A Distributed Online Learning Approach for Pattern Prediction over Movement Event Streams with Apache Flink - v2.0

Ehab Qadah
Fraunhofer IAIS
Sankt Augustin, Germany
Ehab.Qadah@iais.fraunhofer.de

Michael Mock
Fraunhofer IAIS
Sankt Augustin, Germany
Michael.Mock@iais.fraunhofer.de

Elias Alevizos
NCSR "Demokritos"
Athens, Greece
alevizos.elias@iit.demokritos.gr

Georg Fuchs
Fraunhofer IAIS
Sankt Augustin, Germany
Georg.Fuchs@iais.fraunhofer.de

ABSTRACT

In this paper, we present a distributed online prediction system for user-defined patterns over multiple massive streams of movement events, built using the general purpose stream processing framework Apache Flink. The proposed approach is based on combing probabilistic event pattern prediction models on multiple predictor nodes with a distributed online learning protocol in order to continuously learn a global prediction model parameters to share it among the predictors in a communication-efficient way. Our approach enables the collaborative learning between the predictors (i.e., "learn from each other"), thus the learning rate is accelerated with less data. The underlying model provides an online predictions when a pattern (i.e., a regular expression over the event types) will be completed within each event stream. We describe the distributed architecture of the proposed system, its implementation in Flink, and present experimental results over real-world event streams related to trajectories of moving vessels.

KEYWORDS

Big Movement Event Streams, Stream Processing, Event Pattern Prediction, Distributed Pattern Markov Chain.

1 INTRODUCTION

In recent years, the technological advances have led to a growing in the availability of massive amounts of continuous streaming data (i.e., data streams observing events) in many application domains such as social networks [22], Internet of Things (IoT) [23], and maritime surveillance [30]. The ability to detect and predict the full matches of a pattern of interest (e.g., certain sequence of events), defined by a domain expert, is important for several operational decision making tasks.

An event stream is an unbounded collection of timely ordered data observations in the form of an attribute tuple that is composed of a value from finite event types along with other categorical and numerical attributes. In this work we deal with movement event streams, for instance, in the context of maritime surveillance, the event stream of a moving vessel consists of spatial-temporal and kinematic information along with the vessel's identification and its trajectory related events, which is based on the automatic identification system (AIS) [27] messages that continuously sent by the vessel. Therefore, leveraging event patterns prediction over the real-time tracking streams of moving vessels is useful to alert maritime operation mangers about suspicious activities (e.g., fast sailing vessels near ports, or illegal fishing) before they happen. However, processing real-time streaming data with low latency is challenging since data streams are large and distributed in nature and continuously keep on coming at a high rate.

In this paper, we present a design and implementation of an online, distributed and scalable pattern prediction system over multiple massive streams of events. More precisely we consider event streams related to trajectories of moving objects (i.e., vessels). The proposed approach is based on a novel method that combines the distributed online prediction protocol [8, 15] with the event forecasting with Pattern Markov Chain model [2], implemented on top of the Big Data framework for stream processing Apache Flink [13]. We evaluate our proposed system over real-word data streams of moving vessels, which are provided in the context of the datAcron project[1].

The rest of the paper is organized as follows. We discuss the related work and used frameworks in Section 2. In Section 3, we describe the problem of pattern prediction, our proposed approach, and the architecture of our system. The implementation details on top of Flink are presented in Section 4 and the experimental results in Section 5. We conclude in Section 6.

[1] http://www.datacron-project.eu/

2 RELATED WORK AND BACKGROUND

2.1 Related work

Pattern prediction over event streams. The task of forecasting over time-evolving streams of data can be formulated in various different ways and with varying assumptions. One of the most usual ways to formalize this task is to assume that the stream is a time-series of numerical values and the goal is to forecast at each time point n the values at some future points $n + 1$, $n + 2$, etc., (or even the output of some function of future values). This is the task of time-series forecasting [24]. Another way to formalize this task is to view streams as sequences of events, i.e., tuples with multiple, possibly categorical, attributes, like *event type*, *timestamp*, etc., and the target is to predict future events or patterns of events. In this work that we present here, we focus on this latter definition of forecasting (event pattern forecasting).

A substantial body of work on event forecasting comes from the field of temporal pattern mining where events are defined as 2-tuples of the form (*EventType*, *Timestamp*). The ultimate goal is to extract patterns of events in the form either of association rules [1] or frequent episode rules [21]. These methods have been extended in order to be able to learn not only rules for detecting event patterns but also rules for predicting events. For example, in [31], a variant of association rule mining is where the goal is to extract sets of event types that frequently lead to a rare, target event within a temporal window.

In [18], a probabilistic model is presented in order to calculate the probability of the immediately next event in the stream. This is achieved by using standard frequent episode discovery algorithms and combining them with Hidden Markov Models and mixture models. The framework of episode rules is employed in [9] as well. The output of the proposed algorithms is a set of predictive rules whose antecedent is minimal (in number of events) and temporally distant from the consequent. In [34] a set of algorithms is proposed that target batch, online mining of sequential patterns, without maintaining exact frequency counts. As the stream is consumed, the learned patterns can be used to test whether a prefix matches the last events seen in the stream, indicating a possibility of occurrence for events that belong to the suffix of the rule.

Event forecasting has also attracted some attention from the filed of Complex Event Processing (see [7] for a review). One such early approach is presented in [25]. Complex event patterns are converted to automata and subsequently Markov chains are used in order to estimate when a pattern is expected to be fully matched. A similar approach is presented in [2], where again automata and Markov chains are employed in order to provide (future) time intervals during which a match is expected with a probability above a confidence threshold.

Distributed Online Learning. In recent years, the problem of distributed online learning has received significant attention and have been studied in [8, 15, 17, 32, 33]. A distributed online mini-batch prediction approach over multiple data streams has been proposed in [8]. Their approach is based on a static synchronization method, the learners periodically communicate their local models with a central coordinator unit after consuming a fixed number of input samples/events (i.e., batch size b), in order to create a global model and share it between all learners. This work has been extended in [15] by introducing a dynamic synchronization scheme that reduces the required communication by making the local learners communicate their models only if they diverge form a reference model. In this work, we aim to employee this protocol with event patterns prediction models over multiple event streams.

2.2 Technological Background

In the last years, many systems for large-scale and distributed stream processing have been proposed, including Spark Streaming [12], Apache Storm [14], and Apache Flink [13]. These framework allow to ingest and process real-time data streams from different published on distributed message queuing platforms, such as Apache Kafka [11], or Amazon Kinesis [5]. In this work, we implemented the proposed system over Apache Flink that provides the distributed stream processing components of the distributed event pattern predictors, alongside Apache kafka for streaming the input event streams, and as a messaging platform to enable the distributed online learning functionalities.

In the datAcron project, the Flink streaming processing engine has been chosen as a primary platform for supporting the streaming operations, based on an internal comparative evaluation of several streaming platforms. Thus in this work we used it to implement our system. Although the distributed online learning framework has already been implemented in the FERARI project [10] based on Apache Storm.

Apache Flink. Apache Flink is an open source project that provides a large-scale, distributed, and stateful stream processing platform [6]. Flink is one of the recent and pioneer big data processing frameworks, it employees data-stream processing model for streaming and batch data, the batch processing is treated as a special case of streaming applications (i.e., finite stream). The Flink's software stack includes the *DataStream* and *DataSet* APIs for processing infinite and finite data, respectively. These two core APIs built on the top of the Flink's core distributed streaming dataflow engine, and provide operations on data streams or sets such as mapping, filtering, grouping, etc.

The main data abstractions of Flink are *DataStream* and *DataSet* that represent read-only collection of data elements. The list of elements is bounded (i.e., finite) in *DataSet*, while it is unbounded (i.e., infinite) in the case of *DataStream*. The Flink's core is a distributed streaming dataflow engine, with each Flink program is represented by a data-flow graph (i.e., directed acyclic graph - DAG) that executed by the Flink's engine [6]. The data flow graphs are composed of stateful operators and intermediate data stream partitions. The execution of each operator is handled by multiple parallel instances based on the *parallelism* level. While each parallel operator instance is executed in an independent task slot on a machine within a cluster of computers [13].

Apache Kafka. Apache Kafka is scalable, fault-tolerant, and distributed streaming framework/messaging system [11]. It allows to publish and subscribe to arbitrary data streams, which are managed in different categories (i.e., *topics*), and partitioned

in the Kafka cluster. The Kafka Producer API provides the ability to publish a stream of messages to a topic. These messages can be consumed by applications using the Consumer API that allows them to read the published data stream in the Kafka cluster, in addition, the messages stream is distribute and load balance between the multiple receivers within the same consumer group for the sake of scalability.

3 SYSTEM OVERVIEW

3.1 Pattern prediction on a single stream

For our work presented in this paper, we use the approach described in [2]. For the sake of self-containment, we briefly describe this approach in what follows, first assuming that only a single stream is consumed and then adjusting for the case of multiple streams. We follow the terminology of [3, 20, 35] to formalize the problem we tackle.

3.1.1 Problem formulation. We first give the definition for an input event and for a stream of input events as follows:

Definition 3.1. Each event is defined as a tuple of attributes $e_i = (id, type, \tau, a_1, a_2 ....., a_n)$, where *type* is the event type attribute that takes a value from a set of finite event types/symbols Σ, τ represents the time when the event tuple was created, the $a_1, a_2, ..., a_n$ are spatial or other contextual features (e.g., speed); these features are varying from one application domain to another. The attribute *id* is a unique identifier that connects the event tuple to an associated domain object.

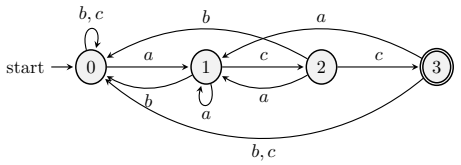
Definition 3.2. A stream $s = \langle e_1, e_3, ..., e_t, ... \rangle$ is a time-ordered sequence of events.

A user-defined pattern $\mathcal{P}$ is given in the form of a regular expression (i.e., using operators for *sequence*, *disjunction*, and *iteration*) over Σ (i.e., event types) [2]. More formally, a pattern is given through the following grammar:

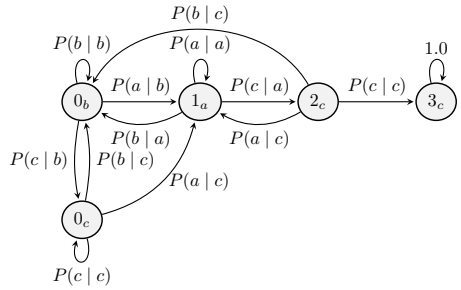
Definition 3.3. $\mathcal{P} := E \mid \mathcal{P}_1; \mathcal{P}_2 \mid \mathcal{P}_1 \vee \mathcal{P}_2 \mid \mathcal{P}_1^*$, where $E \in \Sigma$ is a constant event type, ; stands for sequence, $\vee$ for disjunction and $*$ for *Kleene* $-*$. The pattern $\mathcal{P} := E$ is matched by reading an event e_i iff $e_i.type = E$. The other cases are matched as in standard automata theory.

The problem at hand may then be stated as follows: given a stream s of low-level events and a pattern $\mathcal{P}$, the goal is to estimate at each new event arrival the number of future events that we will need to wait for until the pattern is satisfied (and therefore a match be detected).

3.1.2 Proposed approach. As a first step, event patterns are converted to deterministic finite automata (DFA) through standard conversion algorithms. As an example, see Figure 1a for the DFA of the simple sequential pattern $\mathcal{P} = a; c; c$ and an alphabet $\Sigma = \{a, b, c\}$ (note that the DFA has no dead states since we need to handle streams and not strings). The next step is to derive a Markov chain that will be able to provide a probabilistic description of the DFA's run-time behavior. Towards this goal, we use Pattern Markov Chains, as was proposed in [26]. Under the assumption that the input events are independent and identically distributed (i.i.d.), it can be shown that there is a direct mapping of the states of the DFA to states of a Markov chain and the transitions of the DFA to transitions of the Markov chain. The transition probabilities of the Markov chain are the occurrence probabilities of the various event types. On the other hand, if the occurrence probabilities of the events are dependent on some of the previous events seen in the stream (i.e., the stream is generated by an m^{th} order Markov process), we might need to perform a more complex transformation (see [26] for details) in order to obtain a "proper" Markov chain. The transition probabilities are then conditional probabilities on the event types. In any case, we call such a derived Markov chain a Pattern Markov Chain (PMC) of order m and denote by $PMC_{\mathcal{P}}^m$, where $\mathcal{P}$ is the initial pattern and m the assumed order. As an example, see Figure 1b, which depicts the PMC of order 1 for the generated DFA of Figure 1a.



(a) $DFA_{\Sigma^*.\mathcal{P}}$



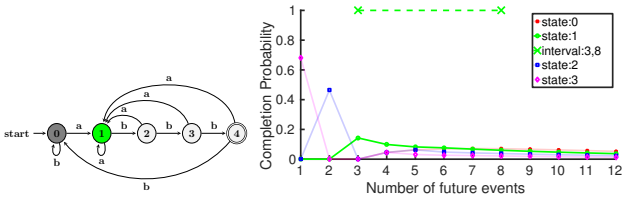
(b) $PMC_{\mathcal{P}}^1$

Figure 1: DFA and PMC for $\mathcal{P} = a; c; c$, $\Sigma = \{a, b, c\}$, and order $m = 1$ [2].

After constructing a PMC, we can use it in order to calculate the so-called *waiting-time* distributions. Given a specific state of the PMC, a *waiting-time* distribution gives us the probability of reaching a set of absorbing states in n transition from now (absorbing states are states with self-loops and probability equal to 1.0). By mapping the final states of the initial DFA to absorbing states of the PMC (see again Figure 1), we can therefore calculate the probability of reaching a final state, or, in other words, of detecting a full match of the original regular expression in n events from now.

In order to estimate the final forecasts, another step is required, since our aim is not to provide a single future point with the highest probability but an interval. Predictions are given in the form of intervals, like $I = (start, end)$. The meaning of such an interval is that the DFA is expected to reach a final state sometime in the future between *start* and *end* with probability at least some constant threshold θ_{fc} (provided by the user). These intervals are estimated by a single-pass algorithm that scans

a waiting-time distribution and finds the smallest (in terms of length) interval that exceeds this threshold. An example is shown in Figure 2, where the DFA in Figure 2a is in state 1, the *waiting-time* distributions for all of its non-final states are shown in Figure 2b and the distribution, along with the prediction interval, for state 1 are shown in green.



(a) DFA, state 1. **(b) Waiting-time distribution, state 1.**

Figure 2: Example of how prediction intervals are produced. $\mathcal{P} = a; b; b; b$, $\Sigma = \{a, b\}$, $m = 1$, $\theta_{fc} = 0.5$ **[2].**

The above described method assumes that we know the (possibly conditional) occurrence probabilities of the various event types appearing in a stream (as would be the case with synthetically generated streams). However, this is not always the case in real-world situations. Therefore, it is crucial for a system implementing this method to have the capability to learn the values of the PMC's transition matrix. One way to do this is to use some part of the stream to obtain the maximum-likelihood estimators for the transition probabilities. If $\mathbf{\Pi}$ is the transition matrix of a Markov chain with a set of states Q, $\pi_{i,j}$ the transition probability from state i to state j, $n_{i,j}$ the number of transitions from state i to state j, then the maximum likelihood estimator for $\pi_{i,j}$ is given by:

$$\hat{\pi}_{i,j} = \frac{n_{i,j}}{\sum_{k \in Q} n_{i,k}} = \frac{n_{i,j}}{n_i}$$

Executing this learning step on a single node might require a vast amount of time until we arrive at a sufficiently good model. In this paper, we present a distributed method for learning the transition probability matrix.

3.2 Pattern prediction on multiple streams

3.2.1 Problem formulation. Let $O = \{o_1, ..., o_k\}$ be a set of K objects (i.e., moving objects) and $S = \{s_1, ..., s_k\}$ a set of real-time streams of events, where s_i is generated by the object o_i. Let $\mathcal{P}$ be a user-defined pattern which we want to apply to every stream s_i, i.e., each object will have its own DFA.

The setting that is considered in this work is then described in the following: a large-scale patterns prediction over multiple input event streams system that consists of K distributed predictor nodes $n_1, n_2..., n_k$, each of which consumes an input event stream $s_i \in S$. Each node n_i $i \in [K]$ handles a single event stream s_i associated with a moving object $o_i \in O$, in addition, it maintains a local prediction model f_i for the user-defined pattern $\mathcal{P}$. The f_i model provides the online prediction about the future full match of the pattern $\mathcal{P}$ in s_i for each new arriving event tuple. In short, we have multiple running instances of an online prediction algorithm on distributed nodes for multiple input event streams. More specifically, we consider as an input massive streams of events that describe trajectories of moving vessels in the context of maritime surveillance, where there is one predictor node for each vessel's event stream.

3.2.2 The proposed approach. We design and develop a scalable and distributed patterns prediction system over massive input event streams of moving objects. We exploit the event forecasting with Pattern Markov Chains [2] as the base prediction model. Moreover, we propose to enable the information exchange between the distributed predictors/learners of the input event streams, by adapting the distributed online prediction protocol [15] to synchronize the prediction models, i.e., the transitions probabilities matrix of the Pattern Markov Chain (PMC) predictors.

Algorithm 1 presents the distributed online prediction protocol by dynamic model synchronization on both the predictor nodes and the coordinator. We refer to the PMC's transition matrix $\mathbf{\Pi}_i$ on predictor node n_i by f_i. That is, when a predictor $n_i : i \in [k]$ observes an event e_j it revises its internal model state (i.e., f_i) and provides a prediction report. Then it checks the local conditions (batch size b and local model divergence from a reference model f_r) to decide if there is a need to synchronize its local model with the coordinator or not. The typical choice of f_r is the last computed aggregated model from the previous synchronization step, which is shared between all local predictors/learners. By monitoring the local condition $\|f_i - f_r\|^2 > \Delta$ on all local predictors, we have a guarantee that if none of the local conditions is violated, the divergence (i.e., variance of local models $\delta(f) = \frac{1}{k} \sum_{j=1}^{k} \|f_i - \hat{f}\|^2$) does not exceed the threshold Δ [15].

On the other hand, the coordinator receives the models from nodes with the violation, then tries to query other nodes for their local models until receiving from all nodes or the variance of the already received models less or equal than the divergence threshold Δ. Finally, an aggregated model $\hat{f}$ is computed and sent back to the predictor nodes that sent their models after violation or have been queried by the coordinator.

This protocol was introduced for linear models, and has been extended to handle kernelized online learning models [16]. In this paper, we address to employee this protocol for the pattern prediction model, which is internally based on the Pattern Markov Chain $PMC^m_{\mathcal{P}}$. This adaption allows the distributed $PMC^m_{\mathcal{P}}$predictors for multiple event streams to synchronize their models (i.e., transition probability matrix of each predictor) within the system in a communication-efficient manner.

We propose a *synchronization operation* for the models parameters ($f_i = \mathbf{\Pi}_i : i \in [k]$) of the k distributed PMC predictors. The operation based on distributing the maximum-likelihood estimation [4] for the transition probabilities of the underlying $PMC^m_{\mathcal{P}}$ models described by:

$$\hat{\pi}_{i,j} = \frac{\sum_{k \in K} n_{k,i,j}}{\sum_{k \in K} \sum_{l \in L} n_{k,i,l}}$$

Moreover, we measure the divergence of local models from the reference model $\|f_k - f_r\|^2$ by calculating the sum of square

Algorithm 1: Communication-efficient Distributed Online Learning Protocol

Predictor node n_i: at observing event e_j
  update the prediction model parameters f_i and provide a prediction service
  if $j \mod b = 0$ *and* $\|f_i - f_r\|^2 > \Delta$ **then**
    send f_i to the Coordinator (violation)

Coordinator:
  receive local models with violation $B = \{f_i\}_{i=1}^{m}$
  while $|B| \neq k$ *and* $\frac{1}{|B|} \sum_{f_i \in \Pi} \|f_i - \hat{f}\|^2 > \Delta$ **do**
    add other nodes have not reported violation for their models $B \leftarrow \{f_l : f_l \notin B \text{ and } l \in [k]\}$
    receive models from nodes add to B
  compute a new global model $\hat{f}$
  send $\hat{f}$ to all the predictor nodes in B and set $f_1 \ldots, f_m = \hat{f}$
  if $|B| = k$ **then**
    set a new reference model $f_r \leftarrow \hat{f}$

difference between the transition probabilities $\mathbf{\Pi}_i$ and $\mathbf{\Pi}_r$:

$$\|f_k - f_r\|^2 = \sum_{i,j} (\hat{\pi}_k i, j - \hat{\pi}_r i, j)^2$$

In general, our approach relies on enabling the collaborative learning between the prediction models of the input event streams. By doing so, we assume that the underlying event streams belong to the same distribution and share the same behavior (e.g., mobility patterns). We claim that assumption is reasonable in many application domains, for instance, in the context of maritime surveillance, vessels travel through defined routes by International Maritime Organization (IMO). Additionally, vessels have similar mobility patterns in specific areas such as moving with low speed and multiple turns near the ports [19, 28]. That allows our system to dynamically construct a coherent global prediction model for all input event streams based on merging their local prediction models.

3.3 Distributed architecture

Our system consumes an aggregated events stream as input[2] of large number of moving objects, which is continuously collected and fed into the system. It allows users to register a pattern $\mathcal{P}$ to be monitored over each event stream of a moving object. Output stream consists of original input events alongside with predictions of full matches of $\mathcal{P}$ is outputted to be displayed to the end users. Figure 3 presents the overview of our system architecture and its main components.

The system is composed of three processing units: (i) pre-processing operators that receive the input event stream, and perform filtration, ordering operations, before partitioned the input event stream to multiple event streams based on the associated moving object (ii) predictor nodes (learners), which are responsible to maintain a prediction model for the input event streams, such that each prediction node is configured to handle an event

[2]In practice, the aggregated input events stream is composed of multiple event streams (partitions) for multiple moving objects, which are reconstructed bt the system internally.

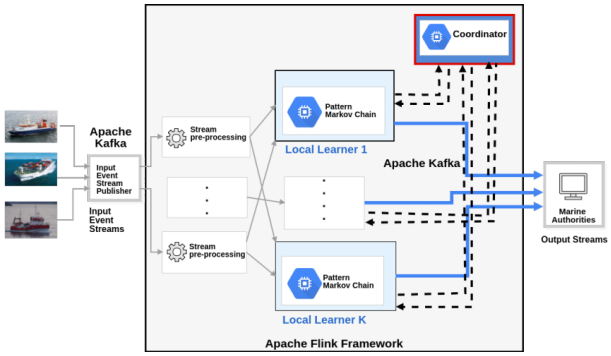


Figure 3: System Architecture.

stream from the same moving object, in order to provide online predictions for a predefined pattern $\mathcal{P}$ (iii) a coordinator node that communicates through a Kafka stream channels with the predictors to realize the distributed online learning protocol, which builds a global prediction model based on the received local models, and then share it among the predictors.

Our system is a distributed system consists of multiple pre-processing operators, prediction nodes, and a central coordinator node. All units are running concurrently and arranged as data processing pipeline as depicted in Figure 3. We leverage the Apache Kafka as messaging platform to ingest the input event streams and to publish the result streams, in addition, it used as the communication channel between the predictor nodes and the coordinator. While Apache Flink is employed to execute the system's distributed processing units over the input event streams including the pre-processing, prediction, and the coordinator unit. Our system architecture can be modeled as logical network of processing nodes organized in a Directed Acyclic Graph (DAG), which is inspired by the Flink runtime dataflow programs [6].

4 IMPLEMENTATION DETAILS

This sections provides a detailed overview of the system's implementation, our system has been implemented on top of Apache Flink and Apache Kafka frameworks. Each of the three submodules described in Section 3.3 have been implemented as Flink operations over the input events stream.

Pre-processing and Prediction Operators. Listing 1 shows how the workflow of the system is implemented as data flow in Flink program.

```
.....
DataStream<Event> inputEventsStream =
     env.addSource(flinkKafkaConsumer);
// Create event tuples (id,event) and assign time stamp
DataStream<Tuple2<String, sEvent>> eventTuplesStream =
inputEventsStream.map(new EventTuplesMapper())
.assignTimestampsAndWatermarks(new
     EventTimeAssigner());
// Create the ordered keyed stream
KeyedStream<Tuple2<String, Event>, Tuple>
     keyedEventsStream =
eventsStream.keyBy(0).process(new EventSorter())
.keyBy(0);
```

```
//Initialize the predictor node
LocalPredictorNode predictorNode =new
     LocalPredictorNode<Event>(P);
// Process the keyedEventsStream by the predictor
DataStream<Event> processedEventsStream =
keyedEventsStream.map(predictorNode);
...
```

Listing 1: Flink pipeline for local predictors workflow

The system ingests the input events stream form a Kafka cluster that mapped to **DataStream** of events, which is then processed by a **EventTuplesMapper** to create tuples of (id,event), where the **id** is associated moving object id. And in order to manage the out of order incoming events the tuples stream is processed by a **TimestampAssigner** that assigns the timestamps for the input events based on the extracted creation time. Afterward, an ordered stream of event tuples is generated using a process function **EventSorter**. The ordered stream is transformed to a **keyedEventsStream** based on partitioning it based on the ids of the associated moving object using a **keyBy** operation. A local **predictor** node in a distributed environment is represented by a **map** function over the **keyedEventsStream**, while each parallel instance of the map operator (predictor) always processes all events with the same id, and maintains a bounded prediction model (i.e., $PMC^m_{\mathcal{P}}$predictor) using the Flink's Keyed State [3]. The output streams of the moving objects form the predictor map parallel instances are sent to a new Kafka stream (i.e., same topic name) that can be processed by other components like visualization or users notifier.

Moreover, the implementation of a **predictor** map function includes the communication with **coordinator** using Kafka streams. At the beginning of the execution it sends a reregistration request to the coordinator. And at runtime it sends its local prediction model as synchronization request, or as a response for a resolution request from the coordinator. In addition, it receives a resolution request from the coordinator to send its model. These communication messages are published onto different Kafka topics as depicted in Table 1.

Table 1: Message to Kafka topics mapping.

Message	Kafka Topic
RegisterNode, **RequestSync** and **ResolutionAnswer**	LocalToCoordinatorTopicId
CoordinatorSync and **RequestResolution**	CoordinatorToLocalTopicId

Coordinator. Listing 2 presents the workflow of the coordinator node that manages the distributed online learning protocol operations, which is implemented as Flink program. The coordinator receives messages from the local predictors through a Kafka Stream of a topic named **"LocalToCoordinatorTopicId"**, while it is implemented as a single **map** function over the messages stream by setting the **parallelism** level of the Flink program to **"1"**, while increasing the parallelism will scale up the number of parallel coordinator instances, but for the current setting a single node in needed. The coordinator map operator handles three type of messages from the predictors: (i) **RegisterNode** that contains a registration request for a new predictor instance, (ii) **RequestSync** to receive a local model of a predictor node after violation, (iii) **ResolutionAnswer** to receive a resolution response from a local predictor node. While it sends **CoordinatorSync** messages for all predictors after creating a new global prediction model, or **RequestResolution** to a ask the local predictors for their prediction models.

```
...
   StreamExecutionEnvironment env = [...];
   // Set a default parallelism for all operators
   env.setParallelism(1);

   // Read messages from local predictors
   DataStream<TopicMessage> messagesStream =
        readKafkaStream(env, "LocalToCoordinatorTopicId");

   // Initialize the coordinator node
   CompunctionEfficientCoordinator coordinatorNode = new
        CompunctionEfficientCoordinator(configs);

   DataStream<CoordinatorMessage>
        coordinatorMessagesStream =
        messageStream.map(coordinatorNode);

   // Send the messages form the coordinator to the local
        predictors
   writeKafkaStream(coordinatorMessagesStream,
        CoordinatorToLocalTopicId);

...
```

Listing 2: The coordinator Flink program.

5 EMPIRICAL EVALUATION

In this section, we evaluate the performance of our system using real-world event streams provided by the datAcron project in the context of maritime domains. The used event streams describe trajectory critical points (i.e., synopses) of moving vessels, which are derived from raw AIS messages as was described in [29]. In particular, for our evaluation we used a data set of synopses contains 4, 684, 444 critical points of 5055 vessels sailing in the Atlantic Ocean during the period from 1 October 2015 to 31 March 2016. We use it to generate a simulated stream of event tuples *(id,type,timestamp, longitude, latitude, annotation, speed, heading)*, where $type \in \Sigma$ and $\Sigma = \{VerySlow, Slow, Moving, Sailing, Stopping\}$ that is based on the vessel speed, and $annotation$ is an attribute to encode the trajectory movement events such as speed change, slow motion, and gap in reporting. In our experiments, we monitor a pattern $\mathcal{P} = Sailing$ that detects when the vessel under way (sailing).

[3] https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/stream/state.html#kayed-state

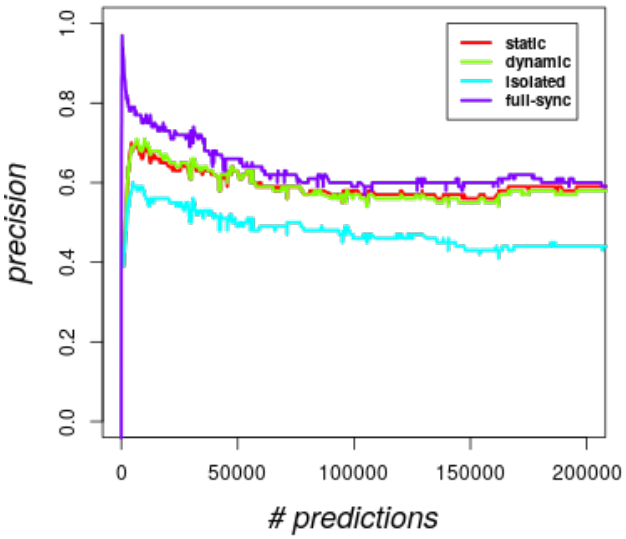


Figure 4: Precision scores.

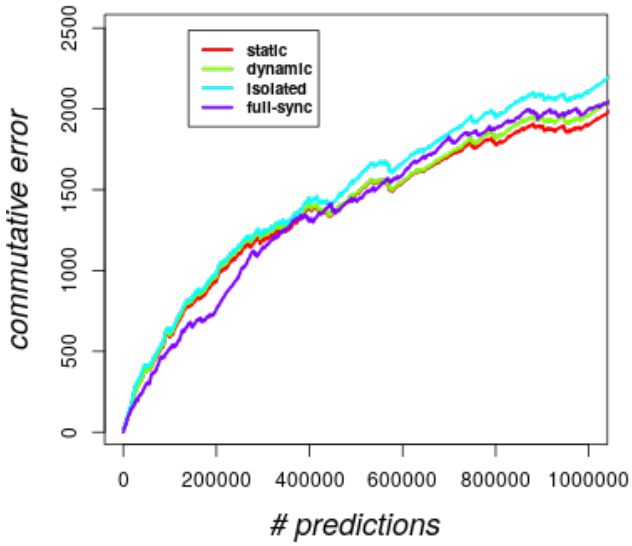


Figure 5: Commutative error.

Experimental setup. We ran our experiments on single-node standalone Flink cluster deployed on a server (Ubuntu 17.04) with Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz X 8 processors and 32GB RAM. We used Apache Flink v1.3.2 and Apache Kafka v0.10.2.1 for our tests.

Evaluation criteria. Our goal is to evaluate our distributed pattern prediction based on enabling the synchronization of PMC models between the predictors against the isolated ones (i.e., without exchanging information), we compare the predictive performance in terms of : (i) average *precision* = $\frac{\#\ of\ \ correct\ \ predictions}{\#\ of\ \ total\ \ predictions}$ per predictor node (ii) *cumulative error* that presents the aggregated number of all wrong predictions. (iii) *recall* that measures the fraction of actual full matches of the defined pattern does the model predicate at least once in previous prediction report Moreover, we study the communication cost by measuring the *cumulative communication* that captures the number of synchronization related messages, which is introduced by employing the static or dynamic synchronization schemes of the distributed online learning protocol. In next, we present the experimental results for the for the pattern $\mathcal{P} = Sailing$ with $m = 2$, batch size $b = 100$, variance threshold $\Delta = 2$, and PMC prediction threshold $\theta_{fc} = 80\%$.

Experimental results. Figure 4 depicts the precision scores (i.e., average precision of prediction model per vessel) of all approaches, namely, isolated without synchronization, continuous synchronization, static (periodic), and our proposed approach based on the dynamic synchronization scheme. While Figure 5 shows the average cumulative error per predictor.

In Table 2, we present the statistics of recall results for the different approaches. It can be seen

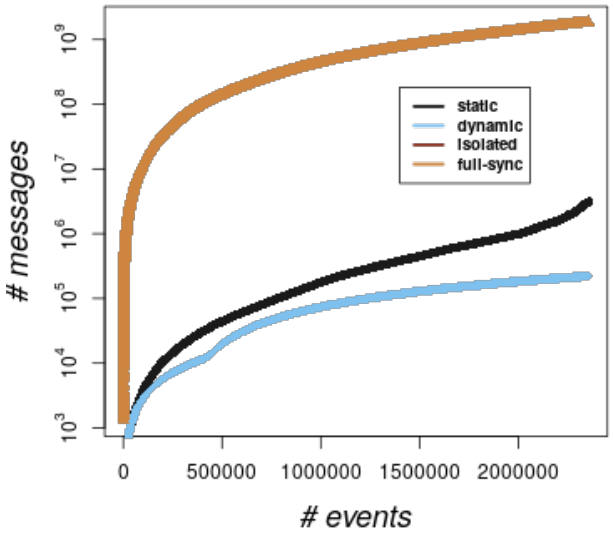


Figure 6: Commutative communication.

Table 2: Recall results.

Approach	Mean	Max
isolated	0.1948	0.23
static	0.1976	0.23
dynamic	0.1974	0.23
full-sync	0.1979	0.33

Figure 6 provides the accumulated communication required for the three approaches based on the distributed online learning. As expected, larger amount of communication is required for the continuous synchronization comparing to static and dynamic approaches, and it can bee seen with small variance threshold $\Delta = 2$ that we can reduce the communication using the dynamic synchronization and comparing the (static) periodic method, and furthermore, still preserving the predictive performance as was shown in Figures 4 and 5.

6 CONCLUSION

In this paper, we presented a system that provides a distributed pattern prediction over multiple large-scale event streams of moving objects(vessels). The system uses the event forecasting with Pattern Markov Chain (PMC) [2] as base prediction model on each event stream, and it applies the protocol for distributed online prediction [15] to exchange information between the models of multiple input event streams. Our proposed system has been implemented using Apache Flink and Apache Kafka, and empirically tested against a large real-world event streams related to trajectories of moving vessels.

ACKNOWLEDGMENTS

This work was supported by the EU H2020 datAcron project (grant agreement No 6875).

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules Between Sets of Items in Large Databases. In *ACM SIGMOD*.

[2] Elias Alevizos, Alexander Artikis, and George Paliouras. 2017. Event Forecasting with Pattern Markov Chains. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. ACM, 146–157.

[3] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. 2015. Complex event recognition under uncertainty: A short survey. *Event Processing, Forecasting and Decision-Making in the Big Data Era (EPForDM)* (2015), 97–103.

[4] Theodore W Anderson and Leo A Goodman. 1957. Statistical inference about Markov chains. *The Annals of Mathematical Statistics* (1957), 89–110.

[5] Amazon Web Services (AWS). 2013. Amazon Kinesis. https://aws.amazon.com/de/kinesis/. (2013).

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

[7] Gianpaolo Cugola and Alessandro Margara. 2012. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Comput. Surv.* 44, 3, Article 15 (June 2012), 62 pages. https://doi.org/10.1145/2187671.2187677

[8] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. 2012. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research* 13, Jan (2012), 165–202.

[9] Lina Fahed, Armelle Brun, and Anne Boyer. 2014. Efficient Discovery of Episode Rules with a Minimal Antecedent and a Distant Consequent. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*. Springer.

[10] Ioannis Flouris, Vasiliki Manikaki, Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Michael Mock, Sebastian Bothe, Inna Skarbovsky, Fabiana Fournier, Marko Stajcer, et al. 2016. FERARI: A Prototype for Complex Event Processing over Streaming Multi-cloud Platforms. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2093–2096.

[11] The Apache Software Foundation. 2012. Apache Kafka. https://kafka.apache.org/. (2012).

[12] The Apache Software Foundation. 2013. Apache Spark Streaming. http://spark.apache.org/streaming/. (2013).

[13] The Apache Software Foundation. 2014. Apache Flink. https://flink.apache.org/. (2014).

[14] The Apache Software Foundation. 2014. Apache Storm. http://storm.apache.org/. (2014).

[15] Michael Kamp, Mario Boley, Daniel Keren, Assaf Schuster, and Izchak Sharfman. 2014. Communication-efficient distributed online prediction by dynamic model synchronization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 623–639.

[16] Michael Kamp, Sebastian Bothe, Mario Boley, and Michael Mock. 2016. Communication-Efficient Distributed Online Learning with Kernels. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 805–819.

[17] John Langford, Alex J Smola, and Martin Zinkevich. 2009. Slow learners are fast. *Advances in Neural Information Processing Systems* 22 (2009), 2331–2339.

[18] Srivatsan Laxman, Vikram Tankasali, and Ryen W. White. 2008. Stream Prediction Using a Generative Model Based on Frequent Episodes in Event Sequences. In *ACM SIGKDD*.

[19] Bo Liu, Erico N de Souza, Stan Matwin, and Marcin Sydow. 2014. Knowledge-based clustering of ship trajectories using density-based approach. In *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 603–608.

[20] David Luckham. 2008. The power of events: An introduction to complex event processing in distributed enterprise systems. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*. Springer, 3–3.

[21] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. 1997. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery* (1997).

[22] Michael Mathioudakis and Nick Koudas. 2010. Twittermonitor: trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 1155–1158.

[23] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. 2012. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks* 10, 7 (2012), 1497–1516.

[24] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. 2015. *Introduction to time series analysis and forecasting*. Wiley.

[25] Vinod Muthusamy, Haifeng Liu, and Hans-Arno Jacobsen. 2010. Predictive Publish/Subscribe Matching. In *DEBS*. ACM.

[26] Grégory Nuel. 2008. Pattern Markov Chains: Optimal Markov Chain Embedding through Deterministic Finite Automata. *Journal of Applied Probability* (2008).

[27] International Maritime Organization. 2001. Automatic identification systems. http://www.imo.org/OurWork/Safety/Navigation/Pages/AIS.aspx. (2001).

[28] Giuliana Pallotta, Michele Vespe, and Karna Bryan. 2013. Vessel pattern knowledge discovery from AIS data: A framework for anomaly detection and route prediction. *Entropy* 15, 6 (2013), 2218–2245.

[29] Kostas Patroumpas, Elias Alevizos, Alexander Artikis, Marios Vodas, Nikos Pelekis, and Yannis Theodoridis. 2017. Online event recognition from moving vessel trajectories. *GeoInformatica* 21, 2 (2017), 389–427.

[30] Kostas Patroumpas, Alexander Artikis, Nikos Katzouris, Marios Vodas, Yannis Theodoridis, and Nikos Pelekis. 2015. Event Recognition for Maritime Surveillance.. In *EDBT*. 629–640.

[31] R. Vilalta and Sheng Ma. 2002. Predicting rare events in temporal domains. In *ICDM*.

[32] Lin Xiao. 2010. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research* 11, Oct (2010), 2543–2596.

[33] Feng Yan, Shreyas Sundaram, SVN Vishwanathan, and Yuan Qi. 2013. Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties. *IEEE Transactions on Knowledge and Data Engineering* 25, 11 (2013), 2483–2493.

[34] Cheng Zhou, Boris Cule, and Bart Goethals. 2015. A pattern based predictor for event streams. *Expert Systems with Applications* (2015).

[35] Cheng Zhou, Boris Cule, and Bart Goethals. 2015. A pattern based predictor for event streams. *Expert Systems with Applications* 42, 23 (2015), 9294–9306.