Extended Kalman Filter for Large Scale Vessels Trajectory Tracking in Distributed Stream Processing Systems

Katarzyna Juraszek[1], Nidhi Saini[2], Marcela Charfuelan[2], Holmer Hemsen[2], and Volker Markl[1,2]

[1] Technische Universität Berlin, Straße des 17. Juni 135, 1062, Berlin, Germany
https://www.tu-berlin.de
[2] DFKI GmbH, Alt-Moabit 91c, 10559, Berlin, Germany
https://www.dfki.de

Abstract. The growing number of vehicle data being constantly reported by a variety of remote sensors, such as Automatic Identification Systems (AIS), requires new data analytics methods that can operate at high data rates and are highly scalable. Based on a real-life data set from maritime transport, we propose a large scale vessels trajectory tracking application implemented in the distributed stream processing system Apache Flink. By implementing a state-space model (SSM) - the Extended Kalman Filter (EKF) - we firstly demonstrate that an implementation of SSMs is feasible in modern distributed data flow systems and secondly we show that we can reach a high performance by leveraging the inherent parallelization of the distributed system. In our experiments we show that the distributed tracking system is able to handle a throughput of several hundred vessels per ms. Moreover, we show that the latency to predict the position of a vessel is well below 500 ms on average, allowing for real-time applications.

Keywords: time-series · state-space models · Extended Kalman Filter · stream processing · spatio-temporal data · remote sensing systems.

1 Introduction

Analysing and understanding of maritime traffic is a topic of increasing interest, due to its direct implications on security and safety, as well as on environmental and socio-economic factors. Nowadays, there is a growing number of ship reporting technologies and remote sensing systems such as the Automatic Identification System (AIS), Long Range Identification and Tracking (LRIT), radar tracking or Earth Observation. Each of these technologies provides spatio-temporal vessel positioning data that contributes to better monitoring of maritime transport. The AIS technology has become a standard in the industry, being mandatory for ships in international voyages, such as cargo vessels, fishing vessels exceeding certain size as well as all passenger vessels, regardless of their size. The AIS information provided by vessels, as a stream of tuples, includes kinematic information such as latitude, longitude, speed and course, voyage information including destination port and estimated time of arrival, as well as static data such as size and type of a ship. The AIS technology, which was originally introduced for collision avoidance, is currently also used for vessel tracking, vessel behaviour identification and anomaly detection [3].

State-space models (SSMs) are a popular methodology to model how different phenomena change over time [16]. The term *state-space* was originally coined by Kalman (1960), and applied to the field of control engineering. An SSM is a representation of some physical system, where input, output and state variables are related by first-order differential equations. The state variables depend on input variables, while the output variables depend on the values of the state variables. The SSM methodology has been successfully applied in various fields, including engineering, statistics, computer science and economics. The Kalman Filter is well-known for being one of the most powerful techniques for state estimation. The purpose of the algorithm is to provide an estimation with minimum error variance. The nonlinear version of the algorithm, the so called Extended Kalman Filter (EKF) is widely used to estimate position in GPS receivers [13] or for robot tracking [17]. The academic community, using EKF in practice, usually focuses on motion of robots in a constrained space. In contrast, applying state space models to vessels' tracking data imposes further challenges such as infrequent or discontinued observations, arbitrary or noisy trajectories and erratic movements.

Distributed computing helps in processing large amounts of raw data in real-time and in a timely manner by parallelising the computation, distributing the data and handling failures [11]. In addition, distributed stream processing solutions are helping to overcome the main obstacles of real-time processing, which are achieving the consistency of states across the system as well as fault recovery, requiring long recovery times. Thanks to these features, new distributed processing engines, which provide users with the scalable execution of data analysis tasks are arising [12]. The main goal of current and popular engines such as Hadoop [2], Apache Spark [7] or Apache Flink [6] is to enable developers to write distributed data analysis applications in an easy and efficient manner.

In this paper we propose an implementation of the EKF in a distributed stream processing system for real-time trajectory tracking of many vessels in parallel. In our experiments each vessel provides a stream of remote sensing AIS data, which is used to create and continuously update its SSM. This way we are able to estimate in real-time a new position of each vessel. We show that in our use case the distributed tracking system is able to handle a throughput of 200 vessels/ms and requires a latency below 500 ms on average to predict the next position of a vessel.

The paper is organised as follows. First we summarize the selection of related work on processing streams, real-time data tracking and EKF. Section 3 includes the theory behind the EKF as well as the practical implementation of the algorithm in Apache Flink. Section 4 describes the technical setup behind the EKF in the distributed environment. In the subsequent part, Section 5, the data set characteristics, accuracy results of the experiments as well as performance evaluation is given, followed by the conclusion in the last section.

2 Related Work

As reported in the survey paper [24], AIS data is used nowadays in several works for mining relevant aspects of navigation such as safety of seafaring, namely traffic anomaly detection, route estimation, collision prediction, and path planning. In this survey several techniques are reported, including EKF as a learning-model-based method for route estimation.

EKF has been used for tracking vessels in many works. For example in Perera and Soares [21] an EKF algorithm is proposed as a vessel state estimator due to its capabilities of fusing nonlinear system kinematics with a given set of noisy measurements. They use the EKF not only for state estimation (i.e. position, velocity and acceleration) but also for trajectory prediction. Their experiments are performed in Matlab and only with simulated data.

SSMs and Kalman Filtering have been studied and implemented in various tools for a long time but mainly applied to single machines and relatively small sets of batch data. Several traditional tools get short on processing the big amounts of data that can be generated nowadays or simply they are not capable of processing stream data. Therefore some researchers have started to consider the possibility to use and implement these techniques in large scale distributed data flow systems. For example Sheng et al. [23] implemented an extended Kalman filter (a recursive filter) using the MapReduce framework, in order to perform prediction in an industrial setting. Moussa [19] used Apache Spark Streaming to implement a scalable application for real-time prediction of vessels' future locations. The method used in this work for estimating a new position is based on a scalable computation of trip patterns, which are efficiently queried using a geo-hashing index. This work also uses the DEBS Challenge 2018 data, but unfortunately it does not report thorough experiments on throughput or latency.

Another interesting work that already addresses the problem of processing streams of AIS data in real-time is reported by Brandt and Grawunder [9], where the whole trajectories of a vessel and its current neighbors are predicted in order to avoid critical situations, such as two vessels being too close to each other. In the setup of this work, the real-time arrival and processing of the data points is simulated by sampling the data and then estimating the trajectory of a vessel ten minutes into the future. In contrast to the approach presented in our paper, the authors calculate ten locations per predicted trajectory rather than the next location of the vessel. The authors admit that the simple linear extrapolation used in their work to predict the future trajectories of the moving objects leads to non-optimal predictions, especially when a vessel is turning. In addition, the authors perform the computation on a single machine and therefore the computationally demanding queries are far from delivering the results in near real-time. Dalsnes et al. [10] present a similar data-driven approach using cubic spline interpolation of trajectories sampled from an AIS historical database. They predict vessel's position 5 to 15 minutes into the future also using the recent past trajectory of the vessel. This work relies heavily on the availability of historical data and there is no information regarding its use in a streaming fashion or real-time.

3 The Extended Kalman Filter

We apply distributed data flow processing to analyse kinematic information, such as latitude, longitude, speed and course from multiple vessels in parallel. While processing the data, we apply the Extended Kalman Filter to real-time stream data in order to estimate the next position of a vessel. To the best of our knowledge, we present the first implementation of an Extended Kalman Filter on the distributed data flow system Apache Flink.

3.1 EKF in Theory

As presented in Figure 1, the EKF consists of two steps, first, the prediction step, and second, the correction step. The prediction step starts with the approximation of the state ahead, the so-called a priori state $\hat{x}_t^-$. The $f(\hat{x}_{t-1}^+)$ function is a non-linear function relating the a posteriori state at the previous time $t-1$ to the state at the current time t. The added w_{x_t} is a white Gaussian process noise with 0 mean and covariance Q_t. The second equation in the prediction step projects the error covariance P_t^-, called the a priori error covariance, where $F(\hat{x}_{t-1}^+)$ is the Jacobian matrix of partial derivatives of f with respect to x. The correction part of the algorithm starts now and the measurement equation z_t is introduced at this point.

$$z_t = h(x_t^-)) + w_{z_t}, \tag{1}$$

where w_{z_t} is white Gaussian measurement noise with zero mean and R_t covariance. The $h(x_t^-))$ function in the measurement equation z_t is a non-linear function relating the a priori state x_t^- to the actual measurement at time t.

The correction part takes the a priori state and the a priori error covariance to compute three values. The first value calculated in the correction step is the *Kalman gain* K_t. It represents how a new measurement improves the predicted state vector, where $H(\hat{x}_t^-)$ is the Jacobian matrix of partial derivatives of h with respect to x. Next, the a posteriori state $\hat{x}_t^+$ is calculated by updating the a priori state with the actual measurement. In the end the a posteriori error covariance P_t^+ is computed using identity matrix, Kalman gain K_t, Jacobian matrix $H(\hat{x}_t^-)$ and a priori error covariance P_t^-. Now the algorithm loops by again starting the prediction part, using the calculated posteriori values as input in the prediction equations. The interested reader is referred to [20] and [25] for a detailed and practical description of SSM and EKF.

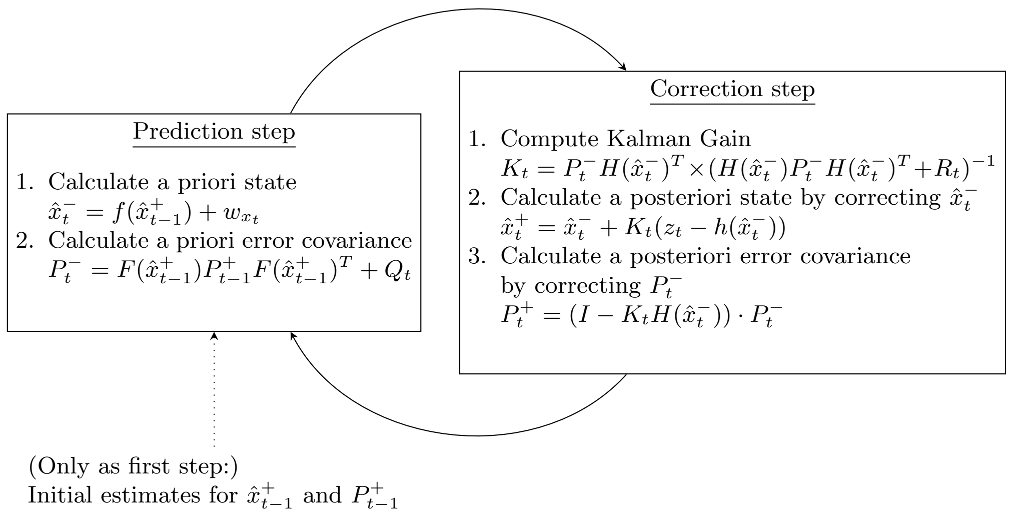
Prediction step
1. Calculate a priori state
$\hat{x}_t^- = f(\hat{x}_{t-1}^+) + w_{x_t}$
2. Calculate a priori error covariance
$P_t^- = F(\hat{x}_{t-1}^+)P_{t-1}^+F(\hat{x}_{t-1}^+)^T + Q_t$
Correction step
1. Compute Kalman Gain
$K_t = P_t^- H(\hat{x}_t^-)^T \times (H(\hat{x}_t^-)P_t^- H(\hat{x}_t^-)^T + R_t)^{-1}$
2. Calculate a posteriori state by correcting $\hat{x}_t^-$
$\hat{x}_t^+ = \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-))$
3. Calculate a posteriori error covariance by correcting P_t^-
$P_t^+ = (I - K_t H(\hat{x}_t^-)) \cdot P_t^-$
(Only as first step:)
Initial estimates for $\hat{x}_{t-1}^+$ and P_{t-1}^+

Fig. 1: Graphical representation of the Extended Kalman Filter operations (adapted from [25]).

3.2 EKF in Practice

One of the prerequisites for implementing EKF in practice is the a priori knowledge of the type of movement of an object. In case of tracking, such as vessels on waters, no a priori knowledge of the directions of the target is generally available, therefore in our case the behaviour of vessels is approximated by a constant velocity model. Since ocean vessels tend to follow a slow parabolic-type movement, where fast changing manoeuvres are not present, this assumption goes in line with other scholars' findings [22]. In order to use the coordinate data, the geodetic coordinates (WGS 84), which are not suitable for data processing are converted so that the next location of a vessel is not predicted with respect to longitude and latitude values, but rather as a latitude and longitude distance in meters from the point where last position of a ship was reported.

EKF Parameters Initialization In order to start the EKF for the first time, two parameters need to be initialized, a posteriori state and a posteriori error covariance. Since the starting point of the route is known, the a posteriori error covariance is set to a small value (0.01) on the main diagonal of the a posteriori error covariance matrix. The initial state estimate is set to zero, as the values will be replaced with the next run of the EKF. Two other parameters, being reused by the EKF on each run, are Q and R, which are the process noise covariance matrix and measurement noise covariance matrix. When using the EKF algorithm for tracking of moving objects, Q represents possible accelerations that allow the tracked object to deviate from constant velocity. Following the assumptions of the acceleration process noise, which can be assumed to be 8.8 m/s^2 and assuming 2 rad/s as maximum turn rate for the vehicle, the following values are set on the main diagonal of the process noise covariance matrix: $[(0.5 \cdot 8.8 \cdot \Delta t^2)^2, (0.5 \cdot 8.8 \cdot \Delta t^2)^2, (2 \cdot \Delta t)^2), (8.8 \cdot \Delta t)^2]$, where Δt is the time difference in seconds between the current and the previous measurement [8]. The last parameter to be initialised is the measurement noise covariance matrix R. The measurement noise covariance R can be defined using the standard deviation of a GPS measurement, which is assumed to be 6.0. The bigger the value, the less "trust" is given to the sensor readings [15].

EKF Implementation In the EKF algorithm implemented for the purpose of finding the next position of a vessel, the belief state to be estimated has four variables: cumulative longitude distance x, cumulative latitude distance y (both calculated from the departure point), heading, and velocity of a vessel at a given time t. The algorithm starts with calculating the a priori state. To do so, the a posteriori state from previous measurements is used with the constant velocity model to predict the new a priori state. The a priori state has the following form:

$$\hat{x}_t^- = \begin{bmatrix} x_t^- \\ y_t^- \\ \psi_t^- \\ \upsilon_t^- \end{bmatrix} = \begin{bmatrix} x_{t-1}^+ + \Delta t \cdot \upsilon_{t-1}^+ \cdot \cos(\psi_{t-1}^+) \\ y_{t-1}^+ + \Delta t \cdot \upsilon_{t-1}^+ \cdot \sin(\psi_{t-1}^+) \\ (\psi_{t-1}^+) \bmod (2 \cdot \pi) - \pi \\ \upsilon_{t-1}^+ \end{bmatrix} \tag{2}$$

where $\hat{x}_t^-$ is the predicted a priori state, x_t^- and y_t^- are respectively the cumulative longitude and latitude distance in meters from the departure port, Δt is the time difference in seconds between the current and the previous measurement, ψ_{t-1}^+ is the heading of a vessel and υ_{t-1}^+ is the velocity of a vessel in meters per second.
To calculate the a priori error covariance, the $F(\hat{x}_{t-1}^+)$, which is the Jacobian matrix of partial derivatives of $f(\hat{x}_{t-1}^+)$ with respect to x, needs to be calculated first.

$$F(\hat{x}_{t-1}^+) = \begin{bmatrix} 1 & 0 & -\Delta t \cdot \upsilon_{t-1}^+ \cdot \sin(\psi_{t-1}^+) & \Delta t \cdot \cos(\psi_{t-1}^+) \\ 0 & 1 & \Delta t \cdot \upsilon_{t-1}^+ \cdot \cos(\psi_{t-1}^+) & \Delta t \cdot \sin(\psi_{t-1}^+) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The a priori error covariance is then predicted following the formula given for P_t^-. The input in the calculation of the a posteriori state is the actual measurement data z_t and the a priori state $\hat{x}_t^-$. In our EKF implementation, the actual measurement data is the actual longitude and latitude distance from the departure point, calculated as the cumulative sum of all the distances between the measurements until this point in time.

$$z_t = \begin{bmatrix} \text{measured cumulative longitudinal distance} \\ \text{measured cumulative latitude distance} \end{bmatrix} \tag{4}$$

The remaining parts of the algorithm are calculated following the equations from Figure 1.

4 Distributed Pipeline

4.1 Technical setup

The technical setup of the processing pipeline for this work is presented in Figure 2. The real-time arrival of the time-series data is simulated using Apache Kafka and the distributed computing of the next location prediction given by the EKF is leveraged with the use of Apache Flink.

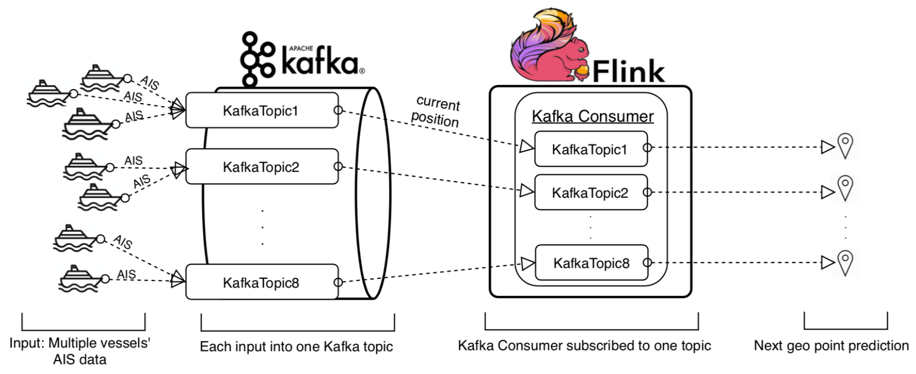
Apache kafka
Flink
AIS
KafkaTopic1
KafkaTopic2
KafkaTopic8
current position
Kafka Consumer
KafkaTopic1
KafkaTopic2
KafkaTopic8
Input: Multiple vessels' AIS data
Each input into one Kafka topic
Kafka Consumer subscribed to one topic
Next geo point prediction

Fig. 2: Detailed Kafka Flink pipeline.

4.2 EKF in the distributed environment

Since every non-trivial streaming application is stateful, applying the EKF in a distributed environment using Flink requires working with the state abstraction. States are an important feature but also have a serious performance impact on the processing in distributed data flow systems, as they require synchronization across machines and need to be managed in a fault-tolerant way in case of machine failures. A stateful application remembers certain events or intermediate results, which can be accessed later, for instance when a new event is arriving [6]. Given the recursive nature of the EKF algorithm, where the a posteriori values calculated in the correction step are further used in the prediction step when a new event arrives, the use of *Keyed State* operators [4] is crucial for implementing this algorithm in a distributed system. We use four different states in our work:

- The first state `prevKalmanParams` is used to store a tuple consisting of a posteriori state and a posteriori error covariance calculated in the Correction step.
- The second state `prevTimestamp` stores the timestamp of the last arriving event so that the time difference (Δt) needed for the prediction of the a priori state can be calculated upon arrival of the next event.

- The third state `prevGeoPoints` remembers the last position reported in the last event in order to calculated the distance travelled.
- The fourth state `prevCumSum` stores the cumulative sum of distance travelled from the reported departure port.

Each of these states is updated on every input tuple whenever an event arrives. The state values from the previous run are fed to the current calculation. In the implementation of the EKF algorithm the *Managed Key State ValueState<T>* was used, which is a state scoped to the key of the current input element. It means that every keyed stream, belonging to one trajectory, will have a corresponding state. This type of state can keep the value, which can be then retrieved and updated per key. In our case one key corresponds to one vessel.

5 Data and Experiments

The data set used in the vessels trajectory tracking use case, was provided by MarineTraffic during the 2018 DEBS Grand Challenge [1] and includes the geo-location data, in terms of latitude and longitude, of vessels departing from 25 ports in the Mediterranean Sea. The data is provided as a continuous stream of tuples. A ship sends a tuple according to its behaviour based on the AIS specifications. Each these tuples includes also the name of the port of origin, unique ID of the vessel, time stamp, vessel's course, heading and draught. The data include several types of vessels, corresponding to 503 trajectories obtained during a period of approximately three months in 2015 (10-03-15 13:13 to 19-05-15 7:32). Many of the vessels report their position every two minutes, but some have very irregular periods, including long periods of time (several hours) with no report.

The experiments are conducted on a single server machine with 48 CPUs, 2.0 Ghz, 126 GB of RAM, running Ubuntu 16.04, Apache Kafka (v. 2.11) and Apache Flink (v. 1.8). Following the recommendations in the Flink documentation, we fixed the number of Flink task slots to 48. Thus in our experiments the level of parallelism is given by the number of slots or CPUs used [5].

In the following we evaluate our system according to accuracy and large scale performance. In the first experiment, Section 5.1, we calculate the next position prediction error for every point of the 503 trajectories in the data. In the second experiment, Section 5.2, we evaluate the performance of the system in terms of event and processing time latency as well as ingestion rate.

5.1 Accuracy Evaluation

The result of the point prediction using the Extended Kalman Filter is a longitude and latitude pair of the next vessel's position. For evaluating accuracy we use the RMSE, which is also used for example in [10] to analyse the proximity of the mean of predicted values to the true value. We apply RMSE to calculate the prediction error for latitude and longitude values but also for distance, that is, the distance between the predicted position point and the actual point. As it is done in [10, 18], we define the distance RMSE for a vessel's trajectory, of L total number of reported AIS tuples, as:

$$RMSE_{dist} = \sqrt{\frac{\sum_{l=1}^{L} \Delta d(l)^2}{L}} \tag{5}$$

where $\Delta d(l)$, for a particular position l, is the actual distance between the true position and the one predicted with EKF, both input as pair of longitude and latitude coordinates. The lower the RMSE, the better the prediction. In our case $\Delta d(l)$ is calculated using the *Haversine*[3] distance to calculate the great-circle distance between two points. The RMSE for longitude and latitude is calculated similarly, subtracting the true longitude (or latitude) from the predicted one.

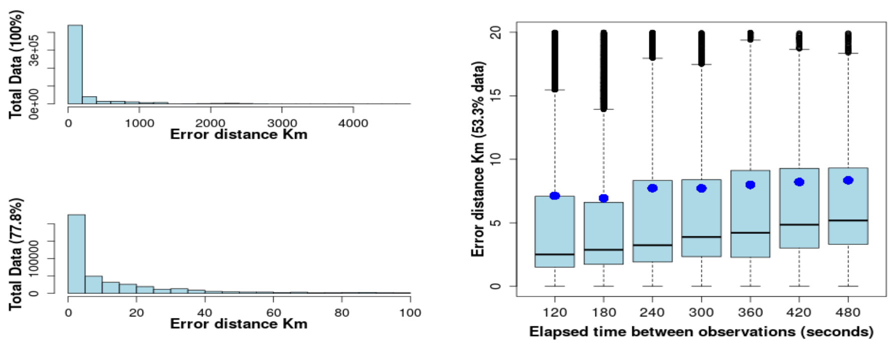

Fig. 3: Accuracy Evaluation: (left) Error distance distribution and (right) Impact of Elapsed Time (ET) between observations, blue points corresponds to $RMSE_{dist}$.

After applying EKF to the 503 trajectories in the data, equivalent to 542,153 position points (AIS tuples), we analyse the overall distribution of distance errors. In Figure 3 (left) we can observe in the histograms that most of the distance errors are below 20 km in around 78% of all the data and will therefore focus now on the data with distance error below 20 km. In a preliminary study we observed that the number of observations per kilometer has an impact on the RMSE for latitude and longitude, i.e. the more reported positions per kilometer of trajectory the better the prediction. Thus we analyse the error distance with respect to the elapsed time between observations. We can observe in the boxplot of Figure 3 (right) that the more frequent the observations (120 seconds), the lower the distance error, in fact, for some trajectories the $RMSE_{dist}$ is well below 1 km (see below Figure 4 and Tables 1 and 2). These levels of $RMSE_{dist}$ (below 1km) are also obtained by Dalsnes et al. [10], in a batch setting where the approach is slightly different than ours and the results are given in *median* $RMSE_{dist}$ values calculated for partial trajectories.

[3] https://www.movable-type.co.uk/scripts/latlong.html

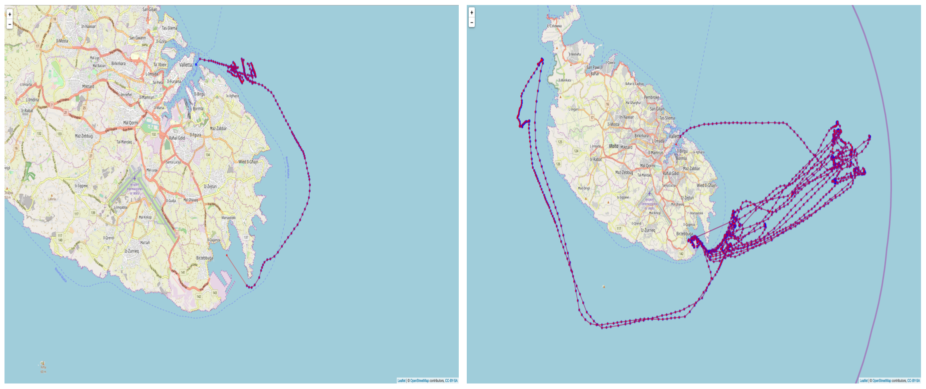
Fig. 4: Accuracy Evaluation: two trajectories corresponding to shipID-28 (left $RMSE_{dist}$=334.14m, mean ET=167.2 sec.) and shipID-57 (right $RMSE_{dist}$=664.23m, mean ET=667.9 sec.), blue points correspond to actual values and red points to the ones predicted with EKF.

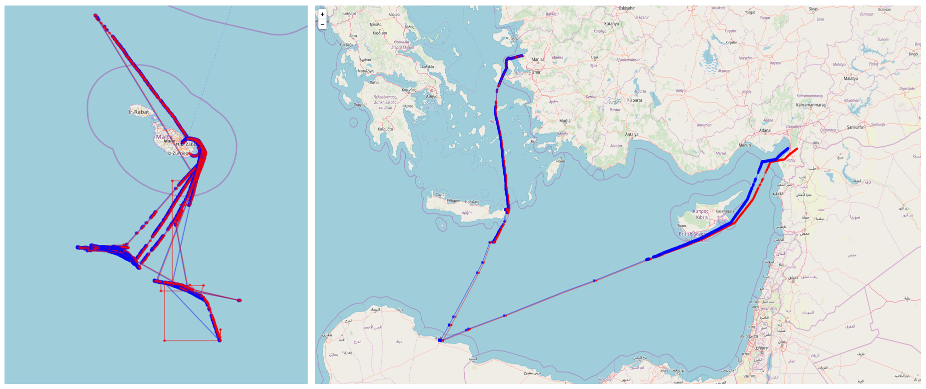
Fig. 5: Accuracy Evaluation: two trajectories corresponding to shipID-2 (left $RMSE_{dist}$=4755.24m, mean ET=1279.7 sec.) and shipID-95 (right $RMSE_{dist}$=20279.30m, mean ET=1644.7 sec.), blue points correspond to actual values and red points to the ones predicted with EKF.

Table 1: Accuracy evaluation: Distance error quantiles for selected ship trajectories.

Ship ID	Distance Error Quantiles					
	Min	1Q	Median	Mean	3Q	Max
28	40.0	133.4	233.5	271.5	361.4	1978.1
57	0.3	17.4	45.2	363.0	612.7	6442.5
2	1.0	331.1	580.1	1110.0	889.1	105040.3
95	601.9	2251.2	4844.1	9034.3	11213.3	477542.5

Table 2: Accuracy evaluation: Distance RMSE for selected ship trajectories.

Ship ID	RMSE			Total route length [km]
	Distance error [m]	Longitude degrees	Latitude degrees	
28	334.1	0.00169	0.00127	193
57	664.2	0.00337	0.00174	2735
2	4755.2	0.02438	0.02725	3881
95	20279.3	0.15373	0.03960	1884

As we will discuss later, there are various factors that impact the accuracy of the prediction, but first of all let us consider some examples of trajectories, in Figures 4 and 5 and corresponding statistics in Tables 1 and 2. In cases where the tracking data is frequently reported per kilometer of distance traveled, the EKF algorithm has the chance to go through more iterations and therefore improve its prediction with each run. As shown in Figure 4, the two vessels have a $RMSE_{dist}$ of 334 and 664 m respectively and in both cases the ET between observations is below 3 and 10 minutes in average. In the case shown in Figure 5, the infrequent observations lead to inaccurate route prediction. For shipID-2 (left) the ET is around 20 minutes on average and the route is pretty complex, with several turns, however its $RMSE_{dist}$ is still below 5 km. For shipID-95 (right) the ET is around 27 minutes in average, including long periods of no reporting at all, which makes the prediction deviate a lot.

Factors influencing accuracy of the EKF prediction. In the EKF algorithm, each new received event provides an adjustment to the model, improving the chances of correctly predicting the next state. The less events arrive, the smaller the chances that the estimated values will be accurate. In order to investigate if more data points indeed improve the prediction, the number of observation points per km of distance traveled was calculated for each trajectory. Figure 6 shows respectively longitude RMSE and latitude RMSE against the number of observations per km. The majority of analyzed vessels report less than two observations per km of distance traveled. In line with the expectations coming from the nature of the EKF algorithm, both diagrams in Figure 6 confirm that the higher number of received observations per km results in more accurate predictions (smaller longitude and latitude RMSE). Another factor, which causes the results to be imprecise is the reported departure port. As the EKF algorithm requires initialization values to be input in the first step of the algorithm, the longitude and latitude of the departure port are used as the initial location coordinates. In some cases, the route was reported to start in the wrong position, which results in a significantly longer convergence time.

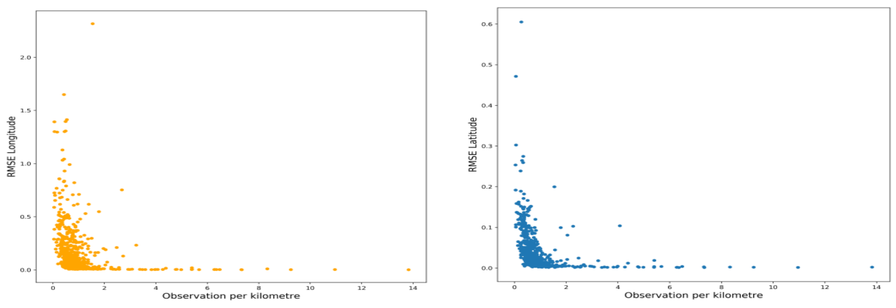

Fig. 6: Accuracy Evaluation: RMSE for Longitude and Latitude.

5.2 Performance Evaluation

In our large scale performance experiment we start simulating the streaming process by injecting the data into Kafka using several topics. Each Kafka topic is then read as a stream data source by Apache Flink. Each tuple in the stream source is processed by Flink using its event time, i.e. the timestamp when the position is reported. In the data, on average 10 vessels report their position at the same time with peaks of up to 50 different vessels reporting their position simultaneously. This means that our processing system must be able to track in real time many vessels at the same time. In order to cover this situation and stress the system even more, we replicated four times the input data assigning different *ships_id*. In this way we simulate the processing of more than 2000 trajectories with peaks of maximum 200 vessels simultaneously reporting their position.

According to the benchmarking study of Karimov et al. [14], in modern distributed stream processing systems two notions of time are distinguished: event- and processing-time latencies. From this study we use the following metrics:

– *event-time latency*, which measures the time that a given event has spent in a queue waiting to be processed. In our case it is the time a tuple spent in the Kafka queue until the EKF operator is able to produce a new position prediction for this tuple.
– *processing-time latency*, which measures the time it took for the event to be processed by the streaming system, what in our case means the time it takes for the EKF operator to produce an output.
– *ingestion rate*, which is the throughput of a streaming system and in our case is measured as the number of tuples per millisecond processed by the EKF operator in Flink.

As pointed out by Karimov et al. [14], in practical scenarios, *event-time latency* is very important as it defines the time in which the user interacts with a given system and should be minimized. It should be noted that processing-time latency makes part of the event-time latency. Thus our objective is to find the configuration that minimizes the processing-time in our system.

As shown in our Kafka-Flink pipeline (see Figure 2) we use 8 Kafka topics and 8 corresponding Flink sources. The input source in each Kafka topic is the data replicated four times, which contains trajectories of more than 2000 vessels. Overall the system receives and processes 17.4 million AIS events.

We use the default configuration settings for Kafka, so the number of partitions per topic is 1. We change the level of parallelism from 1 to 48 and repeat each experiment five times, averaging the results afterwards. To be precise, parallelism 1 means using only one slot or CPU, which is the equivalent of executing the experiment on a single machine. The results are presented in Figure 7. We use logarithmic scale on the y-axis to facilitate comparison.

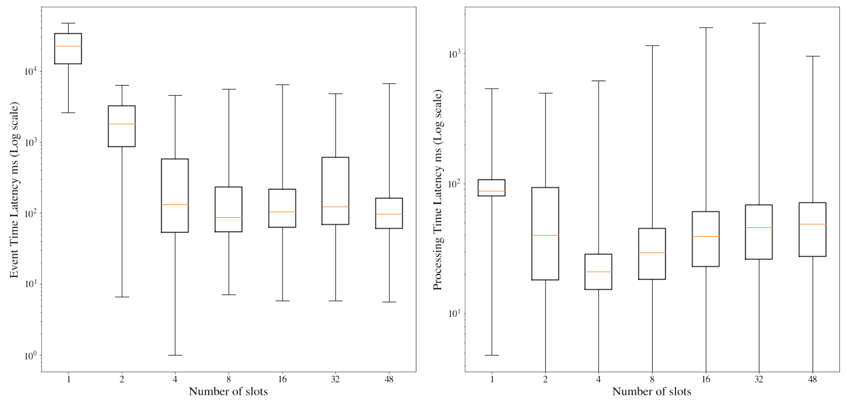

Fig. 7: Performance Evaluation: Boxplot of *event-time latency* and *processing-time latency* in server machine using 8 Kafka topics and 8 Flink sources.

For a single machine (in Figure 7 number of slots equal to 1), we obtain on average the highest *processing-time latency* and *event-time latency*. For parallelism 2 we can observe that the *processing-time latency* decreases but the *event-time latency* is also in the order of seconds, which is still too high for real time processing. The *processing-time latency* decreases significantly when increasing the level of parallelism, with optimal latency values for this setting, between parallelism 4 (mean 27.7 ms) and 8 (mean 37.1 ms). Using higher parallelism (parallelisation 8) also helps us to reduce the *event-time latency* to a mean minimum of 574.7 ms. We can observe that for parallelism above 2 the *event-time latency* is below a second, which can be explained by an increase of the *ingestion rate*. Therefore in the following, we further investigate the *ingestion rate* in our system.

As a comparison in Figure 8 we show a boxplot of the *ingestion rate* in Flink in terms of tuples per millisecond. We can observe that without parallelism the ingestion rate on average is minimum with approx. 170 tuples/ms, when increasing the parallelism the ingestion rate is on average stable in approx. 200 tuples/ms, reaching maximum rates of approx. 500 tuples/ms.

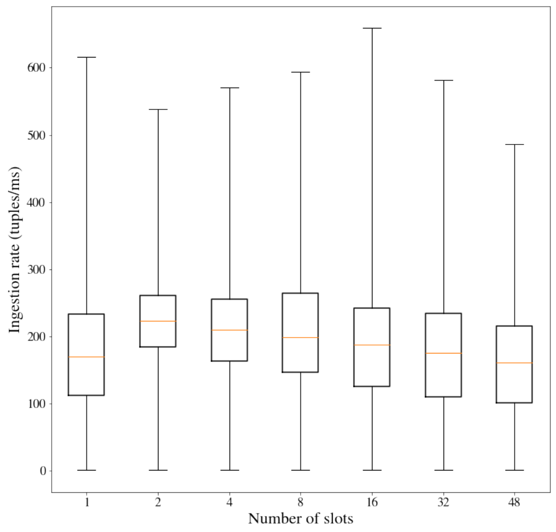

Fig. 8: Performance evaluation: Boxplot of the average Flink ingestion rate in tuples per millisecond.

There are several aspects that are interrelated and contribute to the overall performance of the system. For example, for parallelism 2 the average ingestion rate is higher but the processing latency with only two processors is still high. The optimal configuration in our system is obtained with 8 slots. Although the ingestion rate is stable at approximate 200 tuples/ms, by adding more than 8 slots we do not gain any performance. Such behaviour could be explain by the fact that the overall input data (17.4 million AIS events) is not large enough, so we do not benefit from increasing parallelism, but instead we introduce distribution overhead.

6 Conclusion

In line with the expectations coming from recursive nature of the EKF, where predictions are corrected upon the arrival of a next data point, the frequency of events reception turned out to be an important factor influencing the accuracy of prediction produced. The results show that the complexity or stability of the routes are not the most important factor contributing to the accurate prediction of the vessels' routes. Irrespective of the trajectory complexity, the high frequency of the incoming sensor measurements as well as correct initialisation of the parameters can provide a precise estimation of even more complex routes.

Regarding large scale performance, we showed that using a distributed stream processing system we can process on average 200 different vessels' positions per ms (200 tuples/ms), and our system, under this rate, requires below 500 ms to predict the next position of a vessel. In our setting, this optimal performance was obtained when using 8 Kafka topics and the corresponding 8 Flink sources.

Beyond this optimal value, when we increase the number of Kafka topics and Flink sources, the system introduces some overhead that is reflected on the latencies.

As future work we will consider a more realistic scenario, where massive real data is used and a setting in a cluster of computers. In a cluster setting we should take into account the additional overhead due to the communication between nodes, thus we will study the optimal combination of parallelism, Kafka topics and ingestion rate, in particular when actual big sets in the order of GBs are used. Furthermore, we will address the issue of visualization in real time including a dashboard for indicating various conditions of the vessels, such as elapsed time since last report, distance traveled or big error predictions which may correspond to possible anomalies.

Acknowledgements

This work was partly supported by the German Ministry for Education and Research (BMBF) as Berlin Big Data Center (BBDC2) (grant no. 01IS18025A), the German Federal Ministry of Transport and Digital Infrastructure (BMVI) through the Daystream project (grant no. 19F2031A).

References

1. DEBS 2018 Grand Challenge. http://www.cs.otago.ac.nz/debs2018/calls/gc.html, accessed: 2018-11-27
2. Hadoop. http://hadoop.apache.org/, accessed: 2018-11-27
3. Alessandrini, A., Alvarez, M., Greidanus, H., Gammieri, V., Arguedas, V.F., Mazzarella, F., Santamaria, C., Stasolla, M., Tarchi, D., Vespe, M.: Mining vessel tracking data for maritime domain applications. In: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW). pp. 361–367. IEEE (2016)
4. Apache Flink: Application Development, Working with State: https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/state/state.html, accessed: 2019-06-10
5. Apache Flink: Configuration: https://ci.apache.org/projects/flink/flink-docs-release-1.8/ops/config.html#configuring-taskmanager-processing-slots, accessed: 2019-06-10
6. Apache Flink: Fast and reliable large-scale data processing engine: http://flink.apache.org, accessed: 2018-11-27
7. Apache Spark: https://spark.apache.org/, accessed: 2018-11-27
8. Balzer, P.: Multidimensional Kalman-Filter. https://github.com/balzer82/Kalman/ (2017), accessed: 2018-11-27
9. Brandt, T., Grawunder, M.: Moving object stream processing with short-time prediction. In: Proceedings of the 8th ACM SIGSPATIAL Workshop on GeoStreaming. pp. 49–56. ACM (2017)
10. Dalsnes, B.R., Hexeberg, S., Flåten, A.L., Eriksen, B.H., Brekke, E.F.: The Neighbor Course Distribution Method with Gaussian Mixture Models for AIS-Based Vessel Trajectory Prediction. In: 2018 21st International Conference on Information Fusion (FUSION). pp. 580–587 (Jul 2018). https://doi.org/10.23919/ICIF.2018.8455607

11. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (2008)
12. He, B., Yang, M., Guo, Z., Chen, R., Su, B., Lin, W., Zhou, L.: Comet: batched stream processing for data intensive distributed computing. In: Proceedings of the 1st ACM symposium on Cloud computing. pp. 63–74. ACM (2010)
13. Jwo, D.J., Wang, S.H.: Adaptive fuzzy strong tracking extended kalman filtering for gps navigation. IEEE Sensors Journal **7**(5), 778–789 (2007)
14. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V.: Benchmarking distributed stream data processing systems. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE). pp. 1507–1518. IEEE (2018)
15. Kelly, A.: A 3d state space formulation of a navigation kalman filter for autonomous vehicles. Tech. rep., Carnegie-Mellon University Pittsburgh PA Robotics Institute (1994)
16. Korn, U.: A simple method for modelling changes over time. Casualty Actuarial Society E-Forum (2018)
17. Lee, J.W., Kim, M.S., Kweon, I.S.: A kalman filter based visual tracking algorithm for an object moving in 3d. In: Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots. vol. 1, pp. 342–347. IEEE (1995)
18. Lipka, M., Sippel, E., Vossiek, M.: An Extended Kalman Filter for Direct, Real-Time, Phase-Based High Precision Indoor Localization. IEEE Access **7**, 25288–25297 (2019). https://doi.org/10.1109/ACCESS.2019.2900799
19. Moussa, R.: Scalable maritime traffic map inference and real-time prediction of vessels' future locations on apache spark. In: Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems. pp. 213–216. ACM (2018)
20. Murphy, K.P.: Machine Learning: A probabilistic perspective. The MIT Press (2012)
21. Perera, L.P., Oliveira, P., Soares, C.G.: Maritime traffic monitoring based on vessel detection, tracking, state estimation, and trajectory prediction. IEEE Transactions on Intelligent Transportation Systems **13**(3), 1188–1200 (2012)
22. Perera, S., Suhothayan, S.: Solution Patterns for Realtime Streaming Analytics. In: Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems. pp. 247–255. DEBS '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2675743.2774214, http://doi.acm.org/10.1145/2675743.2774214
23. Sheng, C., Zhao, J., Leung, H., Wang, W.: Extended kalman filter based echo state network for time series prediction using mapreduce framework. In: 2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks. pp. 175–180. IEEE (2013)
24. Tu, E., Zhang, G., Rachmawati, L., Rajabally, E., Huang, G.B.: Exploiting ais data for intelligent maritime navigation: a comprehensive survey from data to methodology. IEEE Transactions on Intelligent Transportation Systems **19**(5), 1559–1582 (2017)
25. Welch, P., Bishop, G.: An Introduction to the Kalman Filter (courseP-ack). http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf (2001), accessed: 2018-11-27