2018 17th IEEE International Conference on Machine Learning and Applications

STARLORD: Sliding window Temporal Accumulate-Retract Learning for Online Reasoning on Datastreams

Cristian Axenie
Huawei German Research Center
Munich, Germany
cristian.axenie@huawei.com

Radu Tudoran
Huawei German Research Center
Munich, Germany
radu.tudoran@huawei.com

Stefano Bortoli
Huawei German Research Center
Munich, Germany
stefano.bortoli@huawei.com

Mohamad Al Hajj Hassan
Huawei German Research Center
Munich, Germany
mohamad.alhajjhassan@huawei.com

Daniele Foroni
Information Engineering and Computer Science Dept.
University of Trento
Trento, Italy
daniele.foroni@unitn.it

Goetz Brasche
Huawei German Research Center
Munich, Germany
goetz.brasche@huawei.com

***Abstract*—Nowadays, data sources, such as IoT devices, financial markets, and online services, continuously generate large amounts of data. Such data is usually generated at high frequencies and is typically described by non-stationary distributions. Querying these data sources brings new challenges for machine learning algorithms, which now need to be considered from the perspective of an evolving stream and not a static dataset. Under such scenarios, where data flows continuously, the challenge is how to transform the vast amount of data into information and knowledge, and how to adapt to data changes (i.e. drifts) and accumulate experience over time to support online decision-making. In this paper, we introduce STARLORD, a novel incremental computation method and system acting on data streams and capable of achieving low-latency (millisecond level) and high-throughput (thousands events/second/core) when learning from data streams. Moreover, the approach is able to adapt to data drifts and accumulate experience over time, and to use such knowledge to improve future learning and prediction performance, with resource usage guarantees. This is proven by our preliminary experiments where we built-in the framework in an open source stream engine (i.e. Apache Flink).**

***Index Terms*—Streaming Data, Online Machine Learning, Distributed Computation, Incremental Computation.**

I. Introduction

Data streams can be seen as stochastic processes in which events occur continuously and independently from each other. Querying, processing and executing machine learning models on data streams is quite different from querying static data stored in the conventional database model, as data might be transient and follow a non-stationary distribution.

Such a context opens new problems in designing machine learning algorithms, that now need to learn from continuous data in a single pass and, potentially, discard data at fixed moments in time [1], [2]. Take for example the case of a 6h moving average calculation of online transactions before Christmas for people located in Berlin.

To handle this self-adjusting computation was developed [3], [4], where computations adjust incrementally to drifts in the data [5] and [2]. In recent years, it has been shown that incremental computation delivers large, even asymptotic speed-ups over full re-evaluation in simple computations [6], or more complex aggregations [7], [8], [9].

Such models can be plugged into data-parallel programming models allowing computation decompositions into associative sub-computations, such as in Apache Spark [10]. However such approaches do not consider the case of large windows (e.g. 1 Mevents representing bank transaction data in the last 3 months for fraud detection) and the resource constrains to store and pass through the data despite the optimized data structures. All these approaches paved the way for what recent years have witnessed an increasing interest: incremental learning on streams [1], [11], [12], [13], [14]. As a consequence many incremental solutions for all kinds of machine learning algorithms, such as support vector machines [15], neural networks [16], or Bayesian models [17] have been developed.

The proposed method and system come as an alternative to well established approaches such as the Massive On-line Analysis (MOA). As a new paradigm, it does not try to optimize the execution of classical machine learning algorithms on distributed systems, [18], [19], [20], rather it analyzes the decoupling between the model structure and parameters, such that we gain flexibility over the traditional approaches [1].

In this context, the first contribution of our system is the capability to integrate previously learnt knowledge with recently received data, in order to ensure low-latency learning. Second, the proposed computation model targets accumulating experience over time to support future decision making with high-throughput and constant resource allocation.

978-1-5386-6805-4/18/$31.00 ©2018 IEEE
DOI 10.1109/ICMLA.2018.00181

II. A DATA FLOW MODEL FOR INCREMENTAL COMPUTATION ON DATA STREAMS

The paper introduces a novel method and a system for data processing and data management for online machine learning, termed STARLORD (Sliding window Temporal Accumulate-Retract Learning for Online Reasoning on Datastreams). It offers a solution for incrementally computing on data streams simple, complex or combinations of features, from statistical measures to machine learning model parameters. The driving design requirements for STARLORD were chosen to allow for:

- low latency - because in many stream applications the speed of reactions to drifts in the data is a fundamental requirement (e.g., IoT or financial applications);
- high throughput - due to the exponential growing rate at which new data is generated and needs to be processed (e.g. applications and services in IoT that easily reach thousands to millions events per second).
- bounded memory allocation - keep memory usage fixed (i.e. only relevant "hot" data) and disk usage flexible (i.e. processed "cold" data) as the amount of events in the data stream increases and can reach very high values (e.g., millions-billions of events) as the stream evolves;
- flexibility - allow arbitrary combinations of analytics to be calculated on the stream, with no time and resource penalty, by exploiting the underlying hardware, data processing and data management.

A. Incremental computation on data streams in distributed systems

In a nutshell, stream processing considers a sequence of data (the stream) and a series of operations (functions) that are applied to each element in the stream, in a declarative way and predefined order (i.e. dataflow model, as defined in [21]). Traditional machine learning is described by finite training sets, static models, and stationary distributions. In streaming context all these conditions must be redefined and the typical pipeline needs to be redesigned as shown in Figure 1.

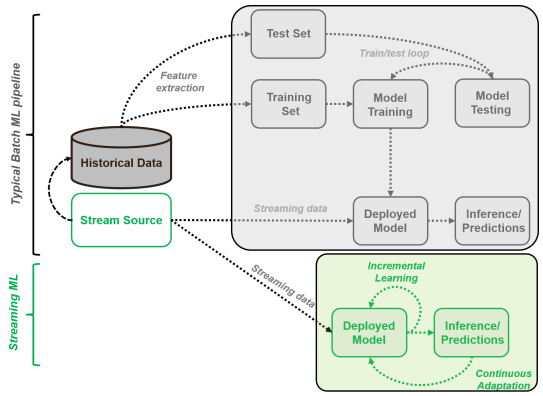

Fig. 1. Typical processing pipeline for batch and for streaming.

Such redefinition implies the addition of a new pipeline, the production pipeline, which is supplementing the creation pipeline, shown in Figure 1 upper panel, by embedding the actual learnt model in the data stream and using it for inference and continuous adaptation, as shown in Figure 1 lower panel.

A new paradigm where there is no need to recompute features by looping through all the acquired data, rather only through simple updates based only on the new values [22], [23], eliminated values and the old feature value [24], [25]. Exploring the existing approaches and systems we can identify two main problems for online machine learning: 1) computing time is too large, as it typically implies recomputing features and learning rules by looping through all acquired/incoming data, and 2) memory is used greedily to accommodate the large amount of data. STARLORD tries to tackle these two problems and is among the first attempts towards this paradigm shift initially set by MOA and algorithms like ADWIN [26].

Despite the fact that ADWIN implementations have theoretical guarantees they do not keep constraints on execution time and resource allocation the core focus of our system. We make a trade-off between these two guarantees to be able to achieve inference with low-latency and high-throughput. Similar to ADWIN a main advantage of STARLORD is that it does not require no guess about how fast or how often the stream will drift, we continuously estimate that while updating the models.

In the upcoming sections we introduce the proposed framework through sample instantiations for calculating complex aggregates (i.e. statistical moments, descriptive statistics, and correlations) and performing supervised learning (i.e. learning a regression function and making predictions simultaneously).

B. Instantiating the model: incremental computation of statistical measures on data streams

Important information about the data generating process are available in summary statistics. Such computations can also capture the characteristics of the data distribution, such as mean, variance, skewness, and correlation coefficient.

The application of statistical methods to data streams requires modifications to the standard calculation schemes over samples and populations in order to be carried out in an online fashion.

Our incremental computation uses dual operations that are applied on the current window of events over the stream. The first operation is accumulation, where the contribution of the latest event added in the window of interest is calculated and added to the overall estimation of the feature at the next step. In a similar manner, after the window slides in time, one or more elements are eliminated from the window. In order to keep a consistent estimate the dual operation (i.e. retraction) is performed. This assumes that the contributions of each event to be removed from the window is calculated and then deducted out of the global estimate at the next step. This approach brings clear advantages over traditional approaches, such as decay techniques [27] and supports window aging principle (i.e. discount for the staleness of certain data elements). This

is due to the fine control of event processing and determinism (i.e. offering the same results as batch execution). Furthermore, adjusted to current privacy requirements of the General Data Protection Regulation (GDPR) [28] , the accumulate/retract framework offers the user the control over data management (i.e. accumulate/retract is transparent and offers user the control on the impact of the incremental operations, opposite to decay techniques [29]).

The core steps are introduced in Figure 2, exemplifying an incremental sum calculation. While the the window slides an event $x_a(t)$ is added to the window and another is removed, $x_r(t)$. The accumulate/retract framework assumes that instead of recalculating the sum for all events in the window, the system just calculates and embeds the contribution of $x_a(t)$ and calculates and discards the contribution of $x_r(t)$.

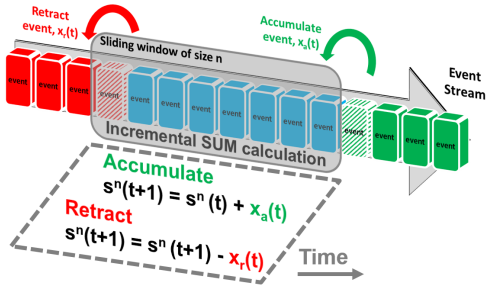

Fig. 2. Incremental computation in the accumulate/retract framework.

In the following, we introduce sample mathematical derivations under the accumulate/retract framework, which applies to all instantiations of STARLORD, as a core design principle.

For example, in order to measure the location of the distribution of the data in the incoming stream, an incremental estimate of the mean of a window/sample of size n at time t, $\bar{x}^n(t)$ needs to be calculated. The incremental version of mean calculation, in Equation 1, is based on the sample formula (Equation 2) and assumes that the mean $\bar{x}^n(t)$ depends on the previously calculated mean $\bar{x}^n(t-1)$ plus a sample size dependent increment, which quantifies the change in mean while the stream progresses by processing a new event $x(t)$.

$$\bar{x}^n(t) = \bar{x}^n(t-1) + \frac{1}{n}(x(t) - \bar{x}^n(t-1)) \tag{1}$$

where,

$$\bar{x}^n(t-1) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2}$$

Such an equation provides the correct estimate only in the case events are added to the window of interest (i.e. accumulation). In our case the sliding window dynamics also considers the elimination of events from the window (i.e. retraction). Such an approach facilitates the derivation of any machine learning algorithm (i.e. basically an iterative process towards convergence) in a form suitable to simply update the model on the incoming and eliminated values in the window.

The dual update rules for accumulation and retraction are shown in the Figure 3.

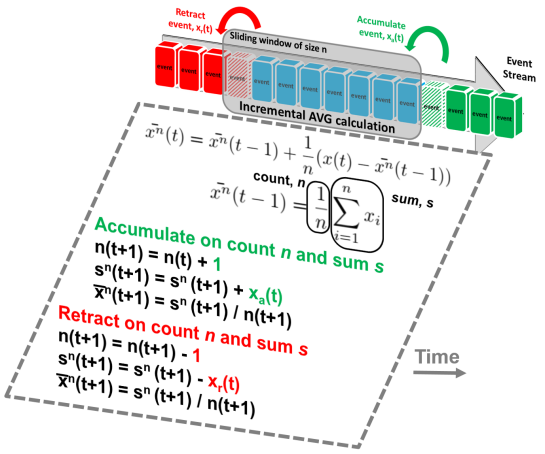

Fig. 3. Dual accumulate/retract update rules for incremental mean calculation.

Another statistical measure for stream data is the spread (i.e. variance). Interestingly, we can look at more subtle properties of the data, such as the central statistical moments to compute variance incrementally. The first central moment provides the mean, $\bar{x}^n(t)$, whereas the spread is captured by the second central moment, as shown in Equation 3

$$\tilde{m_2^n}(t) = \tilde{m_2^n}(t-1) + (x(t) - \bar{x}^{n-1}(t))(x(t) - \bar{x}^n(t)), \tag{3}$$

where $\tilde{m}_{k,t} = nm_{k,t}$.

The derivation suitable for the accumulate/retract framework is computed taking into account the incremental nature of the moment estimation over the sliding window and the impact that the incremental addition and removal of events has upon the estimate at each step, as shown in Figure 4.

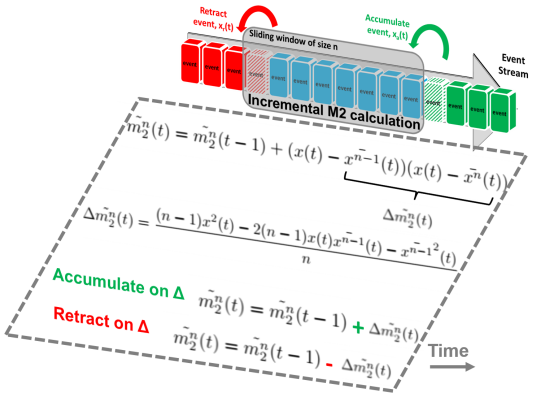

Fig. 4. Dual accumulate/retract update rules for incremental statistical central moment order 2 calculation.

For more advanced statistics one needs to calculate also higher moments, computing, for example, the shape of the distribution of the evolving stream (e.g. third central moment, Equation 5) incrementally.

$$\tilde{m_3^n}(t) = \tilde{m_3^n}(t-1) - 3\frac{x(t) - \bar{x}^{n-1}(t)}{n}\tilde{m_2^n}(t-1) \quad (4)$$

$$+\frac{(n-1)(n-2)}{n^2}(x(t) - \bar{x}^{n-1}(t-1))^3 \quad (5)$$

Such features are used in nonlinear combinations with others to compute statistics of the generating process underlying the stream, such as skewness. Such a metric uses hierarchically the calculation of second moment in Equation 3, and third moment, derived in Equation 5 :

$$skew^n(t) = \sqrt{n}\frac{\sqrt{n(n-1)}}{n-2}\frac{\tilde{m_3^n}(t)}{\tilde{m_2^n}(t)\sqrt{\tilde{m_2^n}(t)}} \quad (6)$$

In order to support such incremental computation the decoupling and the orchestration of the execution flow on the computing infrastructure must be also taken into account. The core component of STARLORD is a global feature extraction (GFE) system. GFE separates the high-level feature extraction and processing from the low level data management, supporting different time scales and the composition of features in a hierarchical manner, as shown in the skewness computation (Equation 6). The architecture is capable of judiciously dispatching computation and data depending on the relevance it has at each step.

Such an approach decouples the details of data management and resource allocation allowing the system to compute several features simultaneously for large windows of events with low latency. This is supported by an optimized flow orchestration that ensures a constant response time at high input rates and large windows (100K to 1M events), as shown in Figure 5.

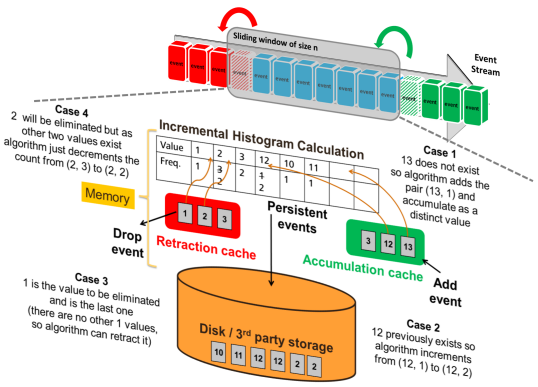

Fig. 5. Orchestrating computation for streaming incremental computation.

The orchestration provides the underlying mechanisms that for each newly received event in the focus window, checks the validity of already saved events, and adds the newest in the accumulation cache and removes oldest from retraction cache. The newest event is then used in two concurrent processes, namely the decision to add the event to cache or disk, and the actual feature(s) update. Next, depending on the decision taken and the previous step the actual event fetching operation takes place and a new update is triggered in a similar manner. Such an orchestration ensures that for each new event (i.e. window slide) the system provides a new estimate of the feature(s).

C. Instantiating the model: incremental linear least-squares regression on data streams

In this context, in order to instantiate our approach for incremental computation, we chose a simple learner design problem, namely a linear least-squares regressor (LLS).

The long term goal of the work is to provide a flexible framework for incremental machine learning, for models which have a closed form and approximations for those which are hard to formulate in closed form. Due to its closed form can be computed from a set of incremental statistics. The implementation assumes that the stream data will feed the system for both learning the parameters of the model (i.e. interpolation) and predicting new values (i.e. extrapolation).

The learner tries to minimize the sum squares of the deviations of a set of n data points

$$S^2 = \sum [y - f(x_i, \beta_1, \beta_2, \beta_3, ..., \beta_N)]^2 \quad (7)$$

which, for the linear case, it simply minimizes the following criteria

$$S^2 = \sum [y - (\beta_1 + \beta_2 x)]^2. \quad (8)$$

The problem can be easily expressed in closed form as statistical quantities (i.e. mean, variance, covariance) which can be incrementally extracted from the evolving stream, as shown in Figure 6. Here the actual derivation is using, hierarchically, the incremental mean, variance, and covariance, of x and y to provide an estimate of the regression parameters β.

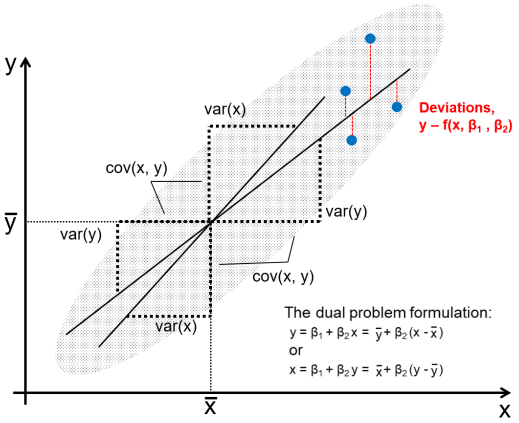

Fig. 6. Linear least squares (LLS) computation.

For example, incremental covariance can be calculated for the incoming samples x_t and y_t based on the statistical central moment one (Equation 1,2) introduced earlier, as

$$cov_{xy}(t) = \frac{n-2}{n-1} cov_{xy}(t-1) \quad (9)$$

$$+\frac{1}{n}(x^n(t) - \bar{x}^{n-1}(t))(y^n(t) - \bar{y}^{n-1}(t)) \quad (10)$$

Finally the optimization problem assumes that the parametrized regressor predicts the next value (i.e. describing a line) in the evolving data stream using the following incrementally updated parameters:

$$\beta_2(t) = \frac{cov_{xy}(t)}{\tilde{m}_n^2(t)}, \beta_1(t) = \bar{y}(t) - \beta_2 \bar{x}(t) \quad (11)$$

Another important aspect in data stream analysis is that the data generating process does not remain static, i.e., the underlying probabilistic model cannot be assumed to be stationary. Incremental computation is intrinsically capturing such drifts and adapts to the irregularities of the data generation process, providing correct updates exploiting the accumulate/retract framework, as shown in Equations 10,11, avoiding recomputing the model by looping through all data in the window by just calculating the impact the newest and oldest event. With additional mathematical treatment we were able to derive other incremental accumulate-retract models for stochastic predictors (e.g. ARMA family) and Bayesian Inference. There is a clear limitation in consistently rewriting the algorithms. Yet the core is to find the correct match between computations and resource allocation (e.g. cache vs. disk, single vs multi-core) from the perspective of the distributed engine processing it.

STARLORD provides a solution for learning on streaming data, such that models can be updated on-the-fly with scalable, low-latency, and high-throughput processing.

III. Experiments and Evaluation

This section introduces the results and the analysis for the multiple instantiations of our framework using Apache Flink as underlying streaming engine [30]. Flink is an open source system for parallel scalable data processing supporting expressive, declarative, fast, and efficient data processing on real-time streaming data. At its core, Flink builds on an optimized distributed dataflow runtime (Figure 7) that supports incremental computations on streams, crucial in obtaining low-latency high-throughput online machine learning. STARLORD utilizes the algorithm dispatching and execution capabilities of Flink (Figure 7) but not the standard aggregates for computation. Later in the section, we compare the performance of our approach against the standard aggregates and machine learning capabilities in Flink and Apache Spark.

The experimental setup (Figure 7) for our tests used 3 machines, each with 24 CPU cores and 132 GB RAM, and Flink for processing and cluster management. During the experiments we consider a fixed sliding window, but the cache-disk orchestration mechanism can support also adaptive windowing. At the same time the caches (i.e. in RAM) mechanism allows to maintain new and old data in order to allow the retraction of individual stream events when sliding. The size of the window in our experiments varied between 100K and 1M events.

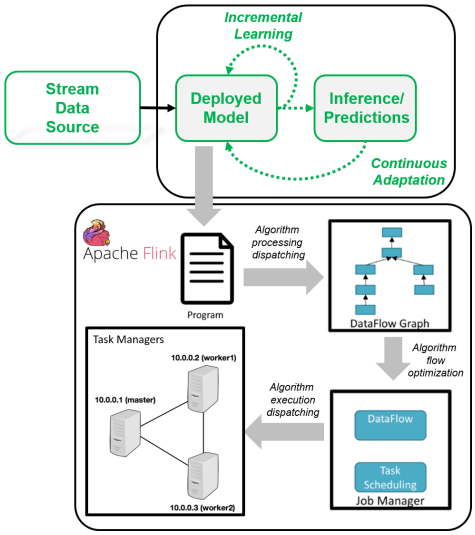

Fig. 7. Experimental setup and execution model of the framework.

A. Incremental statistics on streams

In order to test the incremental computation of statistical measures we used a real-world stream with transactions and queried a set of compound statistics (i.e. AVG, VAR, SKEW). We streamed 303 Mevents at 16 KHz (i.e. online transactions for Black Friday scenario, window size 1Mevents). As we are interested in analyzing the performance of our system on large windows of fast data, we evaluate the latency (Figure 8) and throughput (Figure 9) of the system.

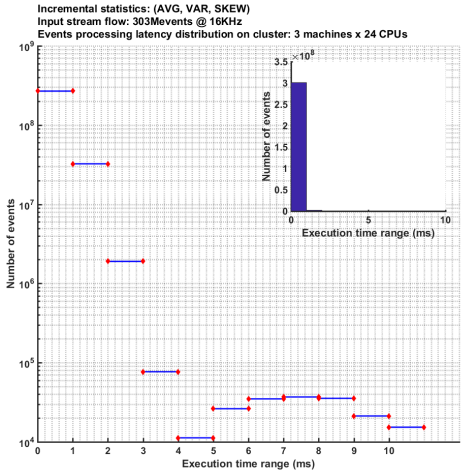

Fig. 8. Experimental results: latency on incremental statistical measures.

In Figure 8 we have depicted the overall latency distribution. This is upper bounded to 10 ms and centered on 1 ms, advocat-

ing an average constant processing time per event. The speed-up is given by our execution pipeline that contains multiple steps, each corresponding to the stages in the orchestration (i.e. Figure 5 adding / removing events in the caches given the sliding focus window, read of disk events for updates). The proposed system is capable to support such low-latency processing through its judicious resource allocation and the incremental processing scheme, as shown in Figure 5. As we can see in Figure 9, the throughput value is centered on an average of $\sim$12 Kevents/s, whereas the variations (up to 2 standard deviation) are mainly caused by the non-uniformity in data ingestion and transport (such as back-pressure), outside the framework.

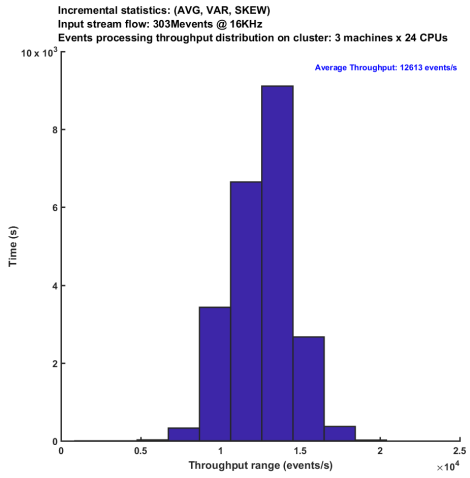

Fig. 9. Experimental results: throughput on incremental statistical measures.

In order to evaluate STARLORD against state-of-the-art implementations we designed similar experiments on two streaming engines, Apache Flink and Apache Spark respectively. The experiments were performed with Flink version 1.4.2 and Spark version 2.3.1 respectively. While in Flink we had to implement ourselves the custom code for the statistical measures, Spark already embeds these functions in its APIs. Hence, for the Flink implementation, we used the same incremental approach as used in our framework (Equations 1 to 6), but without the data orchestration support. For the Spark implementation, due to its different mode of operation (i.e. micro-batches - a record may have to wait for the current micro-batch to be completed), we needed to write custom functions to merge different already aggregated results. Moreover, the computation of AVG, VAR, SKEW was sequential (i.e. one aggregation after the other) and we needed to implement a parametrization trade-off for latency and throughput evaluation (i.e. 2s watermark).

In terms of latency, Flink standard aggregations are distributed between 2 and 55 ms (Figure 10, upper panel), explained by the fact that the incremental approach can offer a speed up in calculation. The Spark approach, due to it's micro-batching mechanisms, introduces unreasonable delays (Figure 10, lower panel), despite the processing parametrization we performed. Recently, Spark has introduced a new "millisecond low-latency mode" called continuous mode (i.e. reaching 100 ms latency), yet this is closed source from Databricks [31].

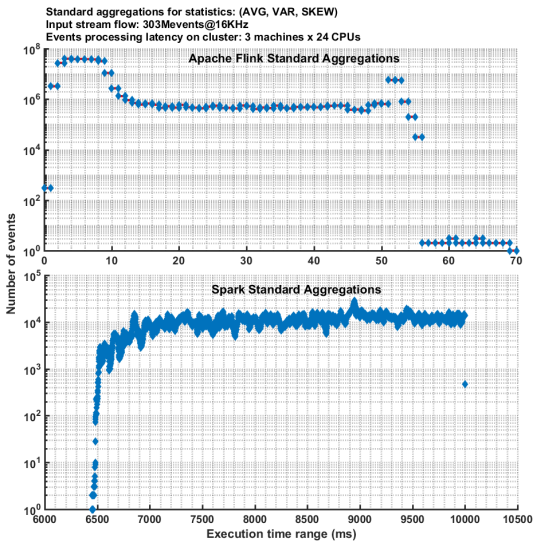

Fig. 10. Experimental results: Comparative evaluation of Apache Flink and Apache Spark for latency on statistical measures.

In terms of throughput, we observed that Spark Aggregations have a skewed distribution, centered on 20 Kevents/s (Figure 11, left panel), whereas the Flink implementation, on the other side, obtained a relatively multimodal distribution of throughput values. This is determined by the custom implementation of the aggregates in incremental form and ingestion irregularities (Figure 11, right panel) without the management offered by the resource orchestration. The sliding window size for this series of experiments was 100Kevents and was maintained fixed.

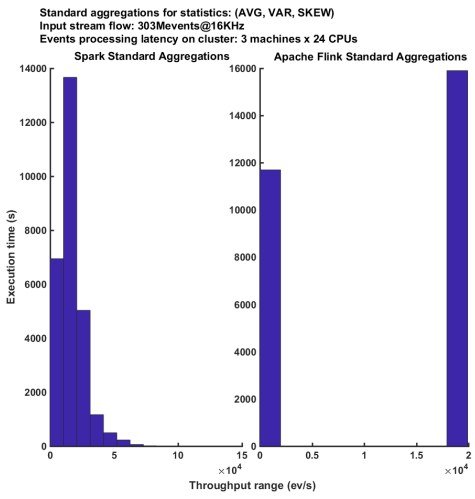

Fig. 11. Experimental results: Comparative evaluation of Apache Flink and Apache Spark for throughput on statistical measures.

B. Incremental Linear Least Squares

In order to evaluate the incremental learning capabilities of the three systems, we implemented an LLS regressor and fed it with 400 Kevents (i.e. a stationary time series) streamed at 16 KHz for the online training (i.e. estimating the β coefficients) and online prediction on windows of 100Kevents. Figure 12 and Figure 13 depict the latency and throughput analysis for the learning scenario using our framework. We observed that, despite the more complex computation, with respect to the simple statistical measures, the system latency is centered around 1 ms. The other two implementations, in Flink and Spark, are in general slower (Figure 14) despite using the same update formula (Equation 11).

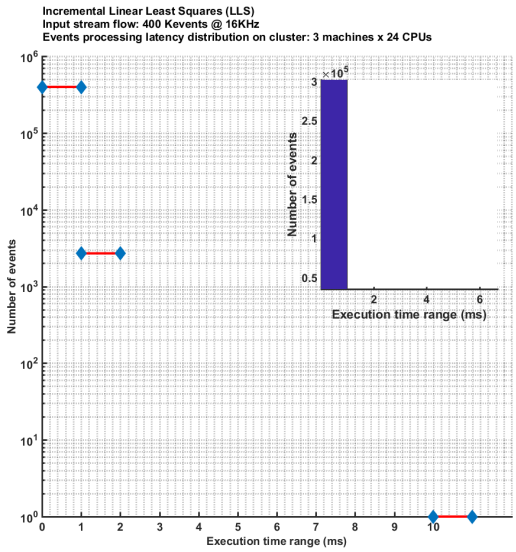

Fig. 12. Experimental results: latency on incremental LLS learning and prediction.

In terms of throughput, the combination of incremental formulation and data orchestration (i.e. accumulate / retract) introduced by STARLORD (Figure 13) provides a higher overall throughput (∼14 Kevents/s) with respect to Flink and Spark implementations of the same update formula (but without the orchestration), as shown in Figure 14 right panel. The sliding window size for this series of experiments was 100Kevents and was maintained fixed.

The experiments were meant to emphasize the advantages that the combined incremental mathematical formulation and the optimized data orchestration can leverage incremental processing in online machine learning. The evaluation introduced the general capabilities, but also limitations and differences among different systems, emphasizing how such a framework can leverage online machine learning implementations. Our focus is high-performance computation, yet a current drawback of the method is the effort to develop closed forms of new algorithms in the accumulate - retract framework while preserving consistency with the batch version. The experiments code and sample data can be found at [32].

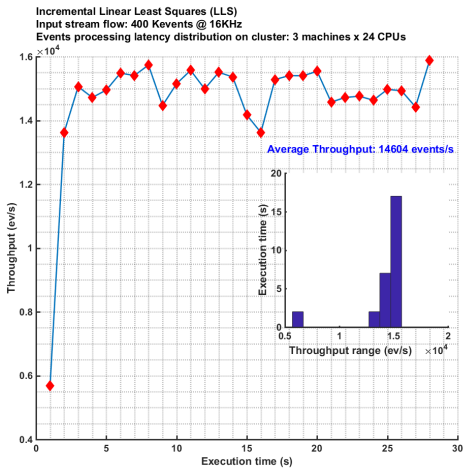

Fig. 13. Experimental results: throughput on incremental LLS learning and prediction.

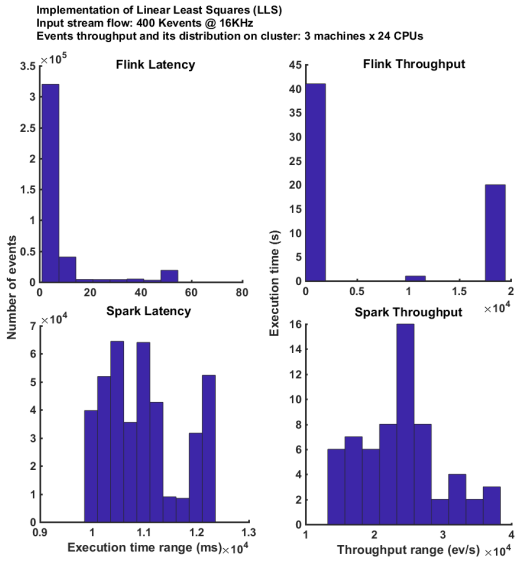

Fig. 14. Experimental results: Comparative evaluation of Apache Flink and Apache Spark for latency and throughput on LLS learning and prediction.

IV. Conclusion

This paper introduces STARLORD, a novel solution for low-latency (1-ms level), high-throughput (Kevents/s/core) computation and learning on streams. This framework leverages the capabilities of distributed incremental computation for online machine learning algorithms targeting real-time applications. Typically, computing features incrementally, for large windows, implies keeping large states in memory and recomputing the functions for every incoming event while trying to keep up with the timing requirements. This is not usually a tractable solution in real-time processing applications. STARLORD provides a solution for implementing online machine learning with very low latency over large event windows by

deriving incremental, accumulate/retract update models for algorithms and leveraging their execution on a distributed system. It offers simultaneous computation of multiple features and learning rules while guaranteeing limited or programmable resource allocation (i.e. memory and disk) by using optimized algorithms and execution for reduced latency. Our preliminary results on simple statistical aggregates, depicted in Figure 8 and Figure 9, and the incremental linear regressor, depicted in Figure 12 and Figure 13, prove the low-latency high-throughput capabilities of our proposed solution.

REFERENCES

[1] G. H. Albert Bifet, Ricard Gavald and B. Pfahringer, *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press, 2018.

[2] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, and T. Radauer, "Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances," *Inf. Sci.*, vol. 355, no. C, pp. 127–151, Aug. 2016. [Online]. Available: https://doi.org/10.1016/j.ins.2016.03.034

[3] U. A. Acar, "Self-adjusting computation:(an overview)," in *Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*. ACM, 2009, pp. 1–6.

[4] M. A. Hammer, K. Y. Phang, M. Hicks, and J. S. Foster, "Adapton: Composable, demand-driven incremental computation," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 14. New York, NY, USA: ACM, 2014, pp. 156–166. [Online]. Available: http://doi.acm.org/10.1145/2594291.2594324

[5] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA 2004*, A. L. C. Bazzan and S. Labidi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 286–295.

[6] O. Sümer, U. A. Acar, A. T. Ihler, and R. R. Mettu, "Adaptive exact inference in graphical models," vol. 12. JMLR.org, Nov. 2011, pp. 3147–3186. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078207

[7] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, "Incoop: Mapreduce for incremental computations," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 7:1–7:14. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038923

[8] P. Bhatotia, P. Fonseca, U. A. Acar, B. B. Brandenburg, and R. Rodrigues, "ithreads: A threading library for parallel incremental computation," ser. ASPLOS '15. New York, NY, USA: ACM, 2015, pp. 645–659. [Online]. Available: http://doi.acm.org/10.1145/2694344.2694371

[9] P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang, "Nectar: Automatic management of data and computation in datacenters," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 75–88. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924949

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[11] J. Yang and J. Widom, "Incremental computation and maintenance of temporal aggregates," in *Data Engineering, 2001. Proceedings. 17th International Conference on*. IEEE, 2001, pp. 51–60.

[12] T. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee, "Estimating statistical aggregates on probabilistic data streams," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 4, p. 26, 2008.

[13] H. He, S. Chen, K. Li, and X. Xu, "Incremental learning from stream data," *IEEE Transactions on Neural Networks*, vol. 22, no. 12, pp. 1901–1914, Dec 2011.

[14] L. Popa, M. Budiu, Y. Yu, and M. Isard, "Dryadinc: Reusing work in large-scale computations," in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855533.1855554

[15] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Proceedings of the 13th International Conference on Neural Information Processing Systems*, ser. NIPS'00. Cambridge, MA, USA: MIT Press, 2000, pp. 388–394. [Online]. Available: http://dl.acm.org/citation.cfm?id=3008751.3008808

[16] S. Furao, T. Ogura, and O. Hasegawa, "An enhanced self-organizing incremental neural network for online unsupervised learning." *Neural Networks*, vol. 20, no. 8, pp. 893–903, 2007.

[17] R. C. Wilson, M. R. Nassar, and J. I. Gold, "Bayesian online learning of the hazard rate in change-point problems," *Neural Computation*, vol. 22, no. 9, pp. 2452–2476, 2010.

[18] J. Gama, *Knowledge Discovery from Data Streams*, 1st ed. Chapman & Hall/CRC, 2010.

[19] M. Sayed-Mouchaweh and E. Lughofer, *Learning in Non-Stationary Environments: Methods and Applications*. Springer Publishing Company, Incorporated, 2012.

[20] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *European Symposium on Artificial Neural Networks (ESANN)*, 2016.

[21] T. Akidau and R. e. a. Bradshaw, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1792–1803, Aug. 2015. [Online]. Available: http://dx.doi.org/10.14778/2824032.2824076

[22] D. Logothetis, C. Olston, B. Reed, K. C. Webb, and K. Yocum, "Stateful bulk processing for incremental analytics," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 51–62. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807138

[23] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 251–264. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924961

[24] C. Demetrescu, D. Eppstein, Z. Galil, and G. F. Italiano, in *Algorithms and Theory of Computation Handbook*, M. J. Atallah and M. Blanton, Eds. Chapman & Hall/CRC, 2010, ch. Dynamic Graph Algorithms, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1882757.1882766

[25] Z. Huang and P. Peng, "Dynamic graph stream algorithms in space," *CoRR*, vol. abs/1605.00089, 2016. [Online]. Available: http://arxiv.org/abs/1605.00089

[26] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis VIII*, N. M. Adams, C. Robardet, A. Siebes, and J.-F. Boulicaut, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 249–260.

[27] E. Cohen and M. Strauss, "Maintaining time-decaying stream aggregates," in *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '03. New York, NY, USA: ACM, 2003, pp. 223–233. [Online]. Available: http://doi.acm.org/10.1145/773153.773175

[28] (2018) 2018 reform of eu data protection rules: Regulation 2016/6791, the european unions new general data protection regulation (gdpr). [Online]. Available: goo.gl/ZFw93G

[29] S. Dawar, V. Sharma, and V. Goyal, "Mining top-k high-utility itemsets from a data stream under sliding window model," *Applied Intelligence*, vol. 47, no. 4, pp. 1240–1255, Dec 2017. [Online]. Available: https://doi.org/10.1007/s10489-017-0939-7

[30] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *IEEE Data Eng. Bull.*, vol. 38, pp. 28–38, 2015. [Online]. Available: https://flink.apache.org/introduction.html

[31] T. D. J Torres, M Armbrust and S. Zhu. (2018) Introducing low-latency continuous processing mode in structured streaming in apache spark 2.3. [Online]. Available: https://databricks.com/blog/2018/03/20/low-latency-continuous-processing-mode-in-structured-streaming-in-apache-spark-2-3-0.html [accessed 03.07.2018]

[32] (2018) Experiments code for paper submitted at icmla2018. [Online]. Available: https://github.com/omlstreaming/icmla2018