Extreme-Scale Interactive Cross-Platform Streaming Analytics – The INFORE Approach

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

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

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

Vasilis Samoladas
Athena Research Center
Technical University of Crete
vsam@softnet.tuc.gr

Alkis Simitsis
Athena Research Center
alkis@athenarc.gr

ABSTRACT

Nowadays, analytics are at the core of many emerging applications and can greatly benefit from the abundance of data and the progress in the processing capabilities of modern hardware. Still, new challenges arise with the extreme complexity of designing applications to exploit in full the potential of such offerings. In this work, we describe the main challenges in delivering advanced, cross-platform, streaming analytics at scale with the primary goal of interactivity. We present the architecture of a prototype system designed from scratch to tackle such challenges and we describe the internals of its components. Finally, we discuss key added-value synergies upon serving target analytics workflows in real-world applications and present a preliminary performance evaluation.

Reference Format:
Antonios Deligiannakis, Nikos Giatrakos, Yannis Kotidis, Vasilis Samoladas, and Alkis Simitsis. Extreme-Scale Interactive Cross-Platform Streaming Analytics – The INFORE Approach. In the 2nd Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores (SEA Data 2021).

1 INTRODUCTION

Interactive analytics at scale lie at the core of many diverse applications. In life sciences, studying the effect of drug applications on simulated tumors may produce data streams of 100Gb/min [8]. These streams need to get analyzed online to interactively determine successive drug combinations. In the financial domain, NYSE alone generates several terabytes of stock data per day [6]. Interactive analytics enable timely reaction to investment opportunities or risks. In maritime surveillance, one needs to fuse heterogeneous, voluminous data composed of position streams of thousands of vessels, satellite images, thermal camera or acoustic signal streams of on-site devices to investigate ongoing illegal activities at sea [14].

Relevant data not only originate from a variety of heterogeneous sources but also often arrive at multiple, geo-dispersed clusters. In life sciences, several efforts aim at federating analytics across data centers[1]. Similarly, discovering dependencies or correlations among economies or industries involves global analysis of stock streams originating from various local market data centers. In maritime data analysis, streams come from a number of ground- or satellite-based receivers before being ingested in proximate data centers. Additionally, Big Data technologies are significantly fragmented. Each of these scenarios, involves delivering analytics over networked clusters each hosting a variety of Big Data platforms.

Typically, such scenarios are designed as streaming analytics workflows. Our goal is to provide a solution for executing global, cross-platform, streaming analytics workflows over a network of computer clusters hosting diverse Big Data technologies, with high interactivity; i.e., continuously deriving the output of the workflow in real-time and adapting to ad-hoc changes required due to observed results or system performance metrics, at runtime.

To this extent, streaming Big Data platforms such as Apache Spark, Flink, Kafka[2] approach the problem by ensuring horizontal scalability, i.e, by scaling out (parallelizing) the computation to a number of Virtual Machines (VM) and providing APIs to program distributed Big Data processing pipelines. These platforms are not designed to support cross-platform execution scenarios. They also provide none or suboptimal support for online Machine Learning (ML) or Data Mining (DM) operators, since the major ML APIs they provide do not focus on streaming data. Frameworks such as Apache Beam or Apache SAMOA [3], allow for creating code that is executable in different, supported Big Data platforms. Nevertheless, this does not amend the lack of resource management in multi-cluster and cross-platform settings.

Related systems such as Rheem [2], Ires [4], BigDawg [5], Musketeer [9] are designed towards cross-platform execution of global workflows, but they can only handle batch, instead of stream, processing pipelines. Even in systems like BigDawg or Rheem, which support stream processing in S-Store [13] and JavaStreams respectively, cross-streaming platform execution remains unsupported.

Our work aims at filling the gap in handling efficiently advanced, cross-platform, interactive analytics at scale. We describe the architecture of an open-source system, namely INFORE [4], designed from scratch for this purpose. A recent INFORE demo [7] (Best Demo

Published in the Proceedings of the 2nd Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores, co-located with VLDB 2021 (August 16-20, 2021, Copenhagen, Denmark) on CEUR-WS.org.

[1] https://elixir-europe.org/about-us, https://commonfund.nih.gov/bd2k/, https://cyverse.org/
[2] https://spark.apache.org/, https://flink.apache.org/, https://kafka.apache.org/
[3] https://beam.apache.org/, https://samoa.incubator.apache.org/
[4] https://bitbucket.org/infore_research_project/

Award at CIKM 2020) showcases how code-free, cross-platform analytics are feasible through user-friendly, graphical design and execution of streaming workflows. [7] presents the frontend, while in the current paper we focus on the backend. We detail its key features, three of its core components and inter-component synergies in real-world applications. We also present a preliminary performance evaluation.

2 OVERVIEW OF THE APPROACH

Requirements and Key Features. Based on our interaction with real-world practitioners and applications, early in INFORE's design, we identified three scalability requirements that should complement *interactivity* and *cross-streaming platform execution* features.

Enhanced Horizontal scalability. Parallelization is not adequate to handle extreme-scale scenarios. The number of available VMs in private clouds has a limit when it comes to extreme scale, while monetary charges arise for cloud providers.

Vertical scalability. Need to scale the computation with the number of processed streams. For instance, computing the correlation matrix of stock data streams results in a quadratic explosion in space and computational complexity, which is infeasible for thousands or millions of streams in our motivating scenarios.

Federated scalability. Need to scale the computation in settings where data arrive at multiple, geo-dispersed sites. In such settings, even when we do everything right within each cluster, the potential for interactivity is network bound [10].

Overview of Key Components. INFORE employs three core components for realizing the aforementioned key requirements.

Synopses Data Engine (SDE). There is a wide consensus in stream processing [1] that approximate but rapid answers to analytics tasks, more often than not, suffice. Synopses, such as samples, sketches, histograms [1], provide approximate answers, with accuracy guarantees, to popular (cardinality, frequency moment, correlation, set membership, quantile) analytic operators. The maintenance and use of data synopses is essential to stream processing applications because applications get only one look of the data. Synopses are used as an approximate version of the infinite data, which due to its small size can be processed and iterated upon, without the need of reading the entire, potentially infinite, data stream multiple times.

The SDE builds and maintains such synopses across platforms and clusters. It achieves enhanced horizontal scalability because synopses shed the load assigned to each VM in a cluster and reduce memory usage. For federated scalability, it reduces communication by transmitting compact data summaries. For interactivity, we add a unique SDE-as-a-Service (SDEaaS) design where the SDE is a constantly running service (job, in one or more clusters) that can accept any query, synopses creation on new streams or plugging code for new synopses at runtime, with zero downtime for running workflows. For vertical scalability, the SDEaaS design maintains thousands of synopses for thousands of streams, in settings where prior efforts [15, 16, 18, 19] can maintain only few tens of synopses. We henceforth use the terms SDE and SDEaaS interchangeably.

Online Machine Learning and Data Mining (OMLDM). OMLDM applies distributed, online ML and DM adopting a Parameter Server (PS) paradigm [12] (Figure 1(e)). The PS paradigm enhances horizontal and federated scalability via the option of an asynchronous (besides synchronous) synchronization policy to reduce the effect of stragglers and bandwidth consumption [12], respectively. Regarding interactivity, the OMLDM component allows to interactively adjust the number of learners and also to deploy the current up-to-date ML/DM model at the PS side, should concept drifts occur, at the very same time the training process in parallel learners proceeds. Established synergies with the SDE and the Optimizer components also boost vertical and federated scalability.

Optimizer Component. The Optimizer is a prerequisite to effectively use the SDE and the OMLDM Components towards interactivity. The Optimizer picks (a) the networked cluster, (b) the Big Data platforms available in that cluster, and (c) the provisioned resources for each SDE, OMLDM (or other) operator of the global workflow. A good solution is determined taking into account performance criteria and cost models, related to interactivity, engaging throughput, (network and processing) latency, memory usage etc. These components cooperate as follows (more in Section 4).

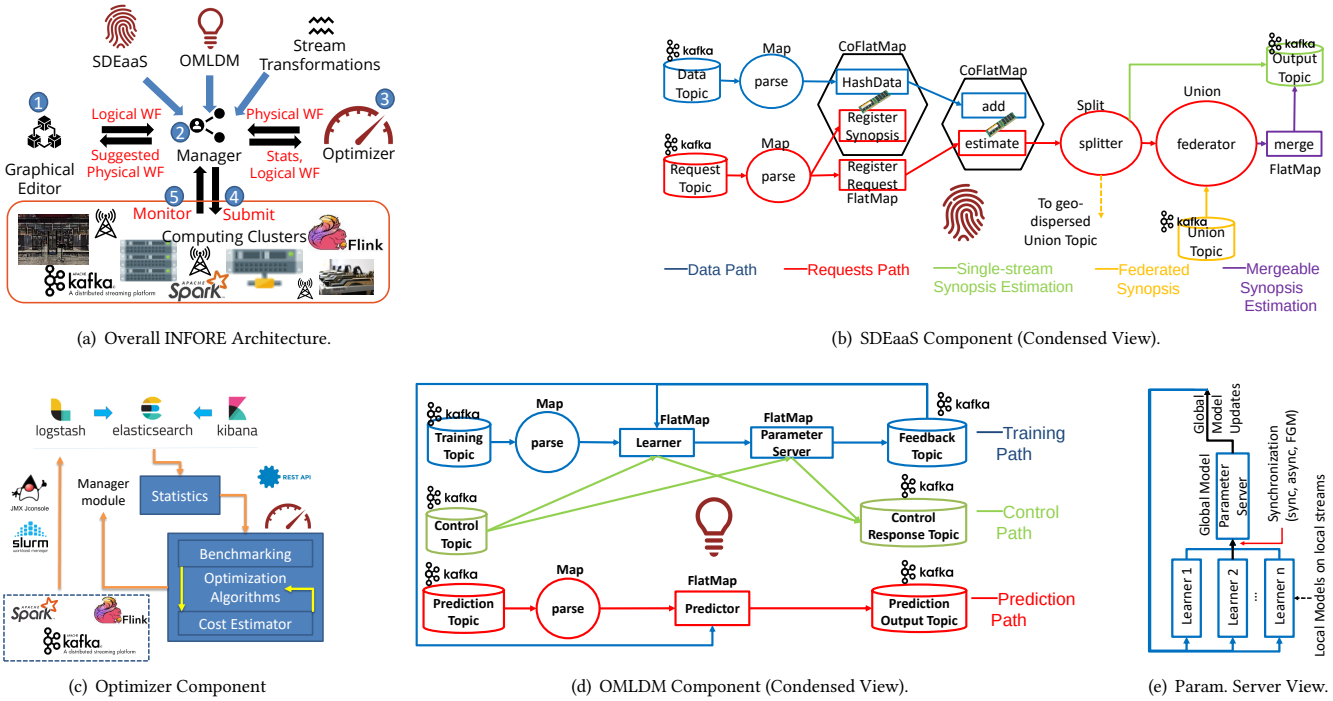
Synopses-based Optimization. Given a workflow, the Optimizer produces alternative physical execution plans where it replaces exact (e.g., count, window, correlation) operators with equivalent, but approximate ones from the SDE. It then provides suggestions for equivalent, but approximate, workflows accompanied by the expected interactivity (execution speed up) vs. accuracy trade-off.

Boosting Vertical Scalability. The SDE Component boosts vertical scalability of the OMLDM Component in ways that are not possible otherwise. Indicatively, we use Discrete Fourier Transform (DFT) and Locality Sensitive Hashing (LSH) bitmaps [11] to bucketize (hash) streams during correlation matrix computation, outlier detection, clustering or feature extraction. Vertical scalability is achieved by hashing streams to learners based on their DFT or LSH coefficients. The complexity of all these operators is reduced since computations are pruned for streams that do not end up nearby in the hashing space.

Pumped Parameter Server. With the help of the Optimizer, we extend the OMLDM Component with an additional synchronization policy, besides the synchronous and asynchronous communication entailed by the classic PS design [12]. We incorporate a novel communication protocol, termed Functional Geometric Monitoring (FGM) [17]. This protocol strengthens horizontal (within a cluster) and federated scalability by bridging the gap between synchronous and asynchronous communication. Instead of having learners communicating in predefined rounds/batches (synchronous) or when each one is updated (asynchronous), FGM requires communication only when a concept drift is likely to have occurred.

INFORE Architecture. We develop a prototype of the INFORE system incorporating operators from SDE, OMLDM and stream transformations provided by the streaming APIs of Big Data platforms such as Apache Flink or Spark. An overview of the INFORE architecture is illustrated in Figure 1(a). Currently, INFORE supports streaming analytics workflows over Apache Flink, Spark Structured Streaming and Kafka. Cross-(Big Data) platform and cluster communication is performed via JSON formatted Kafka messages. In a nutshell, INFORE works as follows:

(Step 1) The user designs in a Graphical Editor (left of Figure 1(a)) the desired workflow composed of stream transformation, OMLDM and SDE operators. These are logical operators composing the logical view of the workflow, independent of the underlying platforms.



(a) Overall INFORE Architecture.

(b) SDEaaS Component (Condensed View).

(c) Optimizer Component

(d) OMLDM Component (Condensed View).

(e) Param. Server View.

Figure 1: Overall INFORE Architecture & Internal Architecture of Key Components

(Step 2) When the designed workflow is submitted, a JSON file is conveyed to a Manager module (middle of Figure 1(a)). Initially, the Manager sends the JSON to the Optimizer (right of Figure 1(a)).

(Step 3) The Optimizer determines the physical view of the workflow, performing a mapping of the logical operators to physical ones to be executed on specific network clusters and Big Data platforms, so that various performance criteria are optimized.

(Step 4) The Optimizer returns a JSON with physical operators to the Manager which compiles *.jar* files deploying stream transformation, OMLDM and SDE operators (top of Figure 1(a)) to be submitted at chosen (cluster, Big Data platform) pairs.

(Step 5) The submitted jobs are monitored and performance statistics are collected.

3 KEY-COMPONENT INTERNALS

This section elaborates on INFORE's key components. For space considerations, we describe the concepts using Flink as an example implementation, but the design is generic to support operators in all streaming platforms mentioned above.

3.1 SDE Component

The architecture of the SDE Component is illustrated in Figure 1(b). In streaming settings, multiple workflows may run continuously providing output streams to application interfaces for timely decision making. The SDEaaS design we adopt (see Section 2) can serve multiple wofklows, running concurrently and continuously, with approximate, commonly used operators (e.g., counts, sums, frequency moments) reusing, instead of duplicating, streams and synopses common to workflows. Moreover, for every new synopsis maintenance request, SDEaaS reserves new execution tasks for the operators in Figure 1(b), while without SDEaaS we would reserve entire threads (task slots in terms of Flink). In that, SDEaaS manages thousands of synopsis, on-demand, for thousands of streams; hence, ensuring vertical scalability [11].

To briefly showcase the performance benefits of our approach, Figure 3(a) shows the results of an experiment over a cluster with 40 task slots. (The hardware details are not important for this experiment.) We compare SDEaaS to a non-SDEaaS approach, such as Proteus [16], that extends Flink with data synopsis utilities, but lacks the SDEaaS paradigm. We design an experiment where we start with maintaining 2 Count-Min sketches for frequency estimations on the volume, price pairs of financial data (stocks). Then, we express demands for maintaining more Count-Min sketches for up to 5000 sketches/stocks. We do that without stopping the already running synopses. Figure 3(a) shows the sum of throughputs of all running jobs for non-SDEaaS and the throughput of SDEaaS. non-SDEaaS starts new jobs for new synopses and assigns at least one entire task slot to each new job. Hence, the 40 task slots in our cluster are depleted and ✘ signs in Figure 3(a) denote that non-SDEaaS cannot maintain more than 40 synopses. SDEaaS has no such limit since it assigns new tasks to a single running service (job) at runtime, with zero downtime. SDEaaS also exhibits higher throughput than non-SDEaaS due to fine-grained resource utilization at the task, instead of task slot, level.

The extensible SDE Library currently includes a variety of synopses techniques for cardinality, frequency moment, correlation,

set membership or quantile estimation [1]. Please check [11] for further details. The SDE API allows on-the-fly requests for building/stopping synopses, plug-in code for new synopses in the SDE Library, ad-hoc and continuous queries. To enable the SDEaaS design for interactivity and vertical scalability, the SDE Library is built using subtype polymorphism and inheritance in Java. This is the key to allow new synopsis code to be loaded at runtime and also allows all requests to be issued on-the-fly by the Manager module, instead of deploying separate jobs. Every new synopsis simply extends a `Synopsis` class with its detailed algorithms for `add` (streaming updates), `estimate` (query answering) and `merge` for outputting an estimation for mergeable synopses [1] kept at multiple workers or clusters.

When a synopsis operator of the SDE is included in a workflow, every request arriving via the `RequestTopic` in Figure 1(b) for building or querying a running synopsis follows the red-colored path. The `RegisterSynopsis`, `RegisterRequest` `FlatMaps` initially produce and then look up the same keys pointing to workers that should maintain the synopsis, but for a different purpose each. `RegisterSynopsis` uses these keys to direct streaming tuples, which follow the blue-colored path, to assigned workers to update the synopsis. `Register Request` uses the same keys to direct queries to those workers and derive estimations at the `estimation` `FlatMap` by reading the status of the synopsis kept at the `add` `FlatMap`.

When a streaming update arrives at the Kafka `DataTopic`, the `FlatMap` termed `HashData` looks up the keys of workers produced by `RegisterSynopsis` and directs the update to the proper workers. There, the `add` `FlatMap` includes the update in the maintained synopsis. For instance, in case a FM sketch is maintained[1], the `add` operation hashes the incoming tuple to a position of the maintained bitmap and turns the corresponding bit to 1 if it is not already set. Upon a query, the `estimation` `FlatMap` uses the lowest set bit of that bitmap to derive a cardinality estimation[1].

The rest of the operators on the right-hand side of Figure 1(b) are used for appropriately controlling the output topics. The `splitter` (`split` operator in Flink) directs the estimation to the `OutputTopic` if a synopsis is defined on a single stream handled by one worker (green path). If a synopses engages many streams (e.g. many stocks) maintained at multiple workers (purple path) or multiple clusters (yellow path – clusters communicate via the `UnionTopic` of Kafka) `merge` `FlatMap` synthesizes the estimations (for instance, executing a logical disjunction operation on various FM sketches [1]) before outputting the estimation via `OutputTopic`.

3.2 OMLDM Component

The OMLDM component fills the gap of existing batch processing-focused APIs such as Spark's MLlib or FlinkML. Its internal architecture is illustrated in Figure 1(d). Two types of pipelines are engaged in the OMLDM architecture: *training* and *prediction* pipelines.

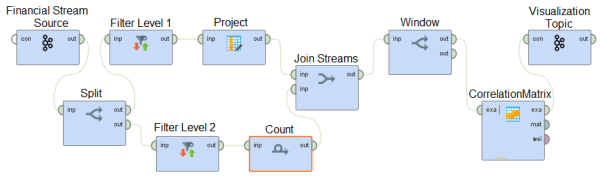
Training pipelines. Training data streams follow the blue-colored path of Figure 1(d). They are ingested via the `TrainingTopic`. The rest of the training pipeline is implemented by a pair of `Learner` and `ParameterServer` `FlatMaps`. We follow a Parameter Server (PS) distributed model (Figure 1(e)), where an interactively defined number of identical learners distribute amongst themselves the computational load of training on an incoming data stream of training samples. In a nutshell, PS holds and updates the state of the learning process, including the current version of the trained model. Learners compute the updates to the model and coordinate via the PS. This is because the iterative nature of ML/DM computations requires communication between all parallel workers, and this is done by the PS in each pipeline. Training pipelines are data sinks, that is, they do not generate an output stream. However, they do have a `Feedback` Kafka topic to support iteration required by involved tasks. The `Feedback` topic is also read by the predictors of the red path, discussed below, to update their models upon a concept drift. For instance, should training on labeled streams of a classification task take place on par with prediction (tagging unlabeled streams), in case of a concept drift, the PS sends a new model to predictors via the `FeedbackTopic`.

For training pipelines, it is the pipeline's responsibility to coordinate the processing between the learners and the PS. As noted in Section 2, the supported coordination policies include synchronous, asynchronous and FGM, a novel and bandwidth-efficient coordination policy that we introduced at [17]. Performance-wise, the synchronous policy does not encourage enhanced horizontal scalability because when many learners are used, the total utilization is usually low, should only few stragglers exist. The asynchronous one is the policy of choice in large-scale ML; the processing speed is much higher when many learners are used, and therefore training is more scalable. The FGM policy provides both enhanced horizontal and federated scalability because (a) the learners do not need to await each other's sync in rounds or contact the PS immediately with each update, (b) communication takes place asynchronously, but only when a concept drift may have occurred. This is achieved in synergy with the Optimizer (see Section 4).

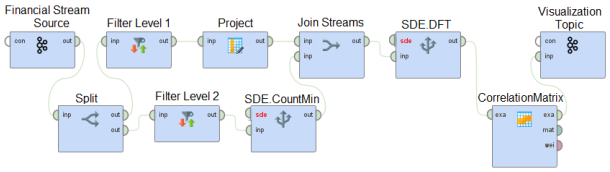
Figure 1(d) shows an instance of the PS and learners running on a Flink cluster. The messaging interface between `Learner` and the PS is handled via Kafka, and hence it is generic and independent of the underlying Big Data platform.

Prediction pipelines. These are shown with the red-colored path of Figure 1(d). Prediction pipelines do not train the models they apply; these models are loaded at `Predictor` `FlatMap` via control messages. A model can be changed interactively by the PS at runtime, if there has been a concept drift, via the `FeedbackTopic`. Prediction pipelines employ some model for classification, interpolation or inference and transform input to output streams, which can be processed further by downstream operators of a global workflow. The control topics (green arrows and Kafka topics in Figure 1(d)) deliver control messages to and from an OMLDM job. The control API understands Create/Read/Update/Delete (CRUD) commands for each type of resource. For instance, using control messages the OMLDM component will respond with the current status of the PS and learners (e.g. how many learners are currently running) or the currently up-to-date model at the PS.

OMLDM currently supports (i) classification: Passive Aggressive Classifier, Online Support Vector Machines and Vertical Hoeffding Trees, (ii) clustering/outlier detection: BIRCH, Online k-Means and StreamKM++ and (iii) regression: Passive Aggressive Regressor, Online Ridge and Polynomial Regression, Autoregressive Integrated Moving Average. Facilities for correlation matrix computation, standardization, polynomial feature extraction are also available.



(a) Initial Workflow at the Graphical Editor.



(b) Suggested Workflow via Synopses-based Optimization.

Figure 2: Initial and transformed workflow using Synopses-based Optimization.

3.3 Optimizer Component

Given a workflow composed of various operators, we have to properly tune their execution prescribing both the Big Data Platform and the networked computer cluster that achieve good overall performance with respect to interactivity-related metrics are met.

The internal architecture of the INFORE Optimizer is shown in Figure 1(c). There are a number of submodules involved before providing logical to physical operator mappings to be deployed for execution. First, we use a statistics collector to derive performance measurements from each executed workflow. Statistics are collected via JMX or Slurm[5] and are ingested in an ELK stack[6] while monitoring jobs.

We have developed a Benchmarking submodule that automates the acquisition of performance metrics for every supported operator run on different Big Data platforms. The Benchmarking submodule circumvents steps 1-3 in Figure 1(a) and enables direct job submision to the underlying clusters engaging supported operators and stream sources. In that, the execution of various operators and (part of) workflows on different (Big Data platform, cluster) pairs is a priori benchmarked, statistics are collected and performance models are built to avoid a cold start.

Cost models are derived with a Bayesian Optimization approach inspired by CherryPick [3]. From the statistics we have collected from either micro-benchmarks and/or actual workflow execution, we form a Gaussian performance/cost prediction model for the various operators and (parts of) workflows executed in different (Big Data platform, cluster) pairs. The cost model is incrementally improved with new workflow executions.

Algorithmically, the Optimizer solves a multi-criteria optimization problem with objectives involving throughput, communication cost, processing and network latency, CPU/GPU and memory usage as well as accuracy for synopses-based optimization purposes. Due to conflicting objectives, we cannot optimize them all simultaneously. Therefore, we resort to Pareto optimal solutions obtained by maximizing a weighted combination of these objectives under clusters' capacity constraints.

Notably, no prior effort [2, 4, 5, 9] examined such a rich set of optimization objectives related to interactivity. Indicatively, only [2] considers network performance metrics (but for batch processing scenarios). To derive Pareto optimal solutions, besides a baseline, Exhaustive Search algorithm that examines every possible mapping of logical to physical operators in a workflow, we have developed a novel variation of an A*-alike algorithm. The classic A* algorithm cannot work in our setup as is, because it can only suggest optimal paths. This would mean that some logical operators would be left out from suggesting a physical operator mapping. Our novel A* variation, fosters equivalent heuristics to the typical A* to prune the search space, but outputs a graph (instead of a path) with physical operators instantiating every logical one in the submitted workflow. This new algorithm can speed up the suggestion of the optimal physical workflow by orders of magnitude (see Section 4). We have also developed a Greedy and a Heuristic algorithm that leverage parallelism and domain knowledge to boost the performance further and to enable adaptivity (i.e., provide in-real time and incrementally a new plan to adapt to ad hoc performance hiccups). All algorithms employ the cost estimation models to evaluate alternative physical operators while exploring the space of alternative mappings.

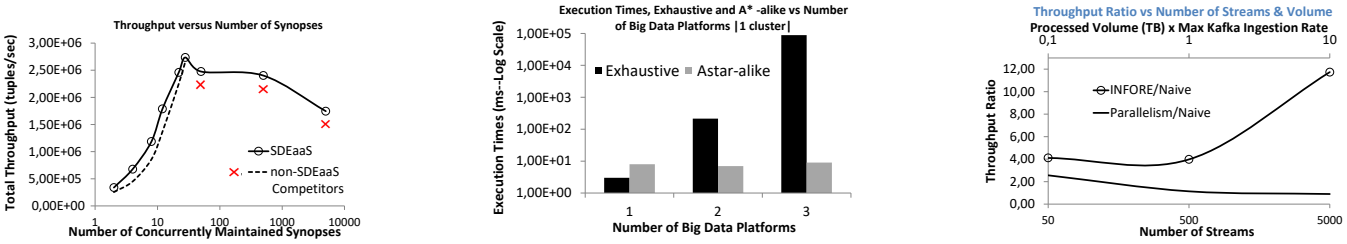
4 SYNERGIES IN A REAL CASE STUDY

This section describes how our prototype works in a real use case scenario from the financial domain and how advanced synergies are established among individual components. For exhibition purposes, we employ a single cluster equipped with Flink and Spark.

The logical workflow submitted to the Optimizer is illustrated in Figure 2(a). The workflow of Figure 2(a) utilizes Level 1, Level 2 stock data[7] to discover highly positively/negatively correlated pairs of stocks. In Figure 2(a), data arrive at a Kafka source. The `Split` operator separates Level 1 from Level 2 data. It directs Level 2 data to the bottom branch of the workflow. There, the Level 2 bids are `Filtered` (i.e., for monitoring only a subset of stocks). The bids per stock are counted using a `Count` logical operator. When a trade for a stock is realized, a new Level 1 tuple is directed by `Split` to the upper part of the workflow. A `Project` operator keeps only the timestamp of the trade for each stock. The `Join` operator joins the stock trade, Level 1 tuple with the count of bids the stock received until the trade. The result is inserted in a time `Window` of recent such counts, forming a time series. Finally, the `CorrelationMatrix` of stocks' time series is computed by that OMLDM operator.

Synopses-based Optimization. When the workflow is submitted, for every logical operator, there are two platform choices, Flink or Spark. The Optimizer holds a dictionary to determine the equivalence of logical operators to physical ones in either platform. In Synopses-based Optimization, the Optimizer attempts to boost interactivity by treating the SDE as an additional platform. That is, its dictionary also holds mappings about which exact operators could be replaced by approximate ones of the SDE. Therefore, the

[5] https://docs.oracle.com/javase/tutorial/jmx/overview/, slurm.schedmd.com/
[6] https://www.elastic.co/what-is/elk-stack
[7] www.investopedia.com/terms/l/level1.asp,www.investopedia.com/terms/l/level2.asp



(a) SDEaaS vs non-SDEaaS.

(b) Optimizer Execution Time.

(c) Horizontal & Vertical Scalability.

Figure 3: Performance Evaluation.

Optimizer will finally suggest an execution plan as shown in Figure 2(b). In this plan the `Count` operator has been replaced by a `SDE.CountMin` sketch operator of the SDE and the `Window` operator has been replaced by `SDE.DFT`. All physical operators are prescribed to be executed over Flink, i.e., where the SDE service runs.

Boosting Vertical Scalability. The `SDE.DFT` operator of the SDE, for every window it approximates, outputs a pair of values: a bucketID and a vector of reduced dimensionality compared to the original window. The bucketID is based on the first few coefficients of the DFT [11] and is used as the key to hash stock streams to learners of the `CorrelationMatrix` OMLDM operator (Figure 2(b)). Reduced vector correlations are pruned for stocks hashed far away in the hashing space since these stock streams are guaranteed to be highly (dis)similar [11]. This boosts vertical scalability for a task of quadratic complexity to the number of input streams.

Pumped Parameter Server. The stream entering the `Correlation Matrix` operator of OMLDM is duplicated to the training and prediction pipelines (Figure 1(d)). Learners (buckets in our previous discussion) compute the pairwise correlations of the streams they receive (correlation coefficient and the beta regression parameter coincide for standardized vectors), while the PS holds the current up-to-date correlation matrix which is essentially the global model at the predictor and the PS sides. A concept drift for the global model would occur if the Frobenius norm measuring the difference between the matrix kept at PS and the true correlation matrix may have exceeded a threshold. The Optimizer will decompose this constraint to local ones installed at each learner. Mathematical details are described in [17], but in a simple version of FGM, the Optimizer will prescribe that learners will sync with the PS when a signed distance value computed locally at each learner, becomes positive. Notice that local constraints are determined by the Optimizer which initially prescribes and then adapts the learners at runtime.

Preliminary Experimental Evaluation. We employ a Kafka cluster with 4 Dell PowerEdge R320 Intel Xeon E5-2430 v2 2.50GHz machines with 32GB RAM and a cluster with 10 Dell PowerEdge R300 Quad Core Xeon X3323 2.5GHz machines with 8GB RAM each, for Flink and Spark Structured Streaming. We use real Level 1, Level 2 stock data (~5000 stocks/~10 TB) provided by a European financial company. We set DFT to use 8 coefficients, CountMin ($\epsilon = 0.002, \delta = 0.01$), we use 0.9 correlation threshold and a time window of 5 minutes, based on input from the data provider. Figure 3(b) shows the time it takes to the two optimization algorithms (Exhaustive and A*–alike) to produce a physical workflow from the time they receive the logical one, when 1 (Flink), 2 (Spark,Flink) or 3 (Spark, Flink, Synopses-based Optimization) are available. As the figure demonstrates for more than 1 platform, our novel A*–alike algorithm reduces the time to prescribe a physical workflow by 1 to 4 orders of magnitude. For 1 platform (still the Optimizer prescribes provisioned resources) the Exhaustive Search Algorithm is faster, since A*–alike requires more initialization time.

In Figure 3(c) we show the scalability of the INFORE approach versus a technique that executes the workflow of Figure 2(a) in Flink (denoted by "Parallelism"), without synopses-based optimization, boosted vertical scalability or the OMLDM Component. INFORE executes the workflow of Figure 2(b). We plot the throughput ratio of INFORE and "Parallelism" over a Naive approach that executes the same workflow with parallelism set to 1. The two horizontal axes in the figure show the number of processed stock streams (bottom) and the volume in terabytes (top) ingested using the maximum Kafka rate. Thus, the top axis accounts for horizontal scalability (volume, speed), while the bottom axis accounts for vertical scalability.

When we monitor 500 streams, i.e., 250K/2 cells in the correlation matrix, the fact that "Parallelism" lacks synopses-based optimization and vertical scalability makes its throughput ratio approaching 1, i.e. the Naive approach. INFORE's performance remains steady when switching from 50 to 500 streams improving "Parallelism" by 3 times. The most important findings come upon switching to 5000 stocks (25M/2 cells in the matrix). Figure 3(c) denotes that because of the lack of enhanced horizontal and vertical scalability, "Parallelism" becomes equivalent to the Naive one. INFORE exhibits more than an order of magnitude improved throughput with the trade-off of highly correlated pairs being identified with 90% accuracy.

For federated scalability, we measure communicated bytes among the operators. Due to space constraints, we only report here that INFORE can reduce communication by more than an order of magnitude when 5000 stock streams are being monitored.

5 CONCLUSIONS & ONGOING WORK

We have presented the design, internal structure, and integration aspects of key components of INFORE, a prototype designed for interactive analytics at scale. To our knowledge, INFORE is the first system for cross-platform, streaming analytics. We currently enrich the SDE and OMLDM libraries to strengthen the support for the three scalability types and enhance analytics capabilities.

ACKNOWLEDGMENTS

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

REFERENCES

[1] 2016. Data Stream Management - Processing High-Speed Data Streams. Springer. https://doi.org/10.1007/978-3-540-28608-0

[2] Divy Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed K. Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, Saravanan Thirumuruganathan, and Anis Troudi. 2018. RHEEM: Enabling Cross-Platform Data Processing - May The Big Data Be With You! -. *Proc. VLDB Endow.* 11, 11 (2018), 1414–1427. https://doi.org/10.14778/3236187.3236195

[3] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, Aditya Akella and Jon Howell (Eds.). USENIX Association, 469–482. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard

[4] Katerina Doka, Nikolaos Papailiou, Dimitrios Tsoumakos, Christos Mantas, and Nectarios Koziris. 2015. IReS: Intelligent, Multi-Engine Resource Scheduler for Big Data Analytics Workflows. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 1451–1456. https://doi.org/10.1145/2723372.2735377

[5] Aaron J. Elmore, Jennie Duggan, Mike Stonebraker, Magdalena Balazinska, Ugur Çetintemel, Vijay Gadepally, Jeffrey Heer, Bill Howe, Jeremy Kepner, Tim Kraska, Samuel Madden, David Maier, Timothy G. Mattson, Stavros Papadopoulos, Jeff Parkhurst, Nesime Tatbul, Manasi Vartak, and Stan Zdonik. 2015. A Demonstration of the BigDAWG Polystore System. *Proc. VLDB Endow.* 8, 12 (2015), 1908–1911. https://doi.org/10.14778/2824032.2824098

[6] Forbes. 2021 (last accessed). https://www.forbes.com/sites/tomgroenfeldt/2013/02/14/at-nyse-the-data-deluge-overwhelms-traditional-databases/#362df2415aab.

[7] 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*. ACM, 3389–3392. https://doi.org/10.1145/3340531.3417435

[8] Nikos Giatrakos, Nikos Katzouris, Antonios Deligiannakis, Alexander Artikis, Minos N. Garofalakis, George Paliouras, Holger Arndt, Raffaele Grasso, Ralf Klinkenberg, Miguel Ponce de Leon, Gian Gaetano Tartaglia, Alfonso Valencia, and Dimitrios Zissis. 2019. Interactive Extreme: Scale Analytics Towards Battling Cancer. *IEEE Technol. Soc. Mag.* 38, 2 (2019), 54–61. https://doi.org/10.1109/MTS.2019.2913071

[9] Ionel Gog, Malte Schwarzkopf, Natacha Crooks, Matthew P. Grosvenor, Allen Clement, and Steven Hand. 2015. Musketeer: all for one, one for all in data processing systems. In *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*, Laurent Réveillère, Tim Harris, and Maurice Herlihy (Eds.). ACM, 2:1–2:16. https://doi.org/10.1145/2741948.2741968

[10] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. 2018. Benchmarking Distributed Stream Data Processing Systems. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 1507–1518. https://doi.org/10.1109/ICDE.2018.00169

[11] Antonis Kontaxakis, Nikos Giatrakos, and Antonios Deligiannakis. 2020. A Synopses Data Engine for Interactive Extreme-Scale Analytics. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. ACM, 2085–2088. https://doi.org/10.1145/3340531.3412154

[12] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, Jason Flinn and Hank Levy (Eds.). USENIX Association, 583–598. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[13] John Meehan, Stan Zdonik, Shaobo Tian, Yulong Tian, Nesime Tatbul, Adam Dziedzic, and Aaron J. Elmore. 2016. Integrating real-time and batch processing in a polystore. In *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13-15, 2016*. IEEE, 1–7. https://doi.org/10.1109/HPEC.2016.7761585

[14] Aristides Milios, Konstantina Bereta, Konstantinos Chatzikokolakis, Dimitris Zissis, and Stan Matwin. 2019. Automatic Fusion of Satellite Imagery and AIS data for Vessel Detection. In *22th International Conference on Information Fusion, FUSION 2019, Ottawa, ON, Canada, July 2-5, 2019*. IEEE, 1–5. https://ieeexplore.ieee.org/document/9011339

[15] Barzan Mozafari. 2019. SnappyData. In *Encyclopedia of Big Data Technologies*, Sherif Sakr and Albert Y. Zomaya (Eds.). Springer. https://doi.org/10.1007/978-3-319-63962-8_258-1

[16] Proteus Project. 2021 (last accessed). https://github.com/proteus-h2020.

[17] Vasilis Samoladas and Minos N. Garofalakis. 2019. Functional Geometric Monitoring for Distributed Streams. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*. OpenProceedings.org, 85–96. https://doi.org/10.5441/002/edbt.2019.09

[18] Stream-lib. 2021 (last accessed). https://github.com/addthis/stream-lib.

[19] Yahoo DataSketch. 2021 (last accessed). https://datasketches.github.io/.