks may be executed over actual samples of data from the involved streams or on synthetic data streams produced



(a) Before D_{init}, blue area shows uncertainty covering the whole range of throughput values.

(b) Fitting GPR around micro-benchmarks in D_{init} (red lines). Blue line shows expected throughput.

(c) Acquisition function graph, $a(x)$ chooses next micro-benchmark x of high uncertainty.

(d) EasyFlinkCEP Cost Model: a GPR after a number of BO iterations - micro-benchmarks.

Figure 2: EasyFlinkCEP BO-based Optimizer Operation.

by data generators. The EasyFlinkCEP Optimizer collects statistics about the achieved throughput of each micro-benchmark using JMX [30]. The benchmarking process reserves cluster resources from other Flink jobs (only for a previously unseen Regex, though) and, thus, we want to keep these micro-benchmarks to a minimum.

The EasyFlinkCEP Cost Modeler Submodule. To sum up, we have a black-box function we wish to learn to estimate the throughput of the submitted (and any similar) FlinkCEP program under different parallelism configurations. We further want to accomplish that, using a limited number of benchmarks. This setup perfectly fits to a Bayesian Optimization (BO) approach [27, 28].

The first time we see a new Regex we know nothing about its expected throughput (Figure 2a). Therefore, the Benchmarking Submodule chooses a few x =(contiguity, consumption, window, parallelism) instances from the set of possible specification combinations and submits micro-benchmarks to the Flink cluster (right-hand side of Figure 1). Throughput statistics are monitored and are compiled in an initial benchmarking set D_{init} fed to the Cost Modeler upon the completion of the corresponding jobs. The outcome of the micro-benchmarks is a series of (contiguity, consumption, window, parallelism, throughput) tuples which will be used as training data. The BO approach used by the Cost Modeler builds an initial performance model fitting a Gaussian Process Regressor (GRP) around D_{init}, as shown in Figure 2b. Then, the Cost Modeler uses an acquisition function $a(x)$ in order to determine the (contiguity, consumption, window, parallelism) instance that should be used in the next micro-benchmark. The acquisition function determines the next micro-benchmark in order to improve the GPR in terms of describing the true throughput. Acquisition functions bargain exploitation and exploration. Exploitation means determining the next x point to be in areas where GPR predicts high/low values for throughput and exploration means choosing x where the prediction uncertainty (e.g. variance) is high. Both correspond to high $a(x)$ values and the goal is to optimize (maximize) $a(x)$ to determine the next benchmark (Figure 2c). Commonly used Acquisition Functions are Expected Improvement (EI), Lower Confidence Bound (LCB), Maximum Probability of Improvement (MPI) or Entropy Search. After a number of such micro-benchmark submission and GPR update iterations, EasyFlinkCEP's Cost Modeler ends up with an accurate GPR model that will be used in the future by the EasyFlinkCEP Optimizer every time the same Regex with any (contiguity, consumption, window) specifications is submitted (Figure 2d). Trained GPRs models are developed using the scikit-optimize [23] library.

When an interpreted FlinkCEP program with a specific (Regex, contiguity, consumption, window) is submitted, the EasyFlinkCEP Optimizer asks the trained GPR for a throughput estimation on x values composed of its (contiguity, consumption, window) specifications and various parallelism levels. For each queried parallelism level we have a different x. So the GPR replies back with the throughput values on the blue line in Figure 2d crossed at these x s. Among these responses, the EasyFlinkCEP Optimizer will choose the level of parallelism that achieves the highest increase in throughput with the minimum number of provisioned resources, i.e., the lowest parallelism, and will configure the respective FlinkCEP job to run accordingly. Applications of this scheme are provided in Section 4.

EasyFlinkCEP & Best Practices. We list a set of best practices, applied in Section 4, extracted from EasyFlinkCEP's adoption in the scenarios of Section 1.

(1) D_{init} should be constructed by sampling with probability ~10% from all possible (contiguity, consumption, window, parallelism) combinations of the specification space.

(2) Choosing LCB as an acquisition function results in GPRs that can better capture the trends in the benchmarked operator's throughput, compared to EI or MPI, with a lower number of BO iterations.

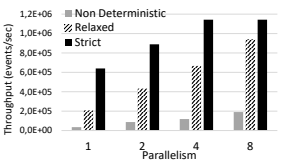
(3) The number of BO iterations required by LCB to establish accurate GPRs correspond to 1/3 of all possible (contiguity, consumption, window, parallelism) specifications.

(4) Contiguity, consumption, and parallelism in the x axis of Figure 2 are categorical features and also window values can be easily discretized based on the number of tuples in the window or its time duration. Therefore, we suggest the GPR model to get parameterized with a weighted Hamming Distance Kernel using equal weights for each specification category.

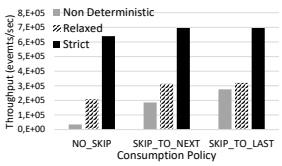
(5) The developed cost model can incrementally get updated for specific (Regex, contiguity, consumption, window, parallelism) with statistics from currently deployed CEP queries, avoiding the need for further micro-benchmarks.

4 EXPERIMENTS

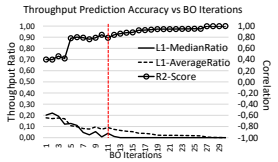
In this section we provide experimental results on the EasyFlinkCEP Operator, the performance of the produced FlinkCEP programs and the ability of the EasyFlinkCEP Optimizer to devise good FlinkCEP job execution configurations. All experiments were run in a VM with 16 cores and 16GB of main memory running Ubuntu 18.04 using the latest version of Flink. The input and output streams were instantiated as Kafka topics using a broker running in the same machine. We used 8 stock streams with 2M simple events each



(a) Varying the Parallelism.



(b) Varying Skip Policy.



(c) Performance Prediction Accuracy vs BO Iterations.

Figure 3: EasyFlinkCEP Performance Study.

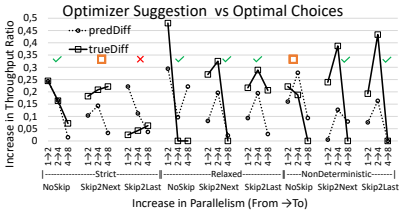


Figure 4: Job Config. Suggestions.

(16M in total). Each stream was populated using events denoted as lower case latin characters. In this particular scenario, we monitor patterns involving stock trade price trends with the following notation a: decreasing trend, b: increasing trend, c: steady trend and d: undetermined trend. We use a Regex of the form $ab\{1,3\}c|d$ and a tumbling (i.e., disjoint) window of 128 events per stream after discussions with the domain experts. The default consumption policy was NO_SKIP, but we vary that parameter as well. To increase the complexity in terms of partial matches (Section 3), we further inject 125K CEs at random positions within the input streams.

In Figure 3a we plot the achieved throughput (processed input events/sec) of the interpreted FlinkCEP program as we vary the degree of parallelism in the EasyFlinkCEP operator and the desired contiguity condition. We notice that all instances of the operator scale by increasing the parallelism except for the execution of the strict contiguity with parallelism = 8, as in that case, the speed of the operator is capped by the rate of reading data from Kafka. As expected, selecting the relaxed or the non-deterministic contiguity conditions results is significantly smaller throughput due to increased number of partial pattern matches monitored before detecting CEs and the larger number of CEs produced (see Section 3).

In Figure 3b we repeat the experiment with parallelism = 1 and vary the contiguity condition and consumption policy. Using a consumption policy that skips events after a match benefits mainly the relaxed and non-deterministic contiguities, as it helps limit the number of partial matches monitored and CEs produced.

Our next set of experiments concentrates on the number of required micro-benchmarks and the effectiveness of the EasyFlinkCEP Optimizer in devising good job execution configurations. Figure 3c plots the prediction accuracy of the trained BO model. In the horizontal axis we include the number of executed micro-benchmarks which correspond to BO iterations. In the first vertical axis (left-hand side of Figure 3c) we measure the L_1 error, then normalize the vertical axis to improve readability and include separate lines for the median and the average such error. In the second vertical axis (right-hand side of Figure 3c) we measure the R^2 score between BO predictions and actual throughput values. The concept is to use L_1 to deduce the prediction error in terms of absolute throughput values, i.e. $|predictedThroughput - actualThroughput|$, but also R^2 score to check the intuition that if the EasyFlinkCEP Optimizer (Cost Modeler in particular) can at least capture the trends of throughput, instead of the absolute values, it will still devise good job configurations. As Figure 3c demonstrates, the L_1 error drops below 5% after 20 BO iterations, while R^2 score exceeds 80% after, roughly, 10 micro-benchmarks – BO iterations.

To test our intuition, in Figure 4, we use a BO model trained with just 1/3 of the possible micro-benchmarks (red vertical line in Figure 3c where R^2 score > 80%) and we compare the optimal FlinkCEP configurations versus those prescribed by the EasyFlinkCEP Optimizer. The horizontal axis holds grouped triplets of all possible specifications regarding parallelism, consumption and contiguity for our Regex. The vertical axis plots the increase (percentage) in throughput upon increasing parallelism. We use two lines for the predicted throughput increase (predDiff) and the true one (trueDiff). The optimal job configuration is the one that achieves the highest increase in throughput with the lowest parallelism. So for instance, for the first triplet, the best FlinkCEP configuration under (Strict Contiguity, NO_SKIP) is to set parallelism to 2, since this parallelism provides the highest increase in throughput using the lowest possible resources (2 VMs or task slots etc). This is captured by EasyFlinkCEP since the (not necessarily the same) maximum for the corresponding predDiff, trueDiff lines is reached for the same value of parallelism. The same holds for the last two (8th and 9th) triplets under Non-Deterministic Contiguity and SKIP_TO_NEXT, SKIP_TO_LAST, respectively. Despite the fact that the maximum throughput increase value differs for the predDiff, trueDiff lines, they both show that the maximum is achieved by increasing parallelism to a value of 4. On the contrary, for the third triplet (Strict Contiguity, SKIP_TO_LAST) the maximum throughput increase is for setting parallelism to 8 (trueDiff), while predDiff erroneously shows that this happens for parallelism=2.

As the marks in Figure 4 show, 8/9 times the EasyFlinkCEP Optimizer devises good job configurations. In 6 out of 9 possible consumption, contiguity specifications, the EasyFlinkCEP Optimizer configured the FlinkCEP job in the optimal way (✓ signs), 2/9 times it devised the second best configuration (□ signs), while only 1/9 times (× sign) it reached a bad decision. Therefore, just capturing the throughput trends in terms of R^2 is proved sufficient to automatically devise good FlinkCEP job configurations, using just 1/3 (12/36) possible micro-benchmarks.

5 CONCLUSIONS

We present EasyFlinkCEP, a CEP component operating on top of a state-of-the-art parallel CEP engine, namely FlinkCEP. EasyFlinkCEP offers (i) the EasyFlinkCEP Operator which abstracts the details of coding directly on FlinkCEP, and (ii) the EasyFlinkCEP Optimizer to optimize FlinkCEP jobs executed in a Flink cluster of choice. Thus, business event analysts can focus on rapidly defining CEP queries and then exploit the processing capacity of modern hardware without any cluster administration background knowledge.

REFERENCES

[1] [n.d.]. Esper. http://esper.espertech.com/release-5.3.0/esper-reference/html/epl-operator.html#epl-operator-ref-keyword-regexp. [Online; accessed 06-June-2021].

[2] [n.d.]. Oracle CEP EPL Language Reference. https://docs.oracle.com/cd/E12839_01/apirefs.1111/e14304/operators.htm#EPLLR206. [Online; accessed 06-June-2021].

[3] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient pattern matching over event streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, Jason Tsong-Li Wang (Ed.). ACM, 147–160. https://doi.org/10.1145/1376616.1376634

[4] Manolis Pitsikalis, Alexander Artikis, Richard Dreo, Cyril Ray, Elena Camossi, and Anne-Laure Jousselme. 2019. Composite Event Recognition for Maritime Monitoring. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems, DEBS 2019, Darmstadt, Germany, June 24-28, 2019*. ACM, 163–174. https://doi.org/10.1145/3328905.3329762

[5] Alexandros Troupiotis-Kapeliaris, Konstantinos Chatzikokolakis, Dimitris Zissis, and Elias Alevizos. 2020. Experimental Comparison of Complex Event Processing Systems in the Maritime Domain. In *21st IEEE International Conference on Mobile Data Management, MDM 2020, Versailles, France, June 30 - July 3, 2020*. IEEE, 293–298. https://doi.org/10.1109/MDM48529.2020.00066

[6] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. 2018. Wayeb: a Tool for Complex Event Forecasting. In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018 (EPiC Series in Computing, Vol. 57)*, Gilles Barthe, Geoff Sutcliffe, and Margus Veanes (Eds.). EasyChair, 26–35. https://doi.org/10.29007/2s9t

[7] Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. 2017. Complex Event Recognition Languages: Tutorial. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*. ACM, 7–10. https://doi.org/10.1145/3093742.3095106

[8] Antonios Deligiannakis, Nikos Giatrakos, Yannis Kotidis, Vasilis Samoladas, and Alkis Simitsis. 2021. Extreme-Scale Interactive Cross-Platform Streaming Analytics – The INFORE Approach. In *Proceedings of the 2nd Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores, SEA Data 2021, Copenhagen, DenMark, August 20, 2021*. http://ceur-ws.org/, 7–13. http://ceur-ws.org/Vol-2929/paper2.pdf

[9] Flink. [n.d.]. https://flink.apache.org/. [Online; accessed 06-June-2021].

[10] FlinkCEP. [n.d.]. https://ci.apache.org/projects/flink/flink-docs-release-1.13/dev/libs/cep.html. [Online; accessed 06-June-2021].

[11] Ioannis Flouris, Nikos Giatrakos, Minos N. Garofalakis, and Antonios Deligiannakis. 2015. Issues in Complex Event Processing Systems. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 2*. IEEE, 241–246. https://doi.org/10.1109/Trustcom.2015.590

[12] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Complex event recognition in the Big Data era: a survey. *VLDB J.* 29, 1 (2020), 313–352. https://doi.org/10.1007/s00778-019-00557-w

[13] Nikos Giatrakos, David Arnu, Theodoros Bitsakis, Antonios Deligiannakis, Minos N. Garofalakis, Ralf Klinkenberg, Aris Konidaris, Antonis Kontaxakis, Yannis Kotidis, Vasilis Samoladas, Alkis Simitsis, George Stamatakis, Fabian Temme, Mate Torok, Edwin Yaqub, Arnau Montagud, Miguel Ponce de Leon, Holger Arndt, and Stefan Burkard. 2020. INforE: Interactive Cross-platform Analytics for Everyone. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, Mathieu d'Aquin, Stefan Dietze, Claudia Hauff, Edward Curry, and Philippe Cudré-Mauroux (Eds.). ACM, 3389–3392. https://doi.org/10.1145/3340531.3417435

[14] Nikos Giatrakos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. 2017. Complex Event Recognition in the Big Data Era. *Proc. VLDB Endow.* 10, 12 (2017), 1996–1999. https://doi.org/10.14778/3137765.3137829

[15] Alejandro Grez, Cristian Riveros, Martín Ugarte, and Stijn Vansummeren. 2020. On the Expressiveness of Languages for Complex Event Recognition. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark (LIPIcs, Vol. 155)*, Carsten Lutz and Jean Christoph Jung (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 15:1–15:17. https://doi.org/10.4230/LIPIcs.ICDT.2020.15

[16] J Hopcroft, R Motwani, and J Ullman. 2007. *Introduction to automata theory, languages, and computation.* Pearson/Addison Wesley.

[17] Antonios Kontaxakis, Antonios Deligiannakis, Holger Arndt, Stefan Burkard, Claus-Peter Kettner, Elke Pelikan, and Kathleen Noack. 2021. Real-time processing of geo-distributed financial data. In *DEBS '21: The 15th ACM International Conference on Distributed and Event-based Systems, Virtual Event, Italy, June 28 - July 2, 2021*, Alessandro Margara, Emanuele Della Valle, Alexander Artikis, Nesime Tatbul, and Helge Parzyjegla (Eds.). ACM, 190–191. https://doi.org/10.1145/3465480.3467842

[18] D. Luckham. 2008. Complex Event Processing in Financial Services. *Journal of Financial Services Technology* 2, 1 (2008), 13–19.

[19] Ruben Mayer. 2018. *Window-based data parallelization in complex event processing.* Ph.D. Dissertation. University of Stuttgart, Germany. https://nbn-resolving.org/urn:nbn:de:bsz:93-opus-ds-97440

[20] Ruben Mayer, Ahmad Slo, Muhammad Adnan Tariq, Kurt Rothermel, Manuel Gräber, and Umakishore Ramachandran. 2017. SPECTRE: supporting consumption policies in window-based parallel complex event processing. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Las Vegas, NV, USA, December 11 - 15, 2017*, K. R. Jayaram, Anshul Gandhi, Bettina Kemme, and Peter R. Pietzuch (Eds.). ACM, 161–173. https://doi.org/10.1145/3135974.3135983

[21] Ruben Mayer, Muhammad Adnan Tariq, and Kurt Rothermel. 2017. Minimizing Communication Overhead in Window-Based Parallel Complex Event Processing. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*. ACM, 54–65. https://doi.org/10.1145/3093742.3093914

[22] Yuan Mei and Samuel Madden. 2009. ZStream: a cost-based query processor for adaptively detecting composite events. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul (Eds.). ACM, 193–206. https://doi.org/10.1145/1559845.1559867

[23] Scikit optimize Library. [n.d.]. https://scikit-optimize.github.io/stable/. [Online; accessed 06-June-2021].

[24] Kostas Patroumpas and Timos K. Sellis. 2006. Window Specification over Data Streams. In *Current Trends in Database Technology - EDBT 2006, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 4254)*, Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen (Eds.). Springer, 445–464. https://doi.org/10.1007/11896548_35

[25] Srinath Perera and Sriskandarajah Suhothayan. 2015. Solution patterns for realtime streaming analytics. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, Oslo, Norway, June 29 - July 3, 2015*, Frank Eliassen and Roman Vitenberg (Eds.). ACM, 247–255. https://doi.org/10.1145/2675743.2774214

[26] Streaming Extension RapidMiner. [n.d.]. https://marketplace.rapidminer.com/UpdateServer/faces/product_details.xhtml?productId=rmx_streaming. [Online; accessed 06-June-2021].

[27] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* 104, 1 (2016), 148–175. https://doi.org/10.1109/JPROC.2015.2494218

[28] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. , 2960–2968 pages. https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html

[29] Eleni Kougioumtzi, Antonios Kontaxakis, Antonios Deligiannakis, and Yannis Kotidis. 2021. Towards creating a generalized complex event processing operator using FlinkCEP: architecture & benchmark. In *DEBS '21: The 15th ACM International Conference on Distributed and Event-based Systems, Virtual Event, Italy, June 28 - July 2, 2021*, Alessandro Margara, Emanuele Della Valle, Alexander Artikis, Nesime Tatbul, and Helge Parzyjegla (Eds.). ACM, 188–189. https://doi.org/10.1145/3465480.3467841

[30] Oracle JMX Technology. [n.d.]. https://docs.oracle.com/javase/tutorial/jmx/overview/. [Online; accessed 06-June-2021].

[31] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2014. On complexity and optimization of expensive queries in complex event processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 217–228. https://doi.org/10.1145/2588555.2593671