Rheinische
Friedrich-Wilhelms-Universität Bonn

Mathematisch-naturwissenschaftliche
fakultät

Master Thesis

Distributed Online Learning for Large-scale Pattern Prediction over Real-time Event Streams

Author:
Ehab Qadah

First Examiner:
PD. Dr. Michael Mock

Second Examiner:
Prof. Dr. Stefan Wrobel

DRAFT

Declaration of Authorship

I herewith confirm, according to §17(7) of the examination regulations for the Master's Programme in Computer Science at the University of Bonn, that I composed the Master's Thesis independently and that none other than the specified sources and aids were used and that any citations have been marked.

Ehab Qadah

Abstract

In many application domains, such as maritime surveillance, financial services, network monitoring, and sensor networks, massive amounts of streaming data are being generated in real-time. The records of these streams can be encoded as events. However, in order to benefit from the live streaming events, there is a need for systems that enable the real-time stream processing and analytics tasks at large-scale. For instance, predicting full matches of complex patterns from the massive streaming events is an important utility for the decision making process. Such a utility allows to react proactively to the new situations and improve the effectiveness of the operational tasks.

In this thesis, we present the design, implementation, and evaluation of a scalable prediction system for user-defined patterns over multiple massive streams of events. The proposed system is based on a novel approach of combining probabilistic events pattern prediction models on multiple predictor nodes with a distributed online learning protocol, to continuously learn the parameters of a global prediction model in a communication-efficient way, and share it among the predictors. For efficiency and scalability, the system is implemented on top of Apache Flink, a popular engine for distributed and large-scale stream processing.

The key idea of our system is to enable the cooperative learning and information exchange between the distributed predictors by sharing a global prediction model, where the learning convergence is accelerated with less data for each predictor. We describe the distributed architecture and implementation of the proposed system along with the theoretical analysis that focuses on giving a probabilistic learning guarantee for the proposed synchronized global model. Also, we provide experimental results on synthetic and real-world event streams that prove the effectiveness of our proposed approach.

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor, PD. Dr. Michael Mock for giving the opportunity to work on this research. The work presented in this thesis would not have been possible without his supervision and guidance throughout the whole work and since its very first stages. I also would like to thank Prof. Dr. Stefan Wrobel for all his help and supervision on this thesis.
This thesis was conducted at Fraunhofer Institute for Autonomous Intelligent Systems (IAIS), and it is partially funded by the EU Horizon 2020 datAcron project (grant agreement No 687591).

Contents

List of Figures

1 Introduction

1.1 Motivation

In the past decade, technological advancements have led to a growing availability of massive amounts of continuous event streams in many application domains such as social networks [45, 34], Internet of Things (IoT) [36], user activities on the web [8, 35] and maritime monitoring [42, 28].

These event streams used to be stored and processed later, while monitoring and processing them in real-time increases their value [13]. Moreover, the real-time streams provide the opportunity to implement reactive components within these domains. For example, the ability to detect and predict the full matches of a pattern of interest (e.g., a certain sequence of events), defined by a domain expert, is typically important for operational decision-making tasks in their respective domains.

An event stream is an unbounded collection of time-ordered data observations in the form of a tuple of attributes that is composed of a value from finite event types along with other categorical and numerical attributes [1, 46, 53, 19]. On the other hand, event patterns are defined using a pattern language such as SQL-like [46] or regular expressions [4].

As an illustrative example, consider the movement event streams of a group of vessels in the context of maritime surveillance. The event stream of a moving vessel consists of spatio-temporal and kinematic information along with the vessel's identification and its trajectory related events, based on the Automatic Identification System (AIS) [40] messages that are continuously sent by the vessel. Therefore, leveraging event patterns prediction over real-time streams of moving vessels is useful to alert maritime operation managers about suspicious activities (e.g., fast sailing vessels near ports or illegal fishing) before they happen [42].

However, processing the real-time streaming data poses new challenges, since the data streams are large, distributed in nature and continuously arrive at a high rate [7, 18]. To deal with these data streams in a fast and efficient manner, a distributed stream processing framework [21, 22, 23] is usually used to implement the stream processing and analytic applications.

1.2 Thesis Overview

In this thesis, we present the design, implementation, and evaluation of a scalable and distributed system that provides online pattern prediction over multiple real-time streams of events. The proposed approach is based on a novel method that combines a distributed online learning protocol [26], with online probabilistic prediction models based on pattern Markov chain technique [4]. The underlying model provides online predictions about when a pattern is expected to be completed within each event stream, where the patterns are defined in the form of regular expressions over the event types in the stream.

In particular, we consider the setting when there are K input event streams, and for each one of these streams, a local prediction model is built on a distinct node. As a result, K models are maintained separately on K distributed predictor nodes each of which is consuming a single event stream.

We propose to make the predictors of the event streams collectively learn a global model in a distributed and communication-efficient fashion. Toward this, we have adapted the distributed online learning protocol [26], where the pattern predictors [4] send their local models to a coordinator node. We introduce a new synchronization operator that combines these independent local models to create a stronger global model and share it among all the predictors.

In addition, we implement our system on top of the Big Data framework for distributed stream processing Apache Flink [22], and the streaming platform Apache Kafka [20]. We also evaluate our proposed system over synthetic event streams, and large-scale real-world data streams related to movement events of vessels, which are provided in the context of the datAcron project[1].

In summary, the main contributions of this thesis are the following:

- A new method of combining a distributed online learning protocol with pattern Markov chain predictors over multiple input event streams. It is based on enabling the collaborative learning and information exchange among the predictors in a communication-efficient manner.

- A study of the characteristics of the proposed method, by providing a theoretical analysis of the synchronization operation within the distributed online learning protocol, in which it is provided a probabilistic guarantee of the learning efficiency.

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

- The description of the architecture of the proposed distributed system for event patterns prediction over massive event streams, along with the building blocks for the system's workflow processing. The implementation details of the system on top of Apache Flink and Apache Kafka are also presented.
- An extensive empirical evaluation of the proposed approach over real-world and synthetic streams of events.

1.3 Publication

Parts of this thesis have been published in [44]:

Ehab Qadah, Michael Mock, Elias Alevizos, and Georg Fuchs. A Distributed Online Learning Approach for Pattern Prediction over Movement Event Streams with Apache Flink. In *EDBT/ICDT Workshops*, 2018.

1.4 Outline

The rest of this thesis is organized as follows:

Chapter 2 provides an overview of the related work on pattern prediction over event streams, and the distributed online learning techniques. It also gives an introduction to Apache Flink and Apache Kafka.

Chapter 3 describes the problem of pattern prediction over a single event stream and introduces the used base model. It also presents our method for pattern prediction over multiple input event streams by leveraging the distributed online learning protocol. Additionally, it provides the theoretical analysis and a probabilistic learning guarantee of the proposed approach.

Chapter 4 presents the architecture of the proposed system, and the implementation details in Flink and Kafka.

In **Chapter 5**, the experiments and results over real-world event streams of moving vessels and synthetic event streams are presented.

Finally, conclusions are given in **Chapter 6**, along with potential directions for future work.

2 Related Work and Background

In this chapter, we survey some of the related research in the areas of pattern prediction over event streams task, and techniques of distributed online learning over data streams. Then we give a brief overview of the Big Data platforms that we used to implement our proposed system.

2.1 Related Work

2.1.1 Pattern Prediction over Event Streams

Several approaches have been proposed to formalize the task of event patterns (complex events) prediction over time-evolving data streams. One common way to formalize this task is to assume that the stream is a time-series of numerical values, and the goal is to predict at each time point t the next observations at some future points $t+1$, $t+2$, etc., (or even the output of some function of future values) [37].

Another way to formalize the prediction task is to view streams as sequences of events, i.e., tuples of multiple attributes, such as *id*, *event type*, *timestamp*, etc., and the goal is to predict future events or patterns of events. In this work, we focus on the latter definition of forecasting (prediction of complex event patterns).

A relevant work of the detection of event patterns task has been established in the field of temporal pattern mining. Where events are defined as 2-tuples of the form (*EventType*, *Timestamp*). The goal is to extract patterns of events in the form of association rules [2] or frequent episode rules [33].

These rule-based methods have been extended in order to be able to learn rules for predicting event patterns. For instance, in [49], an association rule mining technique is introduced. This technique works as following. Firstly, it extracts sets of event types that frequently lead to a target event (i.e., a rare event) within a time window of a fixed size. Then, it uses them to build a rule-based prediction model. Also, Weiss and Hirsh [50] proposed another rule-based method to predict rare events in a stream, using a genetic algorithm to find all predictive patterns to form prediction rules.

On the other hand, Laxman et al. [29] have proposed a probabilistic model for calculating the probability of the immediately next event in the stream. Which is achieved by combining each frequent episode [32] that presents a partially-ordered set of event types, with a Hidden Markov Model (HMM). In addition, Fahed et al. [16] have proposed episode rules mining algorithm which predicts distant events. By generating a set of episode rules in the form $P \rightarrow Q$ where P and Q are two episodes. These rules have a minimal antecedent (in number of events) and temporally distant consequent.

In [53] a mining method is presented that finds the frequent sequential patterns in a stream of events, which are then used to generate prediction rules. The event stream is processed into batches, in each batch the events are consumed to find the prefix matches from the discovered frequent patterns, to predict future events using different strategies of prediction scoring.

Event pattern prediction has also attracted some attention from the field of complex event processing (CEP), where the CEP system consumes a stream of low-level events to detect patterns of events (composite events) that are defined using pattern-based languages, these languages provide logic, sequence, and iteration operators such as SQl-like languages [14].

One such early approach is presented in [38], which is based on converting the complex event patterns to automata, and subsequently, Markov chains are used in order to estimate when a pattern is expected to be fully matched.

Moreover, Alevizos et al. [4] have recently presented a similar approach, where automata and Markov chains are employed in order to provide (future) time intervals during which a full match of the pattern is expected with a probability above a confidence threshold. In this thesis, we leverage this method as the base prediction model for each input event stream (see Section 3.1.2).

2.1.2 Distributed Online Learning

In recent years, there have been many research efforts on the problem of distributed online learning over multiple data streams [47, 51, 11, 52, 15, 26]. In contrast to the centralized learning approach in which all records of the streams are processed at a single central machine, the data streams are processed in a distributed fashion on k local learners. Each of which learner processes a single stream and use a local learning algorithm to provide a real-time prediction based service like Classification [11].

However, this distributed setting requires the learners to exchange the parameters of underlaying learning algorithm to construct a strong global model, in order to preserve a similar predictive performance to the centralized setting [26].

Different communication schemes between the distributed learners have been proposed in the literature. For instance, Dekel et al. [15] have proposed a distributed online mini-batch prediction approach over multiple data streams. Their approach is based on a static synchronization method. The distributed learners/predictors 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 among all local learners.

This work has been extended in [26] by introducing a dynamic synchronization scheme that reduces the required communication to the exchange of information between the learners. The proposed protocol relays on a dynamic synchronization operator that controls when the local learners communicate their models, which is based on only synchronizing the local models of the learners if they diverge from a reference model.

This protocol was introduced for linear models, and has been extended to handle kernelized online learning models [27]. In this work, we consider the event patterns prediction models over multiple event streams as learning algorithms, and we introduce to employ the communication-efficient distributed online learning protocol [26] to synchronize their parameters as illustrated in Section 3.2.2.

2.2 Technological Background

In the last years, many systems for large-scale and distributed stream processing have been proposed, including Spark Streaming [21], Apache Storm [23] and Apache Flink [22]. These frameworks can ingest and process real-time data streams, published from different distributed message queuing platforms, such as Apache Kafka [20] or Amazon Kinesis [6].

In this thesis, we implemented the proposed system in Apache Flink. Flink provides the distributed stream processing components of the distributed event pattern predictors. It works alongside Apache Kafka, which is used for streaming the input event streams and as a messaging platform that enables the distributed online learning functionalities.

2.2.1 Apache Flink

Apache Flink is an open source project that provides a large-scale, distributed, and stateful stream processing platform [12]. Flink is one of the most recent and pioneering Big Data processing frameworks.

It provides processing models for both streaming and batch data, where the batch processing model is treated as a special case of the streaming one (i.e., finite

stream). Flink's software stack includes the *DataStream* and *DataSet* APIs for processing infinite and finite data, respectively. These two core APIs are built on top of Flink's core dataflow engine and provide multiple operations on data streams or sets such as *mapping*, *filtering*, *grouping*, etc.

The two main data abstractions of Flink are *DataStream* and *DataSet*, they represent read-only collections 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*. Flink's core is a distributed streaming dataflow engine.

Flink programs are represented by a data-flow graphs (i.e., directed acyclic graph - DAG) that get executed by Flink's dataflow engine [12]. 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 whose number is determined by the *parallelism* level. Each parallel operator instance is executed in an independent task slot on a machine within a cluster of computers [22]. Figure 2.1 shows an example of data flow graph in for Flink's program.

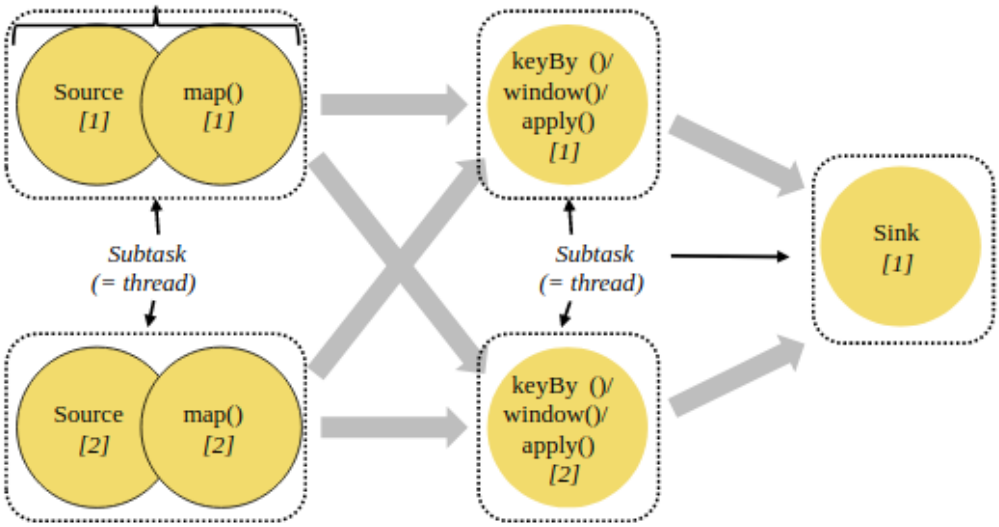

Figure 2.1: A parallel data flow graph in Flink [22].

2.2.2 Apache Kafka

Apache Kafka is a scalable, fault-tolerant, and distributed streaming framework/messaging system [20]. 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 then be consumed by applications, using the Consumer API that allows them to read the published data stream in the Kafka cluster. In addition, the streams of messages are distributed and load balanced between the multiple receivers within the same consumer group for the sake of scalability. In this work, we leverage the topic-based routing of the messages to control the communication that is required to perform the distributed online learning protocol.

3 The Problem of Pattern Prediction and the Proposed Approach

In this chapter, we introduce our proposed approach for pattern prediction over multiple input event streams. We start by defining the problem that we address in this thesis, which is divided into two parts: the formal definition of the pattern prediction over an individual event stream, then it is extended to the case of multiple streams. We present the used base model for event pattern prediction, and our proposed approach to leverage the distributed online protocol to enable knowledge sharing between prediction models over multiple event streams. Furthermore, we provide an analysis of the theoretical aspects of the proposed approach.

We follow the general notation and terminology of [1, 46, 31, 3, 53, 26] to formalize the problem we tackle and our solution approach.

3.1 Pattern Prediction over an Event Stream

In this section, we define the first part of the problem that we tackle in this thesis, by defining the problem of pattern prediction over a single event stream. We also give an introduction of the base prediction model, which is built based on the event forecasting with pattern Markov chains method [4].

3.1.1 Basic Problem Formulation

We first define the input event and the stream of events as follows:

Definition 1. *Each event is defined as a tuple of attributes denoted by e_i, where $e_i = (id, type, \tau, a_1, a_2, \ldots, a_n)$, 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 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 over a set of event types Σ (i.e., alphabet). Let a word over Σ be a sequence of event types, and the set of words over Σ be a language L. Then $L(\mathcal{P})$ is the regular language defined by the regular expression $(\mathcal{P})$ [24, 39, 4], where a regular expression is an event type $\in \Sigma$, or defined using the following operators over regular expressions:

- *sequence* that represents a concatenation of two regular expressions languages
- *disjunction* i.e., union, which is the language that its words belong to one of the languages of the two regular expressions
- *iteration* operation to define the set of all possible concatenation over a regular expression

More formally, a pattern is given through the following grammar:

Definition 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 $\mathcal{P}$ (regular expression) is encoded by a deterministic finite automata (DFA).

Definition 4 ([24])**.** $(\Sigma, Q, s, F, \delta)$ *is a deterministic finite automaton (DFA) where* Σ *a finite alphabet of event types, and* Q *is a finite set of states,* $s \in Q$ *a start state and* $F \subset Q$ *represents all final states.* $\delta : Q \times \Sigma \to Q$ *is a transition function from a state to another state given an input event type, which is defined as recursive function* $\delta(q, e_1 e_2 \dots e_d) = \delta(\delta(q, e_1 e_2 ... e_{d-1}), e_d)$. *If* $\delta(s, w) \in F$ *of a word* $w = e_1 e_2 \dots e_d \in \Sigma^*$, *then the word* w *is said to be accepted by the DFA. In addition,* $\delta(q)^{-m}$ *is the set of words of size* m *defined as* $\{w \in \Sigma^m \mid \exists q \in Q,\ \delta(p, w) = q\}$

The problem setting can be summarized 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 completed (full match).

3.1.2 Base Prediction Model

In this thesis, we use the event forecasting with pattern Markov chains method [4] to construct a pattern prediction model over an event stream. In next, we describe the details of this approach. We first provide an overview of the Patten Markov Chain framework [39]. Then, we describe how this framework is used to build a pattern prediction model [4].

Pattern Markov Chain (PMC)

Alevizos et al. [4] proposed to employ the Pattern Markov Chain (PMC) [39] to build an online prediction model associated with a pattern over an event stream. The algorithm of constructing a PMC associated with a pattern $\mathcal{P}$ over a stream of events s consists of the following steps:

- Build a deterministic finite automata (DFA) that accepts the regular expression $\Sigma^*;\mathcal{P}$. We define the the DFA as following:
  Firstly, the regular expression $\Sigma^*;\mathcal{P}$ is converted to non-deterministic finite automata (NFA), then an equivalent DFA of the constructed NFA is computed using the subset construction [24, 4]. And the events stream s is considered as the input word of the DFA.
  Furthermore, if $\forall q \in Q$ of a DFA the all $\delta(q)^{-m}$ are sets of cardinality 1, then the DFA is called *m-unambiguous*. Nuel [39] proposed a procedure to transform any DFA to an *m-unambiguous* DFA, which is achieved by incrementally duplicate all states have m-ambiguity (i.e., $|\delta(q)^{-m}| > 1$). Figure 3.1 shows an associated DFA of a sequential pattern $\mathcal{P} = a;d;c$ over an alphabet $\Sigma = \{a, b, c, d\}$.
- Given a constructed *m-unambiguous* DFA for the pattern $\mathcal{P}$, and assuming that the input events stream $s = \langle e_1, e_3, ..., e_t, ...\rangle$ is a m-order Markov sequence, where $m \geq 1$. Nuel [39] showed that the sequence of states of the DFA that generated by consuming the input events stream s is a first-order Markov chain, which is represented by $\langle q_0, q_1, ..., q_t, ...\rangle$, where $q_0 = s$ and $q_t = \delta(q_{t-1}, e_t)$. We denote by $PMC_m^{\mathcal{P}}$ the derived Markov chain associated with a pattern $\mathcal{P}$ that is called a Pattern Markov Chain (PMC) of order m [39]. In other words, we perform 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. Thus, the terms *m-unambiguous* DFA of a pattern and the corresponding $PMC_m^{\mathcal{P}}$ we will be used interchangeably.
  Furthermore, the $PMC_m^{\mathcal{P}}$ model is characterized by $|Q| \times |Q|$ transition probability matrix Π where Q is the set of states of the *m-unambiguous* DFA. Figure 3.2 depicts the PMC of order 1 for the generated DFA of Figure 3.1.

Constructing Pattern Prediction Model

Alevizos et al. [4] proposed to use the constructed $PMC_m^{\mathcal{P}}$ to build a probabilistic prediction model that describes the DFA's run-time behavior. The method is based on calculating the *waiting-time* distributions. Given a specific state of the $PMC_m^{\mathcal{P}}$, a *waiting-time* distribution provides the probability of reaching a set of

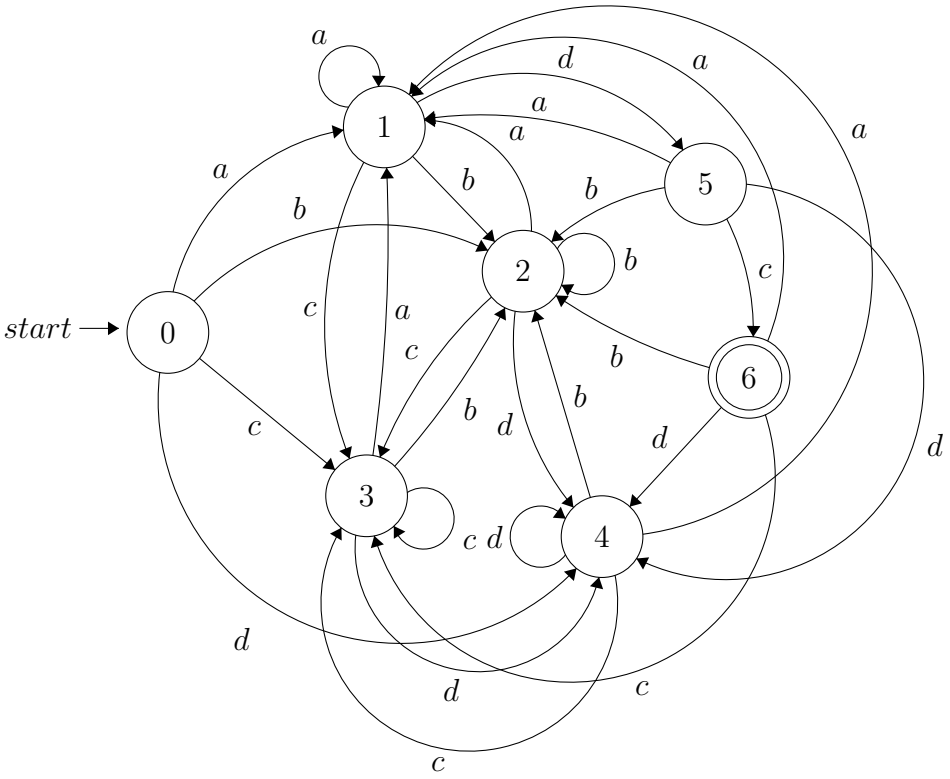

Figure 3.1: $DFA_{\Sigma^*;\mathcal{P}}$ for $\mathcal{P} = a; d; c$ with $\Sigma = \{a, b, c, d\}$, and order $m = 1$.

absorbing states in n transition from current state. So by mapping the final states of the DFA to absorbing states of the $PMC_m^{\mathcal{P}}$ by adding self-loops with probabilities equal to 1.0. Therefore, we can calculate the probability of reaching a final state in n transitions, which means predicting a full match of the defined pattern $\mathcal{P}$.

We denote by $W_{\mathcal{P}}(q)$ the waiting-time random variable that represents the number of transitions from a current state q of DFA to reach a final state [4], where it also represents the expected number of future events from the current time event to a full match of the pattern $\mathcal{P}$, which is given by

$$W_{\mathcal{P}}(q) = inf\{n : q_0, q_1, ..., q_n, q_0 = q, q \in Q \backslash F, q_n \in F\}$$

where Q is the set of states of the DFA, and F the set of the all final states. However, the *waiting-time* distribution of the $W_{\mathcal{P}}(q)$ random variable can be computed based on the transition probability matrix Π of the $PMC_m^{\mathcal{P}}$, where it has h non-final states and d final states (absorbing states) [4], then the distribution is calculated by the following equation

$$P(W_{\mathcal{P}}(q) = n) = \boldsymbol{\xi_q} \boldsymbol{N}^{n-1} (\boldsymbol{I} - \boldsymbol{N}) \mathbf{1}$$

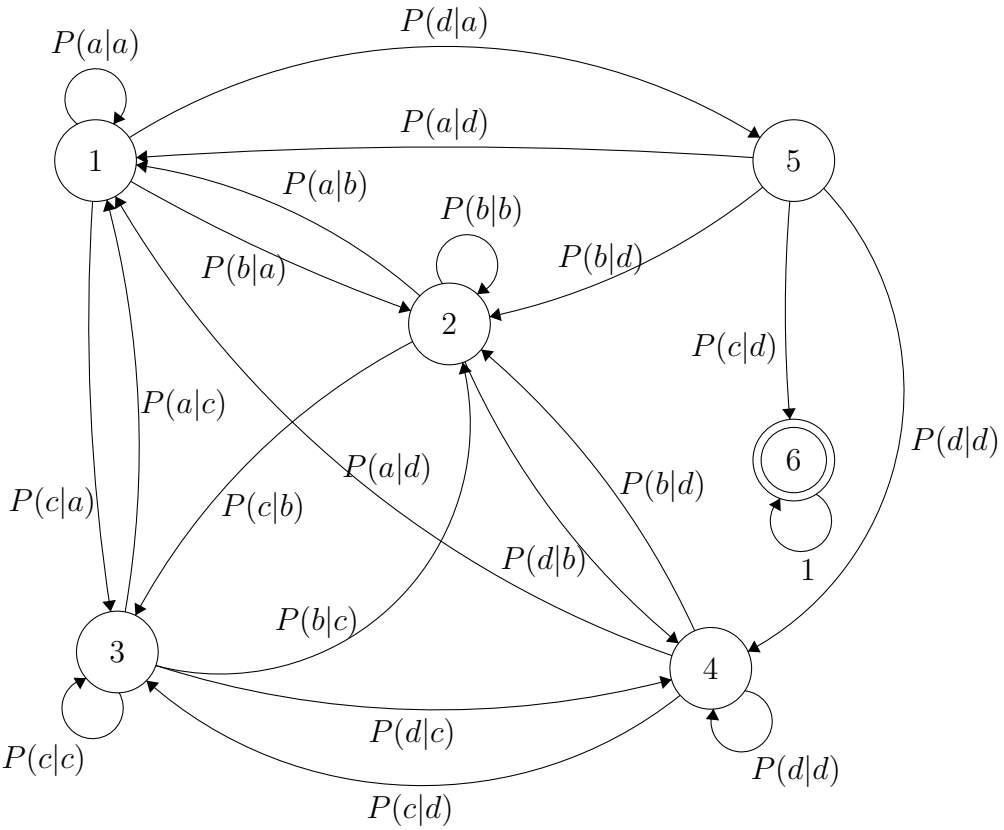

Figure 3.2: $PMC^{1}_{\mathcal{P}}$ for $\mathcal{P} = a; d; c$ with $\Sigma = \{a, b, c, d\}$, and order $m = 1$.

where $\boldsymbol{N}$ is $h \times h$ matrix that obtained by re-arranging the transition matrix Π to include transitions between the non-final states of the DFA, $\boldsymbol{I}$ is an identity matrix of size $d \times d$, and $\mathbf{1}$ is $h \times 1$ vector of ones. The $\boldsymbol{\xi_q}$ is $1 \times h$ row of elements that contains zeros except 1.0 in the cell corresponding to q.

Finally, the model provides the prediction reports in the form of intervals i.e., $I = (start, end)$. Which means that the DFA is expected to reach a final state after future transitions (number of events n) between *start* and *end* with probability $P(I)$ at least some constant threshold θ_p that defined by the user, given by: given by:

$$P(I) = \sum_{n \in I} P(W_{\mathcal{P}}(q) = n) \; where \; P(I) \geq \theta_p$$

where $P(I)$ equals the sum of probabilities of all number of future events n that fall within I ($start \leq n \leq end$). Furthermore, the length of the intervals ($spread(I) = end - start$) can be restricted to not be greater than a maximum spread threshold θ_s.

$$spread(I) \leq \theta_s$$

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 the θ_p and has a spread not greater that θ_s. For example, Figure 3.3 shows the *waiting-time* distributions for all non-final states of the DFA presented in Figure 3.3, and the generated intervals are depicted in Figure 3.4.

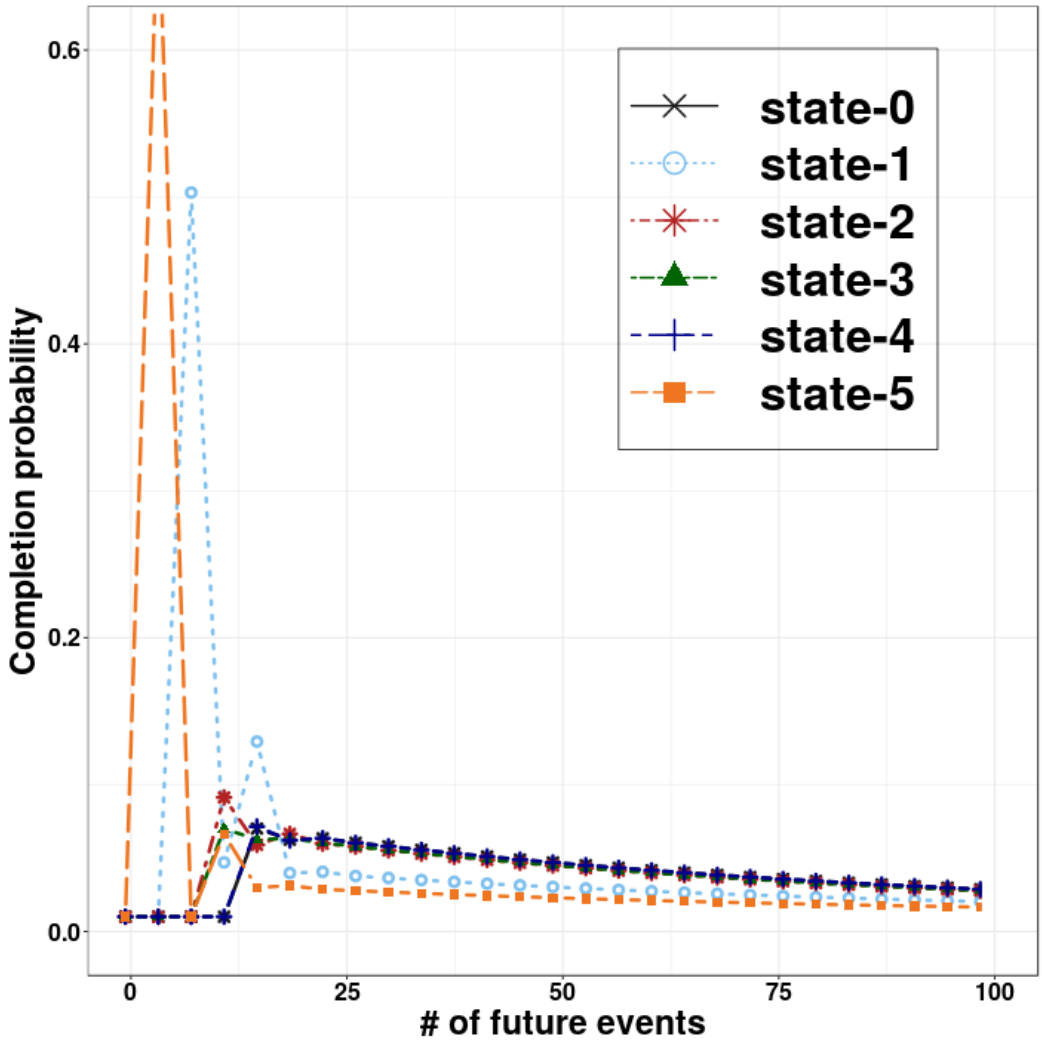

Figure 3.3: Waiting-time distribution for $\mathcal{P} = a; d; c$, $\Sigma = \{a, b, c, d\}$, $m = 1$, $\theta_p = 0.5$ *and* $\theta_s = 20$.

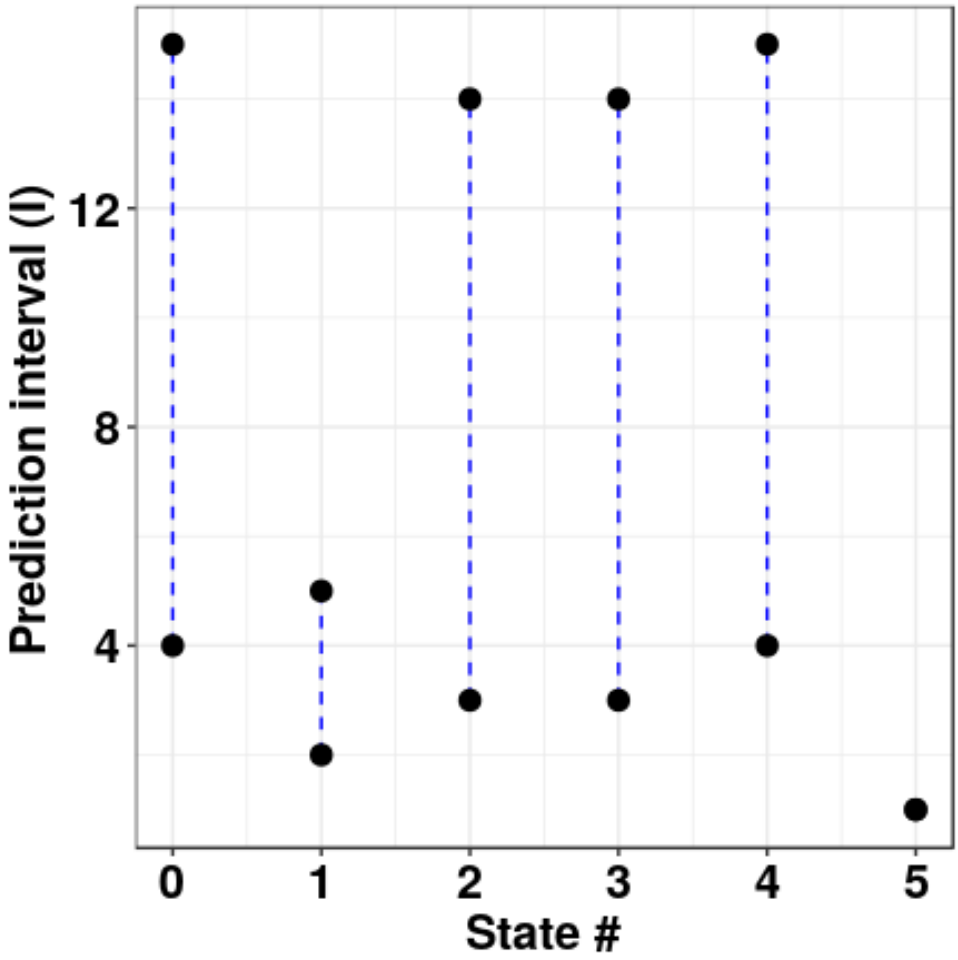

Figure 3.4: Example of the computed prediction intervals for $\mathcal{P} = a; d; c$, $\Sigma = \{a, b, c, d\}$, $m = 1$, $\theta_p = 0.5$ *and* $\theta_s = 20$.

The proposed method assumes that the transition probability matrix Π is available to build the prediction intervals. However, this is not true in the real-world applications. Therefore, it is essential to learn the values of the $PMC_m^{\mathcal{P}}$'s transition probability matrix in order apply this method. One common way, is to use the maximum-likelihood estimator to learn the transition probabilities as illustrated in Section 3.3.2.

This model is performing the learning over an event stream s, which is usually has a slow convergence to a sufficiently good model. Where in the case of multiple event streams, a unique model associated with each stream is needed.

Accordingly, we present in this work, a technique to share information among the $PMC_m^{\mathcal{P}}$ predictors over multiple input event streams (i.e., distributed learning of the transition probability matrix).

3.2 Pattern Prediction over Multiple Event Streams

We extend the pattern prediction problem over a single event stream to the case of multiple distributed input event streams, where there is an associated prediction model for each event stream. We further present our proposed cooperative learning based approach by using a distributed online prediction protocol [26] to synchronize the distributed prediction models in a communication-efficient manner.

3.2.1 Extended Problem Formulation

In this section, we extend the problem of pattern prediction over a single event stream to consider the setting where there are multiple input event streams. Let $O = \{o_1, ..., o_k\}$ be a set of K objects/entities (e.g., moving objects), each of which is generating an event stream s_i, hence, we have a set of real-time input event streams $S = \{s_1, ..., s_k\}$, where these streams are produced independently from the same distribution.

$\mathcal{P}$ is a user-defined pattern given in the form of regular expression, which is monitored in every stream $s_i \in S$. Then, we have a system consists of K local predictor (learner) nodes $n_1, n_2 \ldots, n_k$, each of which receives an input event stream $s_i \in S$ that is associated with a single object $o_i \in O$.

The goal is to provide online predictions about the completion of the pattern $\mathcal{P}$ within each stream s_i. Each node maintains a local prediction model f_i. Then for each new arriving event tuple $e_t \in s_i$, the f_i model provides an online prediction interval about the future full match of the pattern $\mathcal{P}$.

In other words, we have separated running instances of an online prediction algorithm on distributed nodes for K input event streams. For example, massive streams of events that describe trajectories of group of moving vessels in the context of maritime monitoring, where there is one predictor node for each vessel's event stream.

In this work, we employ the $PMC_m^{\mathcal{P}}$ models on the distributed predictor nodes, each model is associated with a particular event stream. While these predictors can work in isolation with no communication with other local predictors, such non-cooperative way leads to inefficiency in terms of learning convergence and predictive performance among all predictors. Thus, in this work, we propose to

use the distributed online prediction protocol [26] to enable the communication among the local predictors to synchronize their models.

3.2.2 The Proposed Method

This work proposes a scalable and distributed prediction system for user-defined patterns over multiple massive input event streams. The system uses the PMC forecasting method [4] as the base prediction model, where there is a $PMC_m^{\mathcal{P}}$ model associated with each input event stream s_i on a predictor node n_i. And it employs the distributed online learning protocol [26] to exchange information among the distributed predictors of the input event streams, which allows to synchronize the parameters of their prediction models, which are represented by the transition matrices.

Distributed Online Learning for Pattern Prediction

In next, we describe the details of adapting the distributed online learning protocol [26] with the pattern prediction ($PMC_m^{\mathcal{P}}$) models. Where this protocol provides a communication-efficient dynamic synchronization scheme based on a periodic static scheme.

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 w_i, which represents the parameters of the local prediction model f_i.

That is, when a predictor $n_i : \ i \in [k]$ observes an event e_j it revises its internal model state (w_i) and provides a prediction report. Then it checks the local conditions, by checking the local model divergence from a reference model w_r after consuming a predefined number of events (batch size b), to decide whether there is a need to synchronize its local model with the coordinator [or not]. w_r is maintained in the predictor node as a copy of the last computed aggregated model $\hat{w}$ from the previous full synchronization step, which is shared among all distributed predictors.

By monitoring the local condition $\|w_i - w_r\|^2 > \Delta$ 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 $\delta(w) = \frac{1}{k}\sum_{j=1}^{k} \|w_i - \hat{w}\|^2$) does not exceed the threshold Δ [26]. This is the key point of this protocol that reduces the communication overhead comparing to the static based communication scheme, where the predictors always communicate periodically after observing a fixed number (b) of events [15].

On the other hand, the coordinator receives the parameters of prediction models

from the predictor nodes that requested for model synchronization (violation). Then it tries to keep incrementally querying other nodes for their local prediction models until reaching out all nodes, or the variance of the aggregated parameters $\hat{w}$ that is computed from the already received models less or equal than the divergence threshold Δ. The coordinator uses a synchronization operation to compute the parameters of a global model $\hat{w}$. Finally, $\hat{w}$ is sent back to the predictor nodes that sent their models after the violation or have been queried by the coordinator.

Algorithm 1: Communication-efficient Distributed Online Learning [26].

Predictor node n_i: at observing event e_j
  update the parameters of the local prediction model w_i and provide a prediction interval I ;
  if $j \mod b = 0$ *and* $\|w_i - w_r\|^2 > \Delta$ **then**
    send w_i to the Coordinator (violation) ;

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

We use this protocol for the pattern prediction models, which are internally based on the $PMC_m^{\mathcal{P}}$. This allows the distributed $PMC_m^{\mathcal{P}}$ predictors for multiple event streams to synchronize their models (i.e., the transition probability matrix of each predictor), and learn a shared model in a communication-efficient way.

Learning a Global Transition Probability Matrix

We propose a *synchronization operation* for the transition probability matrices ($w_i = \mathbf{\Pi}_i : i \in [k]$) of the k distributed $PMC_m^{\mathcal{P}}$ predictors. The operation is based on distributing the maximum-likelihood estimation [5] for the transition probabilities of the $PMC_m^{\mathcal{P}}$ models described by:

$$\hat{\pi}_{i,j} = \frac{\sum_{k \in K} n_{k,i,j}}{\sum_{k \in K} \sum_{l \in L} n_{k,i,l}} \tag{3.1}$$

Where $\hat{\Pi}$ is the global transition probability matrix that will be shared among all $PMC_m^{\mathcal{P}}$ predictors.

Moreover, we measure the divergence of the local models from the reference model $\|w_i - w_r\|^2$ by calculating the sum of square difference between the transition probabilities $\mathbf{\Pi}_i$ and $\mathbf{\Pi}_r$:

$$\|w_i - w_r\|^2 = \sum_{l,j} (\hat{\pi}_i l, j - \hat{\pi}_r l, j)^2$$

3.3 Analysis of the Proposed Approach

3.3.1 General Overview

Generally, our approach relies on enabling the collaborative learning among the distributed predictors. Each predictor node receives a stream of events related to a distinct moving object, and the central coordinator is responsible of synchronizing their prediction models using the *synchronization operation*. Moreover, the predictors only need to share the parameters of their models without aggregating all input event streams.

In addition, 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 this assumption is reasonable in many application domains: for example, in the context of maritime surveillance, vessels travel through standard routes, defined by the 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 [41, 30]. That allows our system to construct a coherent global prediction model dynamically for all input event streams based on merging their local prediction models.

The underlying learning process is based on estimating the transition probability matrix of the $PMC_m^{\mathcal{P}}$ models, where the maximum-likelihood estimator [5] is used to learn the transition probability matrix for each $PMC_m^{\mathcal{P}}$. In our approach, we propose a synchronization operation (Equation 3.1) to build a global estimation of the transition probability matrix among the K predictors. In next, we present the maximum-likelihood estimator, and we provide a theoretical analysis of the proposed operation, where we further present a probabilistic guarantee of the efficiency of our distributed learning estimation against the isolated estimations.

3.3.2 Theoretical Analysis

In this section, we present preliminaries of the Markov chain models and the maximum-likelihood estimator of the transition probabilities, and we describe the theoretical properties of our proposed synchronization operator and its relation with the maximum-likelihood estimator. Also, we derive a probabilistic guarantee of our method estimations for the transition probabilities.

Preliminaries

We first give a brief introduction to the Markov chain theory, where the theoretical definitions and notations presented in this section are based on the work described in [9, 10, 5, 25].

Definition 5. *Let* $\{q_0, q_1, \dots q_n\}$ *be a sequence of random variables as* ***Markov chain****, where the variable* q_i *belongs to a finite state space* $\mathbf{S} = \{\mathbf{1}, \dots \mathbf{m}\}$ *and represents the observed state of the chain at time* i*. Let the transition probabilities of the Markov chain* $p_{ij}(t+1)$ *such that* $i, j \in S$ *and* $t = 0, \dots, n$*, where* $p_{ij}(t+1)$ *is the probability of the state* j *at time* $t+1$*, given state* i *at time* t*, then the sequence* $\{q_0, q_1, \dots q_n\}$ *satisfies the* ***Markov property***

$$P(q_{t+1} = j | q_t = i, q_{t-1} = i_{t-1}, \dots, q_0 = i_0) = P(q_{t+1} = j | q_t = i) \quad \forall i, j, i_{t-1}, i_0 \in S \tag{3.2}$$

That is, the probability of moving to a future state only depends on the current state (Markov chain of order 1). While for higher order m Markov chains the conditional probabilities are modeled to be dependent on the last m states.

When the conditional probabilities $P(q_{t+1} = j | q_t = i)$ are independent of the time t, the Markov chain is called **homogeneous** such that $p_{ij} := P(q_{t+1} = j | q_t = i)$.

The transition probabilities of the Markov chain are represented by a $m \times m$ matrix that is called **transition probability matrix $\mathbf{\Pi}$** with p_{ij} elements

$$\mathbf{\Pi} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdot & \cdot & \cdot & p_{1,m} \\ p_{2,1} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{m,1} & p_{m,1} & \cdot & \cdot & \cdot & p_{m,m} \end{bmatrix} \tag{3.3}$$

where $0 \leq p_{i,j} \leq 1$ and the rows sum up to one

$$\sum_{j=1}^{m} p_{i,j} = 1 \qquad i = 1, 2 \dots m \tag{3.4}$$

3.3.3 Learning the Transition Probability Matrix for a Single Markov Chain

As mentioned in Section 3.1.2, we rely on the transition probability matrix of $PMC_m^{\mathcal{P}}$ to build the prediction intervals, given that the sequence of the states $\{q_0, q_1, \dots q_n\}$ that DFA visits after consuming the input event stream is a 1-order Markov chain.

However, in practice the underlying transition probability matrix is unknown. Thus, it is required to estimate or learn it form the observed sequence $\{q_0, q_1, \dots q_n\}$. The maximum-likelihood estimator is a common method to estimate the transition probability matrix [5].

Definition 6. *Let* $\mathbf{\Pi}$ *is the transition probability matrix of a single Markov chain with a set of states* S, $\pi_{i,j}$ *the transition probability from state* i *to state* j, $n_{i,j}$ *the number of observed transitions from state* i *to state* j, *then the maximum-likelihood finds* $\hat{\mathbf{\Pi}}$ *as an estimate for* $\mathbf{\Pi}$, *where its elements* $\hat{p}_{i,j}$ *are*

$$\hat{p}_{i,j} = \frac{n_{i,j}}{\sum_{l \in S} n_{i,l}} = \frac{n_{i,j}}{n_i} \tag{3.5}$$

Where this estimator is used separately to learn each transition probability matrix of all $PMC_m^{\mathcal{P}}$ models.

Proprieties of the maximum-likelihood estimates. The maximum likelihood estimates of transition probabilities of a single sequence $\{q_0, q_1, \dots q_n\}$ are obtained based on the observed transitions between the states of the chain. That is, the maximum likelihood estimates are basically the count of transitions from i to j divided by the total count of the chain being in state i.

Anderson and Goodman [5] have shown that

$$\sqrt{n}\ (\hat{p}_{i,j} - p_{i,j}) \xrightarrow{d} \mathcal{N}(\mu,\, \sigma^2) \qquad as\ n \to \infty \tag{3.6}$$

Thus, the random variable $\sqrt{n}\ (\hat{p}_{i,j} - p_{i,j})$ has asymptotically normal distribution

with mean $\mu = 0$, and variance σ^2 is given by

$$\sigma^2 = \text{Var}(\sqrt{n}\ (\hat{p}_{i,j} - p_{i,j})) = \frac{p_{i,j}\ (1 - p_{i,j})}{\phi_i}$$
$$\text{s.t. } \phi_i = \sum_{l=1}^{m} \sum_{t=1}^{n} \eta_l\ p_{l,j}^{t-1} \tag{3.7}$$

Where $p_{l,j}^{t-1}$ is the probability of state j at time $t-1$ given that the state l at time 0 [5]. We are interested in the variances of $(\hat{p}_{i,j} - p_{i,j})$ that represents the error in the estimation of $p_{i,j}$ by the maximum-likelihood estimator, which is given by:

$$\text{Var}(\hat{p}_{i,j} - p_{i,j}) = \frac{\text{Var}(\sqrt{n}\ (\hat{p}_{i,j} - p_{i,j}))}{n} = \frac{\sigma^2}{n} \tag{3.8}$$

It can be easily observed that variances are dropping as the sample size n grows large. In next, we will show that our proposed approach of synchronizing the maximum likelihood estimators over k chains is preserving a similar asymptotic behavior and gives efficient estimates of the transition probabilities.

3.3.3.1 Probabilistic Learning Guarantee

The proposed synchronization operator aggregates the maximum-likelihood estimates over k observed sequences (i.e., sequences of the DFA states based on the consumed event streams), the operator estimates a global transition probabilities matrix for a set of k sequences, which are arranged in serial order as one large chain with length $N = k * n$, where we assume that all k sequences have n observations. For the sake of simplicity, we assume that the synchronization phase happens on batch size equals n (i.e., $b = n$) given that the Δ is zero, then the global transition probabilities $\hat{\pi}_{i,j}$ are given

$$\hat{\pi}_{i,j} = \frac{\sum_{k \in K} n_{k,i,j}}{\sum_{k \in K} \sum_{l \in L} n_{k,i,l}} = \hat{p}_{i,j}(N)$$
$$where\ N = k * n. \tag{3.9}$$

Thus, this operation allows to observe more samples, which is naturally producing better estimates of the transition probabilities. In addition, our proposed synchronization operation of the k transition matrices has the same proprieties as the maximum likelihood estimator over a serial sequence of all k sequences, but with skipping $k-1$ transitions between every two consecutive sequences, which is

in practice a small number that can be neglected comparing to the total transitions count $k * n$. As result, the global transition probabilities of our approach have the same properties as maximum likelihood estimates, in particular, the the random variable $\sqrt{N}\ (\hat{\pi}_{i,j} - p_{i,j})$ has asymptotically normal distribution with mean $\mu = 0$, and following Equation 3.6 we have variance

$$\begin{aligned} \sqrt{N}\ (\hat{\pi}_{i,j} - p_{i,j}) &\xrightarrow{d} \mathcal{N}(0,\, \sigma^2) \\ as\ N &\to \infty \\ where\ N &= n * k. \end{aligned} \tag{3.10}$$

So,

$$\mathrm{Var}(\hat{\pi}_{i,j} - p_{i,j}) = \frac{\sigma^2}{N} = \frac{\sigma^2}{k * n} \tag{3.11}$$

That is, since $N > n$ combining k sequences, the variances of estimations of our method $\mathrm{Var}(\hat{\pi}_{i,j} - p_{i,j})$ are smaller than the estimates of maximum-likelihood over a single sequence $\mathrm{Var}(\hat{p}_{i,j} - p_{i,j})$.

Thus, it follows from the Chebyshev's inequality [17] that we have for the random variable $\hat{p}_{i,j} - p_{i,j}$, for any constant $c > 0$

$$\Pr\left(|(\hat{p}_{i,j} - p_{i,j}) - \mu| \geq c\right) \leq \frac{\mathrm{Var}(\hat{p}_{i,j} - p_{i,j})}{c^2}$$

The mean $\mu = 0$ is zero and the $\mathrm{Var}(\hat{p}_{i,j} - p_{i,j})$ equals $\frac{\sigma^2}{n}$, and therefore

$$\Pr\left(|\hat{p}_{i,j} - p_{i,j}| \geq c\right) \leq \frac{\sigma^2}{c^2 * n}$$

$\hat{p}_{i,j} - p_{i,j}$ represents the deviation/error between the estimates of maximum-likelihood over a single (i.e., isolated) sequence and the true probabilities. On the other hand, we can obtain, in the same way, the probability bound of deviations for our synchronization operator estimates as follows:

$$\Pr\left(|\hat{\pi}_{i,j} - p_{i,j}| \geq c\right) \leq \frac{\sigma^2}{c^2 * n * k}$$

Using Equation 3.11 we obtain the value $\mathrm{Var}(\hat{\pi}_{i,j} - p_{i,j})$, and Equation 3.11 for $\mathrm{Var}(\hat{p}_{i,j} - p_{i,j})$. Given that $k \geq 1$, then the variance of $\hat{\pi}_{i,j} - p_{i,j}$ is less than or equal to the variance of $\hat{p}_{i,j} - p_{i,j}$

$$\frac{\sigma^2}{c^2 * n * k} \leq \frac{\sigma^2}{c^2 * n}$$

So, we have for any constant $c > 0$ and $k \geq 1$, a probabilistic learning guarantee of the transition probabilities:

$$\Pr\left(|\hat{\pi}_{i,j} - p_{i,j}| \geq c\right) \leq \Pr\left(|\hat{p}_{i,j} - p_{i,j}| \geq c\right)$$

where $\hat{p}_{i,j}$ a transition probability, which is learned on a single $PMC_m^{\mathcal{P}}$ predictor, and the $\hat{\pi}_{i,j}$ is the global transition probability that computed by the synchronization operation (Equation 3.1). To summarize, our approach is based on aggregating the maximum-likelihood estimates over k sequences, which speeds up the convergence to reach the true transition probabilities as result of the smaller variances of the estimates.

3.3.4 Transition Matrix of the Underlaying Markov Chain

In order to empirically study the learning efficiency of our proposed approach, we need to map the transition probabilities of the underlying Markov chain with the transition probability matrix ($\hat{\Pi}$) of the $PMC_m^{\mathcal{P}}$ model. Nuel [39] showed in **Theorem 3** the relation between the elements of Π and the conditional probabilities of the $m-$order Markov chain $X = \{X_1, X_2, \ldots X_n\}$ described by

$$\Pi(p,q) = \begin{cases} P(X_{m+1} = b | X_1 \ldots X_m = \delta^{-m}(p)) & \text{if } \delta(p,q) = b \\ 0 & \text{if } p \notin \delta(p, X) \end{cases}$$

Using this theorem, in the case of the underlying transition probability matrix is known (e.g., synthetic event streams), we then can compare the estimation efficiency of the different approaches for the transition probabilities of the underlying Markov chain X. For example, the transition probability matrix of the $PMC_m^{\mathcal{P}}$ model for the pattern $\mathcal{P} = a; d; c$ over $\Sigma = \{a, b, c, d\}$ is represented by:

$$\mathbf{\Pi} = \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{Bmatrix} \begin{bmatrix} P(a|a) & P(b|a) & P(c|a) & 0 & p(d|a) & 0 \\ P(a|b) & P(b|b) & P(c|b) & P(d|b) & 0 & 0 \\ P(a|c) & P(b|c) & P(c|c) & P(d|c) & 0 & 0 \\ P(a|d) & P(b|d) & P(c|d) & P(d|d) & 0 & 0 \\ P(a|d) & P(b|d) & 0 & P(d|d) & 0 & P(c|d) \\ 0 & 0 & 0 & 0 & 0 & 1.0 \end{bmatrix} \tag{3.12}$$

4 System Overview

In this chapter, we present the architecture of the proposed system for pattern prediction over multiple input event streams in a scalable distributed setting. To build a scalable and distributed system, we implement the system on top of Apache Flink engine for distributed and large-scale stream processing. We also use Apache Kafka for the distributed communication and to ingest and emit the input/output data streams. In what follows we describe the building blocks of the proposed system and the important aspects of the implementation.

4.1 System Architecture

The input to our system is an aggregated stream of events coming from a large number of objects, which is continuously collected and fed into the system. In practice, the aggregated input events stream is composed of multiple event streams (partitions) from a set of objects. Our system allows the users to register a pattern $\mathcal{P}$ to be monitored over each event stream. The output stream consists of original input events along with the prediction intervals of the full matches of monitored pattern $\mathcal{P}$, which are displayed to the end users. Figure 4.1 presents the overview of our system architecture and its main components.

The system is composed of three main processing elements:

- The pre-processing operators: Operators receive the input event stream and perform filtering and ordering operations, before partitioning the input event stream to multiple event streams based on the associated object.

- The predictor nodes (learners): Each predictor is responsible for maintaining a prediction model for an input event stream. Each prediction node is configured to handle an event stream from the same object, in order to provide online predictions for the user-defined pattern $\mathcal{P}$.

- The coordinator node: A central node communicates through Kafka stream channels with the predictors to realize the distributed online learning protocol. It builds a global prediction model, based on the received local models, and then shares it among the predictors.

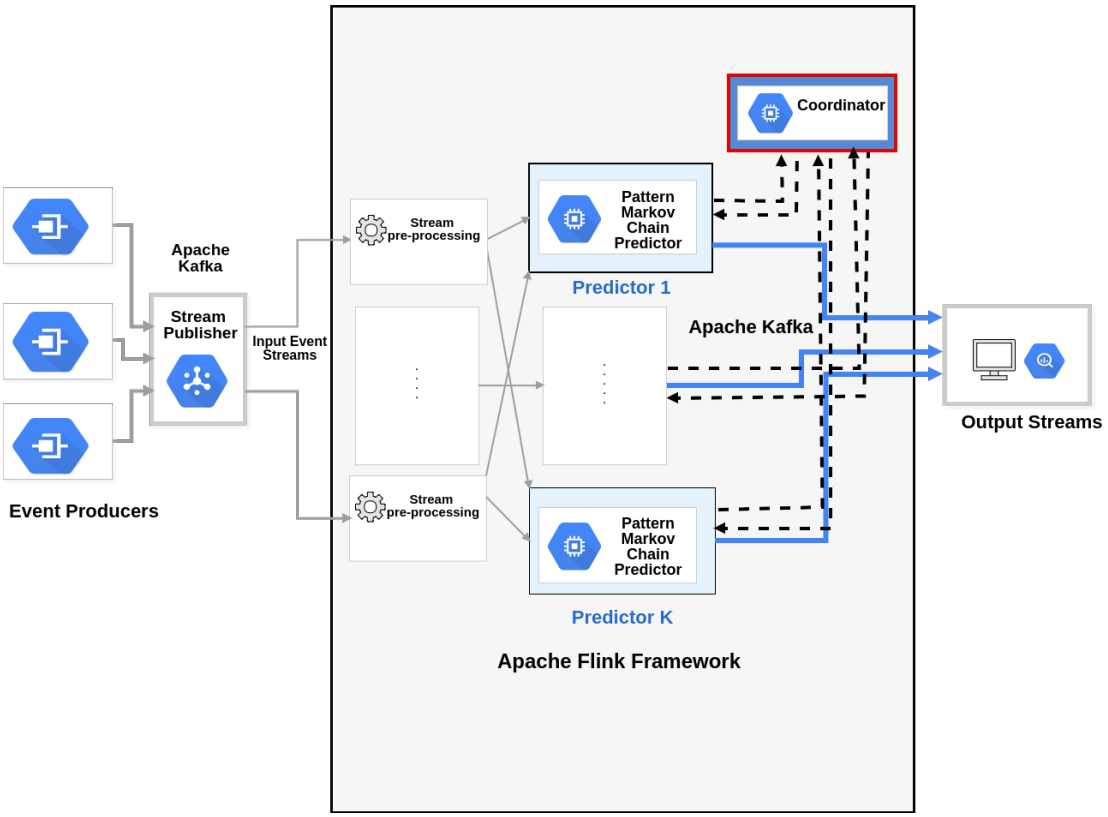

Figure 4.1: System architecture overview.

Our distributed system consists of multiple pre-processing operators, prediction nodes, and a central coordinator node. All units run in parallel and are arranged as a data processing pipeline, depicted in Figure 4.1. We leverage Apache Kafka as a messaging platform to ingest the input event streams and to publish the resulting streams. Also, it is used as the communication channel between the predictor nodes and the coordinator. Apache Flink is employed to execute the system's distributed processing units over the input event streams, namely, the pre-processing operators, the prediction units, and the coordinator node. Our system architecture can be modeled as a logical network of processing nodes, organized in the form of a DAG, inspired by the Flink runtime dataflow programs [12].

4.2 Implementation Details

In this section, we describe in detail the implementation of our system. It has been implemented on top of Apache Flink and Apache Kafka frameworks. Each of the three sub-modules, described in Section 4.1, have been implemented as Flink operations over the input events stream.

Pre-processing and Prediction Operators. Listing 4.1 shows how the main workflow of the system is implemented as Flink data flow program.

The system ingests the input events stream from a Kafka cluster that is mapped to a *DataStream* of events, which is then processed by an *EventTuplesMapper* to create tuples of *(id, event)*, where the *id* is associated to the identifier of the moving object. To handle events coming in out of order in a certain margin, the stream of event tuples is processed by a *TimestampAssigner*, it assigns the timestamps for the input events based on the extracted creation time. Afterwards, an ordered stream of event tuples is generated using a process function *EventSorter*.

```
...
DataStream<Event> eventsStream = env.addSource(kafkaConsumer);
// Create event tuples (id,event) and assign time stamp
DataStream<Tuple2<String,Event>> eventTuplesStream =
inputEventsStream.map(new EventTuplesMapper())
.assignTimestampsAndWatermarks(new EventTimeAssigner());
// Create the ordered keyed stream
 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 4.1: Flink pipeline for local predictors workflow

The ordered stream is then transformed to a *keyedEventsStream* by partitioning it, based on the ids values, using a *keyBy* operation. A local *predictor* node in a distributed environment is represented by a *map* function over the *keyedEventsStream*. Each parallel instance of the map operator (predictor) always processes all events of the same moving object (i.e., equivalent id), and maintains a bounded prediction model (i.e., $PMC_m^{\mathcal{P}}$ predictor) using the Flink's Keyed State [1]. The output streams of the moving objects from the parallel instances of the predictor map functions are sent to a new Kafka stream (i.e., same topic name). They then can be processed by other components like visualization or users notifier.

Moreover, the implementation of the *predictor* map function includes the communication with *coordinator* using Kafka streams. Figure 4.2 presents the general

[1] Keyed State in Flink: https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/stream/state.html#kayed-state

class hierarchy of the predictor node. At the beginning of the execution, each predictor instance sends a registration request to the coordinator. Also at the run-time, it sends its local prediction model as synchronization request, or as a response for a resolution request from the coordinator. These coordination messages are published into different Kafka topics as depicted in Table 4.1.

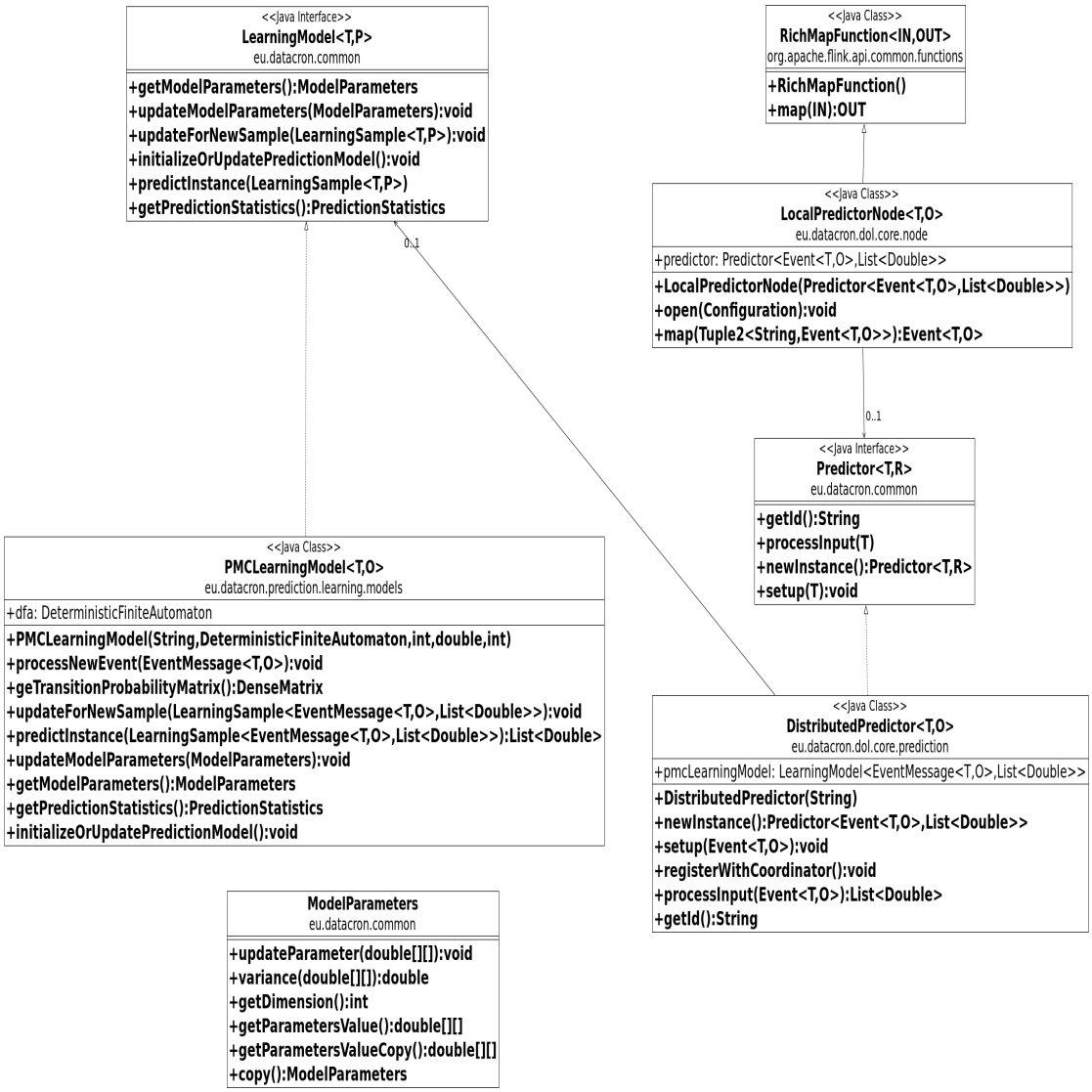

Figure 4.2: Predictor Class Hierarchy.

Coordinator. Listing 4.2 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

Table 4.1: Messages to Kafka topics mapping.

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

through a Kafka Stream of a topic named *LocalToCoordinatorTopicId*. It is implemented as a single *map* function over the messages stream, by setting the *parallelism* level of the Flink program to *1*. Increasing the parallelism will scale up the number of parallel coordinator instances, for example, in order to handle different groupings of the input event streams. The map operator of the coordinator handles three message types from the predictors:

1. **RegisterNode** that contains a registration request for a new predictor node.
2. **RequestSync** to receive a local model after violation.
3. **ResolutionAnswer** to receive a resolution response from a local predictor node.

On the other hand, it sends **CoordinatorSync** messages for all predictors after creating a new global prediction model, or *UpdateReference* to set the reference model. It also send **RequestResolution** messages to a ask the local predictors for their prediction models.

```
 streamExecutionEnvironment.setParallelism(1);
// Read messages from local predictors
DataStream<TopicMessage> messagesStream = readKafkaStream(env,
    "LocalToCoordinatorTopicId");
// Initialize the coordinator node
CoordinatorNode coordinatorNode = new CoordinatorNode(configs);
// Ingest the messages stream by the coordinator
DataStream<CoordinatorMessage> coordinatorMessagesStream =
    messageStream.map(coordinatorNode);
// Send the messages from the coordinator to the local predictors
writeToKafka(coordinatorMessagesStream, CoordinatorToLocalTopicId);
```

Listing 4.2: The coordinator Flink program.

5 Empirical Evaluation

In this chapter, we report and discuss the comprehensive set of experiments to evaluate the performance of our pattern prediction system. We make use of both synthetic and real-word event streams. Section 5.1 provides a brief overview of the used event streams for our evaluation and the experimental environment. Section 5.2 introduces the metrics used for the evaluation. In Section 5.3 and 5.4, the evaluation results of our system over synthetic and real-world event streams are reported. Finally, Section 5.5 discusses the scalability evaluation of the proposed system.

5.1 Setup

Experimental setup

We ran our experiments on a single-node standalone Flink cluster deployed on an Ubuntu Server 17.04 with Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz X 8 processors and 32GB RAM. We used Apache Flink v1.4.0 and Apache Kafka v0.10.2.1 for our experiment.

We also conducted experiments to measure the throughput of our system on a YARN [48] cluster (v2.7.2) with 9 physical nodes, where each node has 10 cores CPU, and 100 GB memory.

Experimental Streams

Real-word Event Streams

We used real-world event streams provided by the datAcron project in the context of maritime monitoring. These event streams describe critical points (i.e., synopses) of moving vessels trajectories, which are derived from raw AIS messages as described in [43]. In particular, for our evaluation experiments we used a data set of synopses that contains $4,684,444$ critical points of 5055 vessels that sailed in the Atlantic Ocean during the period from 1 October 2015 to 31 March 2016.

We used the synopses data set to generate a simulated stream of event tuples i.e., *(id, timestamp, longitude, latitude, annotation, speed, heading)*. The system

further processes the stream to attach an extra event *type* attribute, where *type* $\in \Sigma 1$ or *type* $\in \Sigma 2$.

Σ_1 ={ *VerySlow, Slow, Moving, Sailing, Stopping*}, which is based on a discretization of the speed values. That is, Σ_1 includes a simple derived event types based on the speed value that can be used over streams of raw AIS or critical points. While Σ_2 = {*stopStart, stopEnd, changeInSpeedStart, changeInSpeedEnd, slowMotionStart, slowMotionEnd, gapStart, gapEnd, changeInHeading*}, which is derived based on the values of the *annotation* attribute that encodes the extracted trajectory movement events [43]. Σ_2 represents the set of possible mobility changes in the vessel's trajectory [43], each critical point has at least one event. Where in the case of multiple values we generate duplicate points each of which corresponding to one event in the same order of Σ_2.

In our experiments, we monitor a pattern $\mathcal{P}_1$ = *Sailing* with Σ_1 that detects when the vessel is underway (sailing). Likewise, we test a second pattern $\mathcal{P}_2$ =*changeInHeading; gapStart; gapEnd; changeInHeading* with Σ_2 that describes a potential illegal fishing activity [4].

Synthetic Event Streams

In order to get a better insight of our proposed approach, we evaluate it on synthetic event streams. For our experiments, we generate 20 streams of size 25,000 events from a simulated Markov process over $\Sigma = \{a, b, c, d\}$.

We define a simple pattern $\mathcal{P} = a; d; c$ to study the predictive performance along with the learning efficiency of the transition probability matrix of the Markov chain source. We compare the learned matrix of $PMC_m^{\mathcal{P}}$ models in our approach and the isolated case to the used transition matrix to generate the streams. We map the transition probability of the $PMC_m^{\mathcal{P}}$ and the transition probability matrix of the Markov chain source as was mentioned earlier in Section 3.3.4.

5.2 Evaluation Metrics

We compare our proposed system, which enables the synchronization of prediction models (i.e., $PMC_m^{\mathcal{P}}$ models) on the distributed predictor nodes, against the isolated prediction mode, in which models are computed on single streams only. We compare the performance in terms of following metrics:

- $Precision = \frac{\#\ of\ correct\ predictions}{\#\ of\ total\ predictions}$. The fraction of the predictions that are correct. For each new event in the stream, the predictor provides a prediction interval where the full match of the pattern might occur. Thus, the predictions are temporarily stored until a full match is detected. At that point, all

stored prediction intervals are evaluated by considering those intervals where the full match occurred within as correct.

- *Spread*= $end(I) - start(I)$. The width of the prediction interval I, which represents the number of events between the start and the end of I.

- *Cumulative communication*. We study the communication cost by measuring the *cumulative communication* that captures the number of messages, which are required to perform the distributed online learning modes to synchronize the prediction models.

- $Throughput$. The number of events processed per unit time.

Our proposed system can operate in three different modes of synchronization schemes:

(i) continuous full synchronization for each incoming event (hypothetical).

(ii) static scheme based on synchronizing the prediction models periodically every b of input events in each stream.

(iii) dynamic synchronization protocol based on making the predictors communicate their local prediction models periodically but only under condition that the divergence of the local models from a reference model exceeds a variance threshold Δ (recommended).

5.3 Results on Real-word Event Streams

In this section, we present the experimental results for two different patterns $\mathcal{P}_1 =$ *Sailing* (with an order of)$m = 2$), and $\mathcal{P}_2 =$*changeInHeading; gapStart; gapEnd; changeInHeading* (with first order $m = 1$). All experiments are performed with the batch size of 100 ($b = 100$), the variance threshold of 2 ($\Delta = 2$), 80% as PMC prediction threshold ($\theta_p = 80\%$), and 200 for the maximum spread ($\theta_s = 200$).

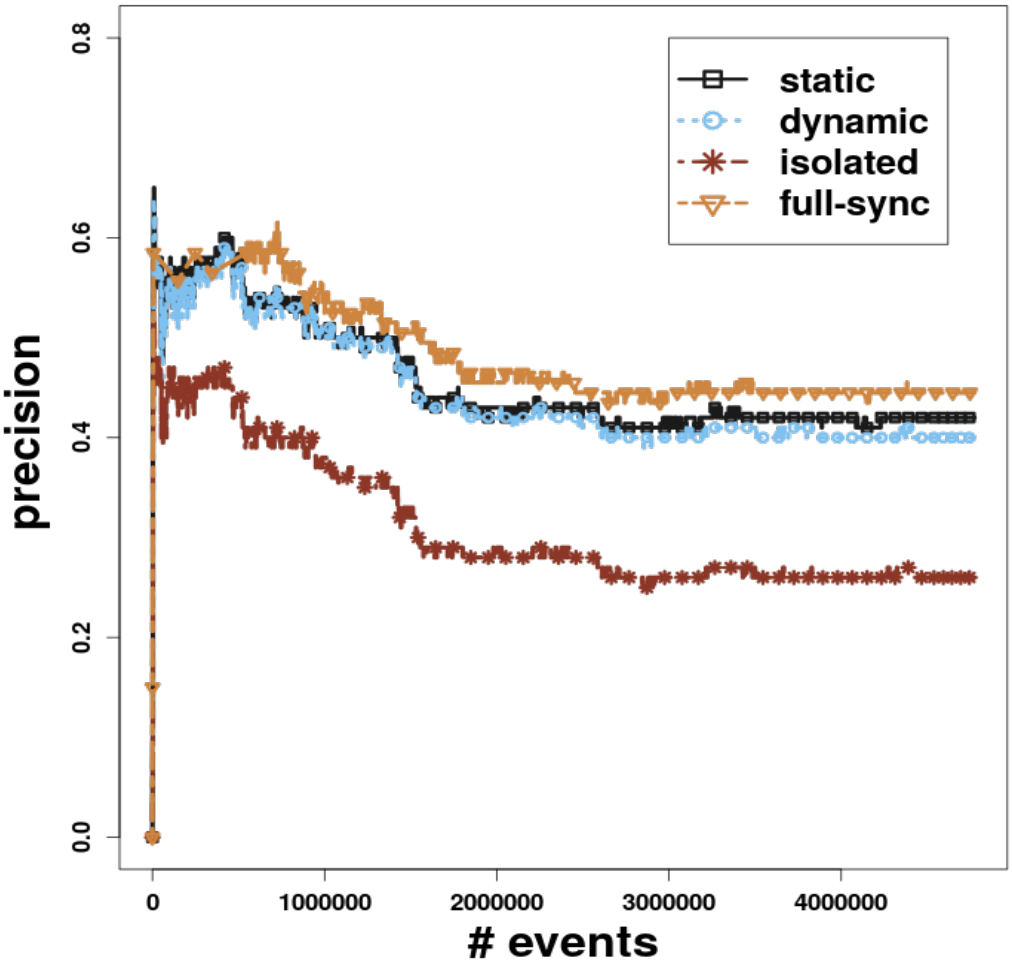

Figure 5.1: Precision scores with respect to the number of input events over time for $\mathcal{P}_1$.

Figure 5.1 depicts the average precision scores of predictions models for the first pattern $\mathcal{P}_1 = \mathit{Sailing}$ (one prediction model per vessel) of all synchronization modes, namely, isolated without synchronization, continuous (full-sync), static, and our recommended approach based on the dynamic synchronization scheme. It can be clearly seen that all methods of distributed learning outperform the isolated prediction models. The full continuous synchronization method has the highest precision rates, while the static and dynamic synchronization schemes have close precision scores. Consequently, dynamic synchronization is not much weaker than the static synchronization, but requires much less communication, as explained below.

Figure 5.2 provides the accumulated communication cost that is required by

the three distributed online learning modes, while the isolated approach does not require any communication at all. These results are shown for $\mathcal{P}_1$. As expected, a larger amount of communication is required for the full synchronization comparing to the static and dynamic approaches. Also, it can be seen that we can reduce the communication overhead by applying the dynamic synchronization protocol (a reduction by a factor of 100) compared to the static synchronization scheme, even with a small variance threshold $\Delta = 2$. Furthermore, the dynamic protocol still preserves a similar predictive performance to the static one as illustrated in Figure 5.1. Therefore, we will only consider the dynamic synchronization and the isolated approach in the evaluation of the second pattern.

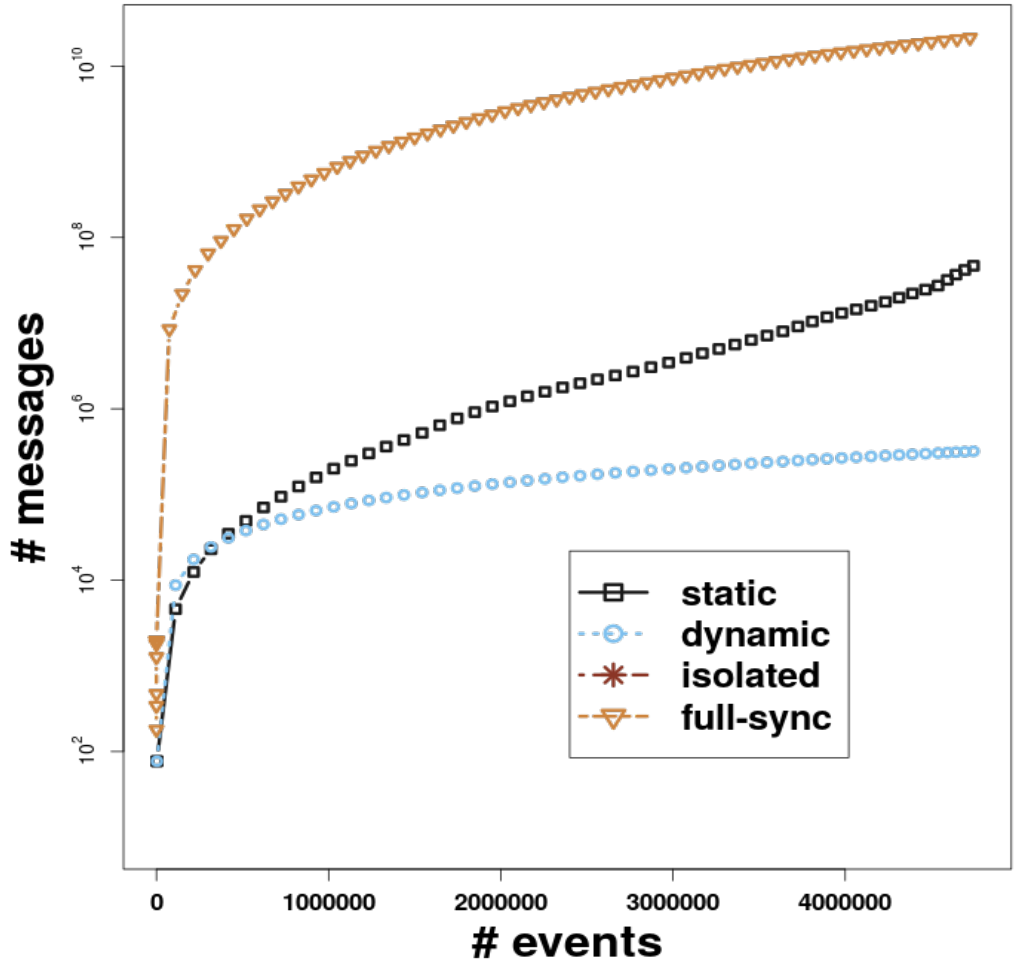

Figure 5.2: Cumulative communication with respect to the number of input events over time for $\mathcal{P}_1$.

In Figure 5.1, we observe that the precision decreases initially and then stabilizes. This seems to be counter-intuitive, as we expect the models to improve up to a certain point as it get more data. We investigated the effect of the distributed synchronization of the prediction models on the average spread value, Figure 5.3 shows the spread results for all approaches. It can be seen that the spread is higher for the distributed learning based methods comparing to the isolated approach. Furthermore, the average spread decreases over time until convergence, as result of confidence increase in the models, which is inherited from the $PMC_m^{\mathcal{P}}$ models.

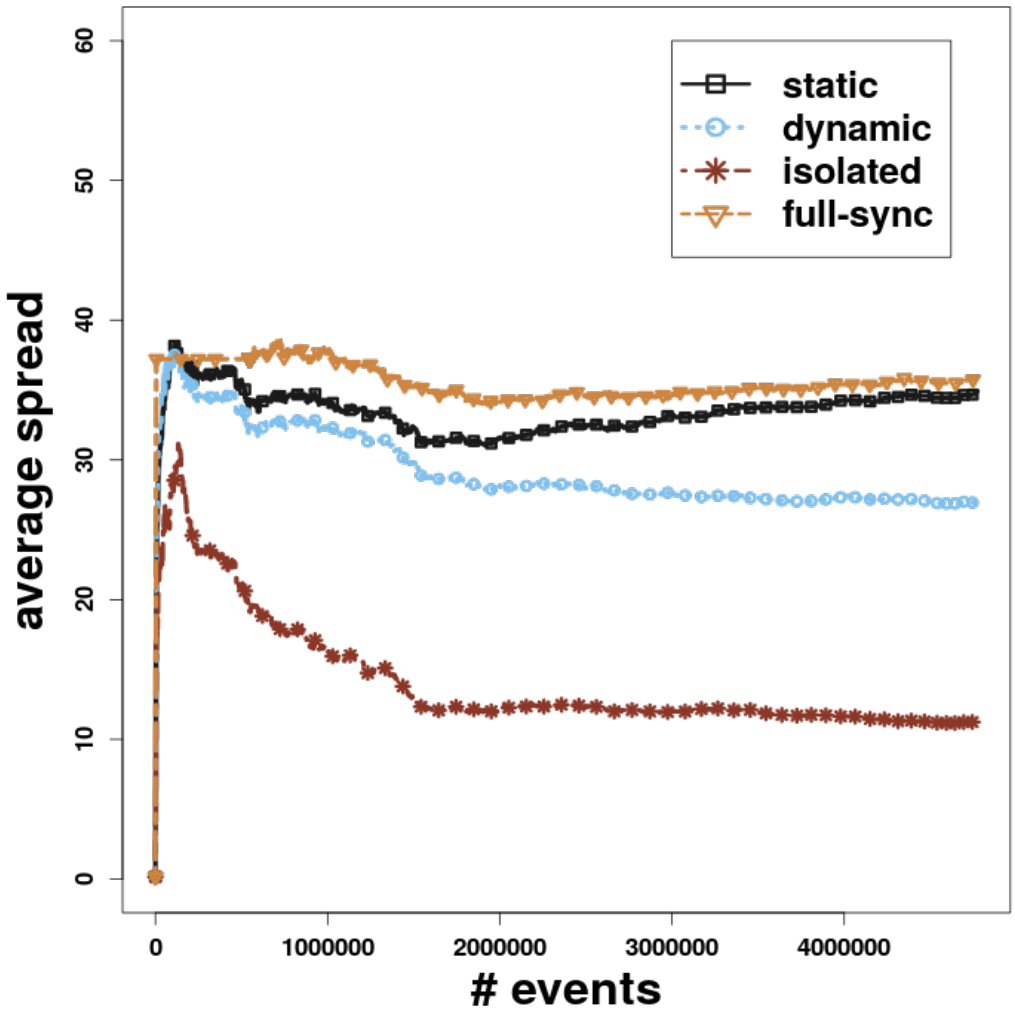

Figure 5.3: Average spread for $\mathcal{P}_1$.

For a clearer picture of the comparison between our distributed learning method and the isolated approach, Figure 5.4 depicts the measurement of a combined metric of the precision and spread. It is clear that our approach achieves better performance in terms of the precision and spread combined.

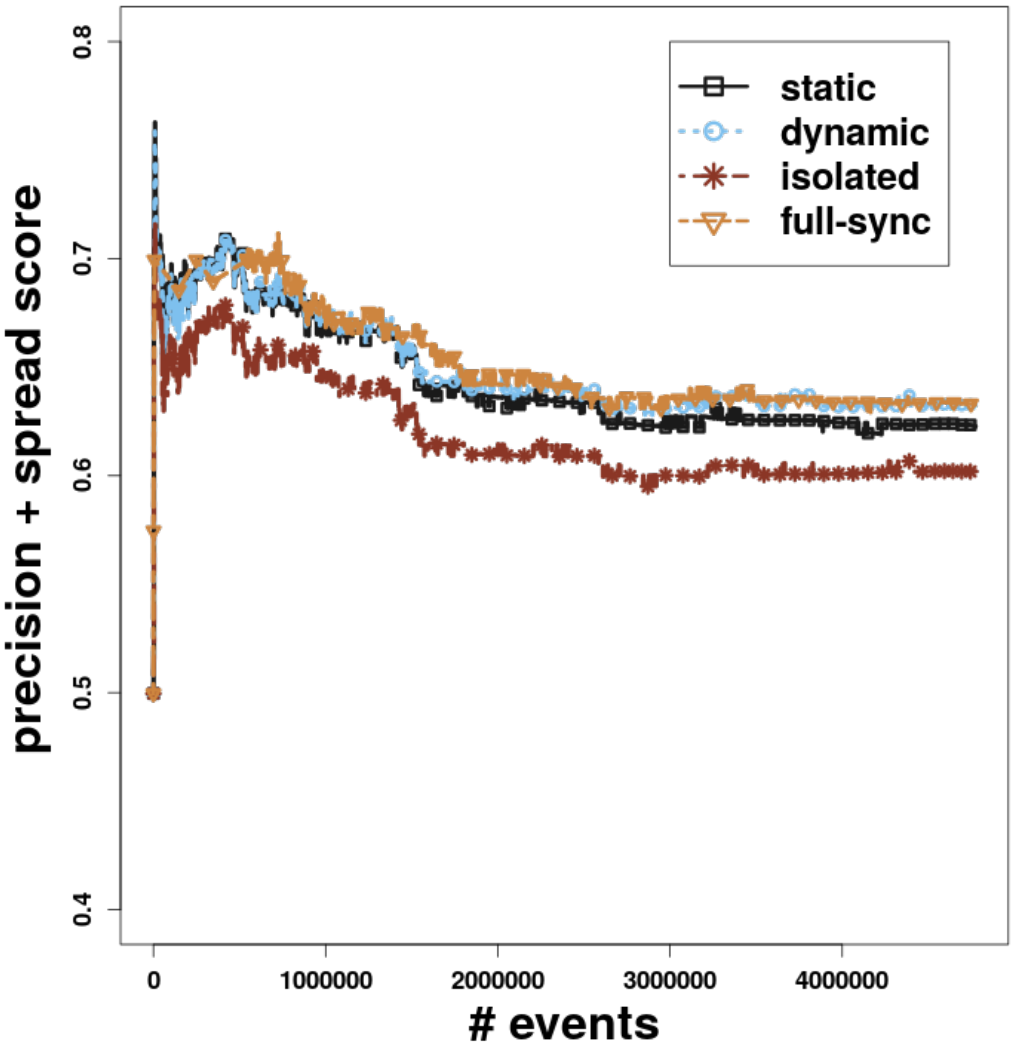

Figure 5.4: $\alpha * precision + (1 - \alpha) * (1 - \frac{spread}{max\ spread})$ for $\mathcal{P}_1$ with $\alpha = .5$.

For the second, more complex pattern ($\mathcal{P}_2$), we have found that the precision was worse for a distributed model generated over all vessels than in the model created for each vessel in isolation. This indicates that there is no global model

describing the behavior of all models consistently. However, when we look at specific groups of vessels, we achieve an improvement in terms of precision. As initial experiment, we only enable the synchronization of the prediction models associated with vessels that belong to the same vessel class. Currently, this change is technically performed by an extra filter step that passes only one type of vessels, while multiple runs of the system are required for all vessel types. For example, Figure 5.5 shows the precision scores for vessels of class *PLEASURE CRAFT*. This case might seem to contradict of our assumption that the input event streams belong to the same distribution and share the same behavior, but it actually follows the same assumption but between the predictors of vessels within the same type group. An interesting observation is that the dynamic synchronization approach still has a higher precision scores than the isolated approach.

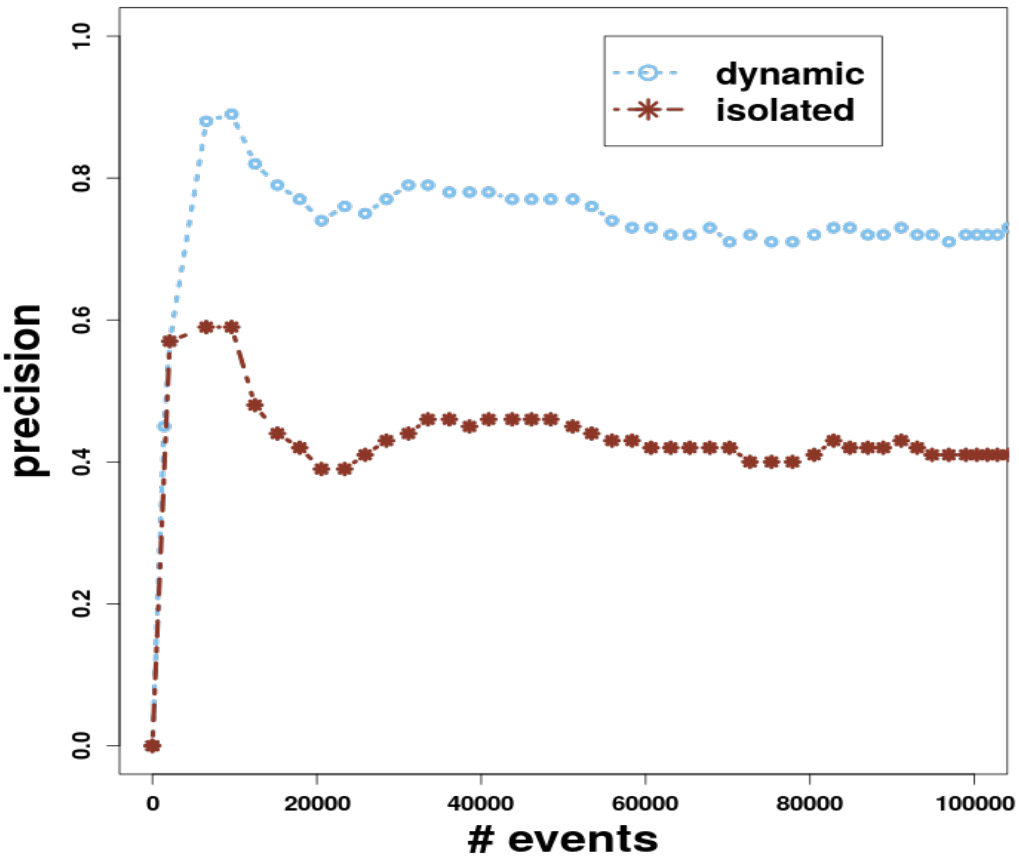

Figure 5.5: Precision scores of $\mathcal{P}_2$ for *PLEASURE CRAFT* vessels.

5.4 Results on Synthetic Event Streams

We further extend the evaluation of our system by reporting the results of experiments for $\mathcal{P} = a; d; c$ over synthetically generated streams. We set the batch size to 15 ($b = 15$), the variance threshold to .0001 ($\Delta = .0001$), the $PMC_m^{\mathcal{P}}$ prediction threshold to 50% ($\theta_p = 50\%$), and the maximum spread to 10 ($\theta_s = 10$).

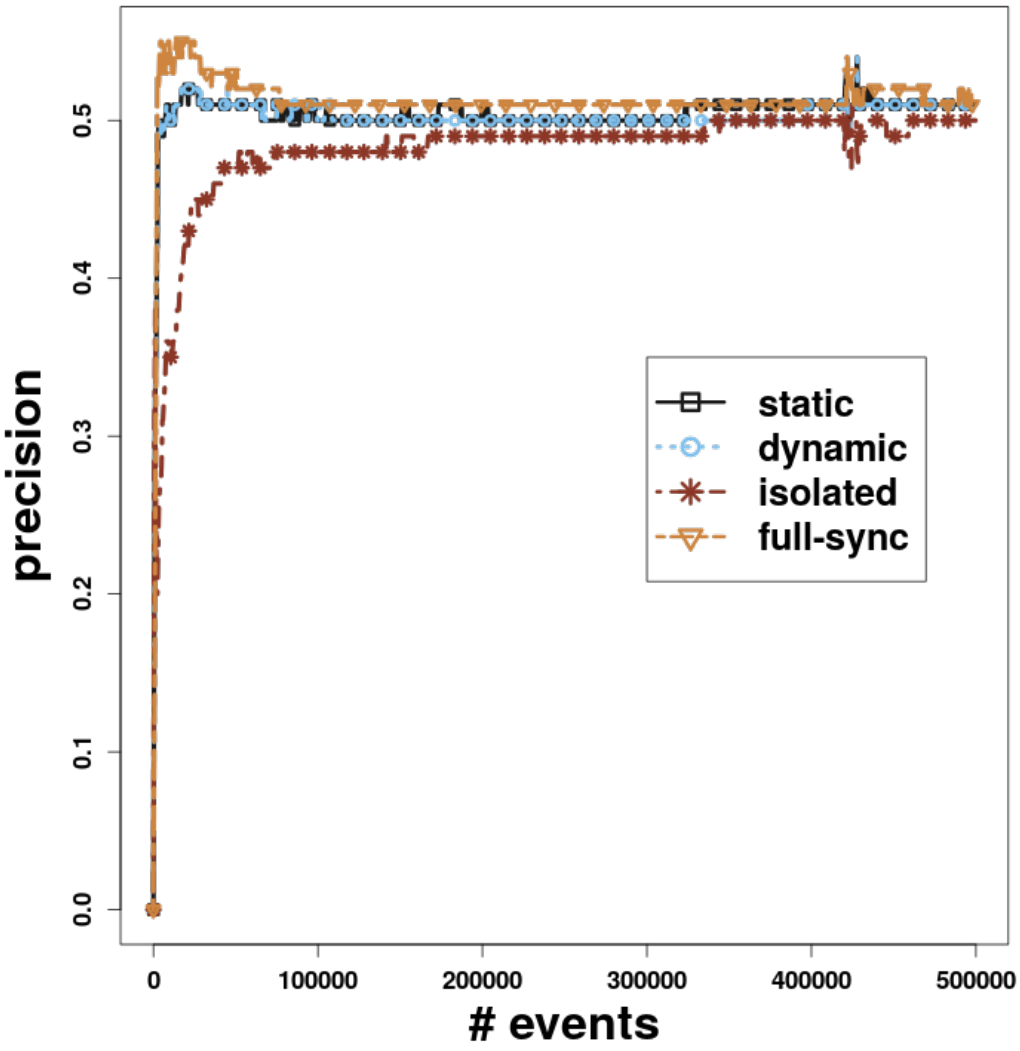

Figure 5.6: Precision scores with respect to the number of input events over time for $\mathcal{P} = a; d; c$.

Figure 5.6 shows the average precision of the three modes of our system and the isolated approach. We can see that our proposed system in all its modes has higher precision scores than the isolated method. Also, it can be seen that the static and

the dynamic approaches have similar performance while the more complex full synchronization mode has a slightly better performance. As expected, we also note that our method converges faster. The isolated models converge to the same accuracy but in much longer time.

On the other hand, Figure 5.7 presents the average spread. As it can be seen, the average spread of the predictions in our approach (three modes) decreases over time. While the isolated approach has a higher spread of the prediction intervals.

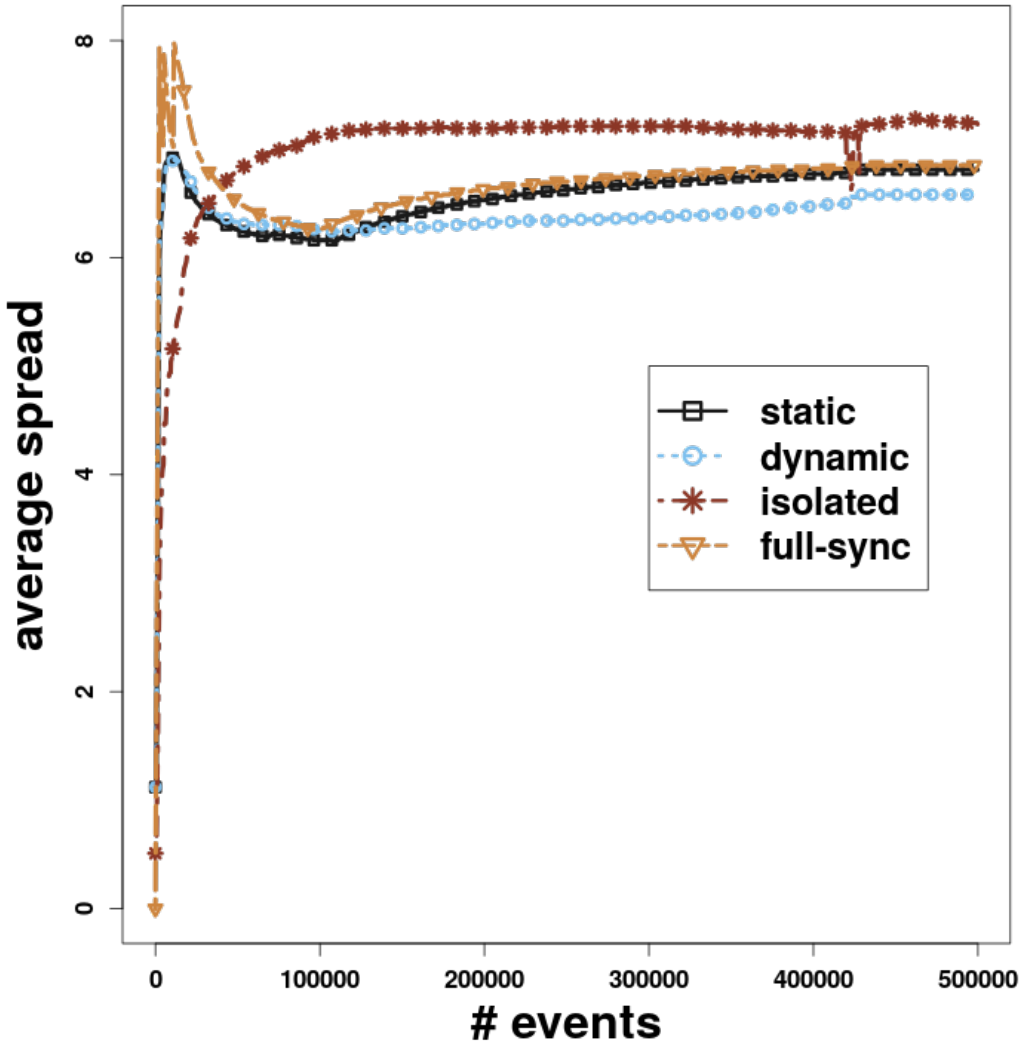

Figure 5.7: Average spread with respect to the number of input events over time for $\mathcal{P} = a; d; c$.

As discussed in Section 3.3.2, our proposed method improves the estimation of the transition probabilities of the underlying Markov chain for the $PMC_m^{\mathcal{P}}$ models.

Figure 5.8 shows the difference between the estimated transition probabilities and the reference probabilities of the Markov chain source ($\sum_{i,j} |\hat{p}_{i,j} - p_{i,j}|$), the results confirm the derived learning guarantee of our proposed approach that gives a better estimation of the transition probability matrix. It can be seen that the distributed modes have less estimation error and are faster in reaching the correct probabilities than the isolated approach.

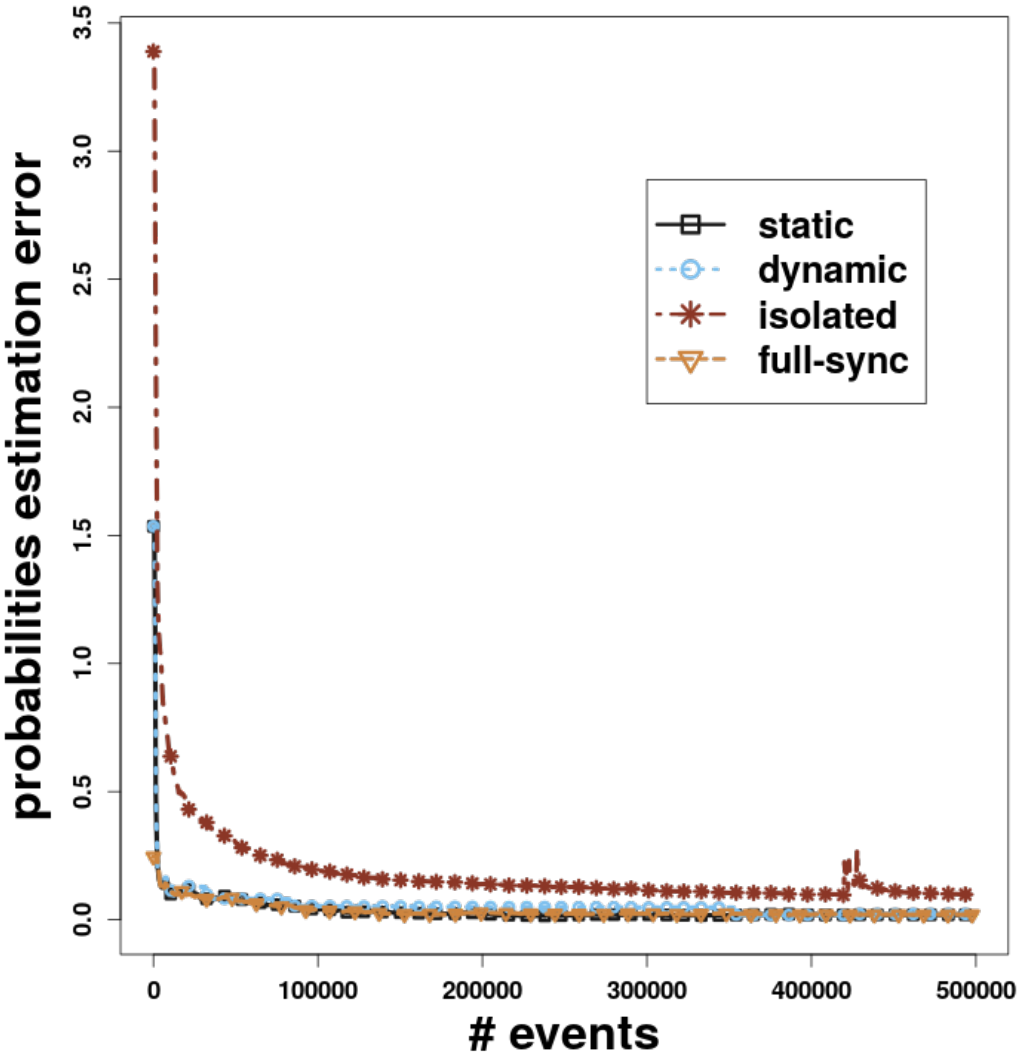

Figure 5.8: The error $\sum_{i,j} |\hat{p}_{i,j} - p_{i,j}|$) of estimating the transition probabilities for $\mathcal{P} = a; d; c$.

5.5 Throughput Results

Figure 4.8 presents the throughput measurements of our system on the YARN cluster.

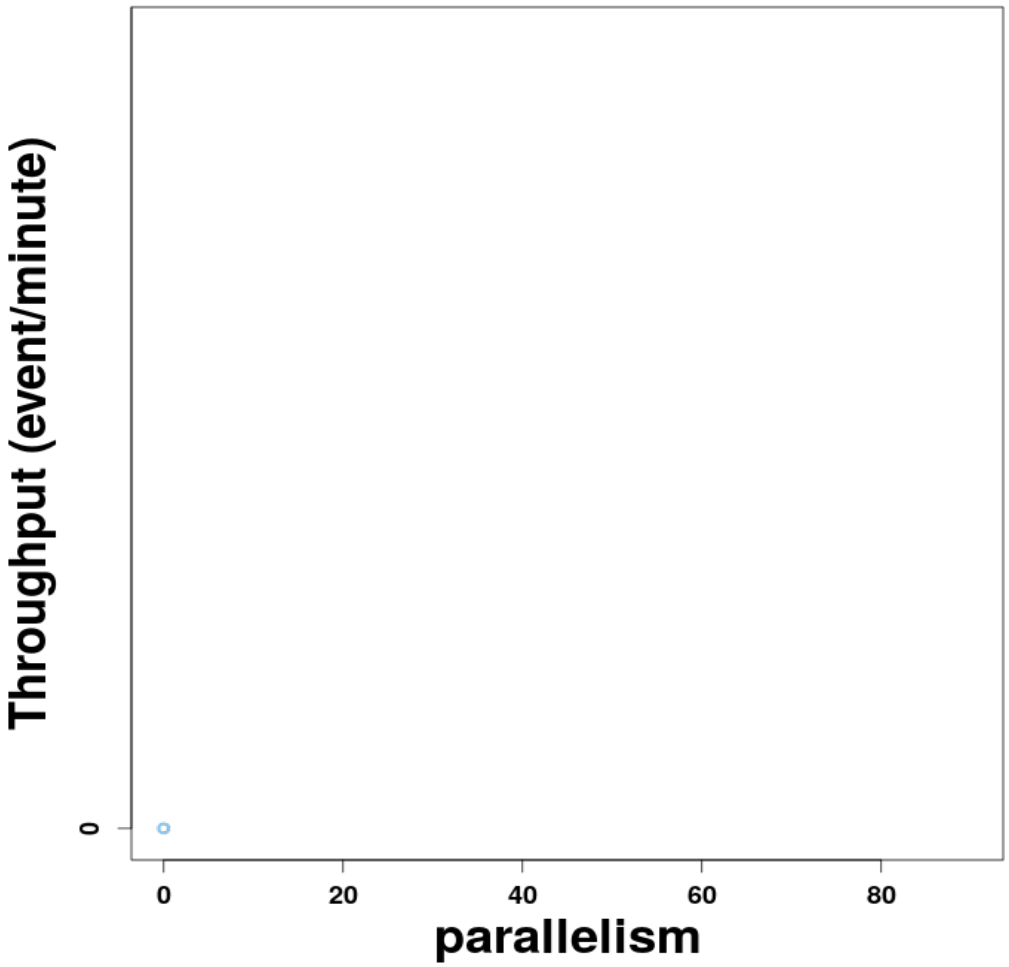

Figure 5.9: Throughput of the proposed system on YARN cluster.

6 Conclusion and Future Work

In this chapter, we discuss the results of our system, and some of the aspects of underlying method. Also we give some proposals for future work.

6.1 Conclusion

6.2 Future Work

Bibliography

[1] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 147–160. ACM, 2008.

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

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

[4] Elias Alevizos, Alexander Artikis, and George Paliouras. Event forecasting with pattern markov chains. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pages 146–157. ACM, 2017.

[5] Theodore W Anderson and Leo A Goodman. Statistical inference about markov chains. *The Annals of Mathematical Statistics*, pages 89–110, 1957.

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

[7] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16, 2002. ISSN 1581135076. doi: 10.1145/543614.543615.

[8] Arindam Banerjee and Joydeep Ghosh. Clickstream clustering using weighted longest common subsequences. In *Proceedings of the web mining workshop at the 1st SIAM conference on data mining*, volume 143, page 144, 2001.

[9] Dimitri P Bertsekas and John N Tsitsiklis. *Introduction to probability*, volume 1. Athena Scientific Belmont, MA, 2002.

[10] P Billingsley. Statistical Methods in Markov Chains. *The Annals of Mathematical Statistics*, 32(1):12–40, 1961. ISSN 00034851. doi: 10.1214/aoms/1177705148. URL `http://www.jstor.org/stable/2238700`.

[11] Luca Canzian, Yu Zhang, and Mihaela van der Schaar. Ensemble of distributed learners for online classification of dynamic data streams. *IEEE Transactions on Signal and Information Processing over Networks*, 1(3):180–194, 2015.

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

[13] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Nesime Tatbul, Stan Zdonik, and Michael Stonebraker. Monitoring streams—a new class of data management applications. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 215–226. Elsevier, 2002.

[14] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3): 15:1–15:62, June 2012. ISSN 0360-0300. doi: 10.1145/2187671.2187677. URL `http://doi.acm.org/10.1145/2187671.2187677`.

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

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

[17] William Feller. *An introduction to probability theory and its applications: volume I*, volume 3. John Wiley & Sons New York, 1968.

[18] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Michael Kamp, and Michael Mock. Issues in complex event processing: Status and prospects in the Big Data era. *Journal of Systems and Software*, 127:217–236, 2017. ISSN 01641212. doi: 10.1016/j.jss.2016.06.011.

[19] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Michael Kamp, and Michael Mock. Issues in complex event processing: Status and prospects in the big data era. *Journal of Systems and Software*, 127:217–236, 2017.

[20] The Apache Software Foundation. Apache Kafka. `https://kafka.apache.org/`, 2012.

[21] The Apache Software Foundation. Apache Spark Streaming. `http://spark.apache.org/streaming/`, 2013.

[22] The Apache Software Foundation. Apache Flink. `https://flink.apache.org/`, 2014.

[23] The Apache Software Foundation. Apache Storm. `http://storm.apache.org/`, 2014.

[24] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Automata theory, languages, and computation. *International Edition*, 24, 2006.

[25] Ronald A Howard. *Dynamic probabilistic systems: Markov models*, volume 1. Courier Corporation, 2012.

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

[27] Michael Kamp, Sebastian Bothe, Mario Boley, and Michael Mock. Communication-efficient distributed online learning with kernels. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 805–819. Springer, 2016.

[28] Rikard Laxhammar and Göran Falkman. Conformal prediction for distribution-independent anomaly detection in streaming vessel data. In *Proceedings of the First International Workshop on Novel Data Stream Pattern Mining Techniques*, pages 47–55. ACM, 2010.

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

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

[31] David Luckham. 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*, pages 3–3. Springer, 2008.

[32] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1 (3):259–289, 1997.

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

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

[35] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Duplicate detection in click streams. In *Proceedings of the 14th international conference on World Wide Web*, pages 12–21. ACM, 2005.

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

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

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

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

[40] International Maritime Organization. Automatic identification systems. `http://www.imo.org/OurWork/Safety/Navigation/Pages/AIS.aspx`, 2001.

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

[42] Kostas Patroumpas, Alexander Artikis, Nikos Katzouris, Marios Vodas, Yannis Theodoridis, and Nikos Pelekis. Event recognition for maritime surveillance. In *EDBT*, pages 629–640, 2015.

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

[44] Ehab Qadah, Michael Mock, Elias Alevizos, and Georg Fuchs. A Distributed Online Learning Approach for Pattern Prediction over Movement Event Streams with Apache Flink. In *EDBT/ICDT Workshops*, 2018.

[45] Timo Reuter and Philipp Cimiano. Event-based classification of social media streams. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, page 22. ACM, 2012.

[46] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter Pietzuch. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, page 4. ACM, 2009.

[47] Cem Tekin, Simpson Zhang, and Mihaela van der Schaar. Distributed online learning in social recommender systems. *IEEE Journal of Selected Topics in Signal Processing*, 8(4):638–652, 2014.

[48] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

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

[50] Gary M Weiss and Haym Hirsh. Learning to predict rare events in event sequences. In *KDD*, pages 359–363, 1998.

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

[52] Yu Zhang, Daby Sow, Deepak Turaga, and Mihaela van der Schaar. A fast online learning algorithm for distributed mining of bigdata. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):90–93, 2014.

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