Integrating EvolvingClusters to Apache Kafka – A Technical Aspects Overview

Andreas Tritsarolis
Data Science Lab. (datastories.org)
andrewt@unipi.gr

Yannis Theodoridis
Data Science Lab. (datastories.org)
ytheod@unipi.gr

Compiled at: 2020/07/18

1 Introduction

Mobility Data Analytics [1, 2, 3] is a growing branch of the general spectrum of Data Science. GPS enabled mobile phones as well as specialized equipment on board of cars, aircrafts and vessels, are the most common data sources, broadcasting huge volumes of location information. Using them as-is (i.e., in their "raw" form) offers limited usefulness to data scientists and domain experts. Thus, proper processing (cleansing, transformation, enrichment, simplification, etc.) and analysis (pattern discovery, behavioural profiling, etc.) are vital tools that aid us in understanding and making effective use of the available data. Grouping objects together based on the mobility behaviour that can be inferred from the data-points generated by them, for example, can provide us with a useful background that can benefit other analytics techniques in multiple ways. Some relevant techniques include [4]:

- *Object Profiling*, i.e., creating models that are mapped to multiple different objects with similar behaviour;
- *Trajectory compression/simplification*, by using the aforementioned models as points of reference that are able to adequately describe multiple trajectories;
- *Classification*, by using a more diverse set of trajectories/objects that are sampled from multiple of the available models, therefore reducing training times as well as preventing overfitting caused by a disproportionate amount of samples from one or more classes.

This report focuses on the technical aspects of integrating the EvolvingClusters algorithm [4] to the i4Sea Pipeline (i4sea.eu) using Apache Kafka®. More specifically, given a datastream of AIS positions, we aim to discover evolving clusters in real time. The rest of the report is structured as follows: Section 2 provides a brief description of the (integrated) method. Section 3 presents the algorithms that are used for discovering evolving clusters in real-time. In Section 4, we demonstrate the results of the aforementioned method based on a real-life maritime dataset. Finally, Section 5 concludes the paper and summarizes the leasons learnt.

2 Evolving Clusters Online Discovery Method

Figure 1 illustrates the block diagram of the evolving clusters integration as part of a real-time datastream application. The platform that handles the streaming part is based upon the popular publish-subscribe messaging system Apache Kafka®. At first, each AIS message transmitted by the vessels' antenna is sent to a Kafka Topic. Then, a Kafka Consumer reads the data reserved at the Topic and for each record:

- Calculates the pending timestamp, by rounding to the nearest multiple of the given sampling rate ($rate$)

  A necessary clarification at this point is that rounding depends on the alignment mode. If the alignment mode is (delayed) linear interpolation, the pending timestamp is the floored multiple of $rate$, otherwise, if the alignment mode is linear extrapolation, the pending timestamp is the ceiling multiple of $rate$, according to the following equation:

$$t_{pending} = \begin{cases} rate \times \left\lfloor \dfrac{t_{current}}{rate} \right\rfloor & mode = \text{“}inter\text{”} \\[2ex] rate \times \left\lceil \dfrac{t_{current}}{rate} \right\rceil & mode = \text{“}extra\text{”} \end{cases}$$

where the two modes “inter” and “extra” will be explained later in the presentation of Algorithm 3

- Afterwards, the consumed record – if it not noise – is appended to the objects’ buffer ($ObjectPool$) and;
- If the time is right (i.e., $t_{current} > t_{pending}$) $ObjectPool$ is used to produce the aligned timeslice as well as the evolving clusters up to $t_{pending}$.

Finally, the aforementioned results are sent to their respective Kafka Topics, where they can be connected to another consumer in order to perform visualizations and/or calculate statistics (e.g., avg. duration, speed, etc.). Figure 1 summarizes the previous discussion by illustrating the block diagram of the evolving cluster discovery online methodology.

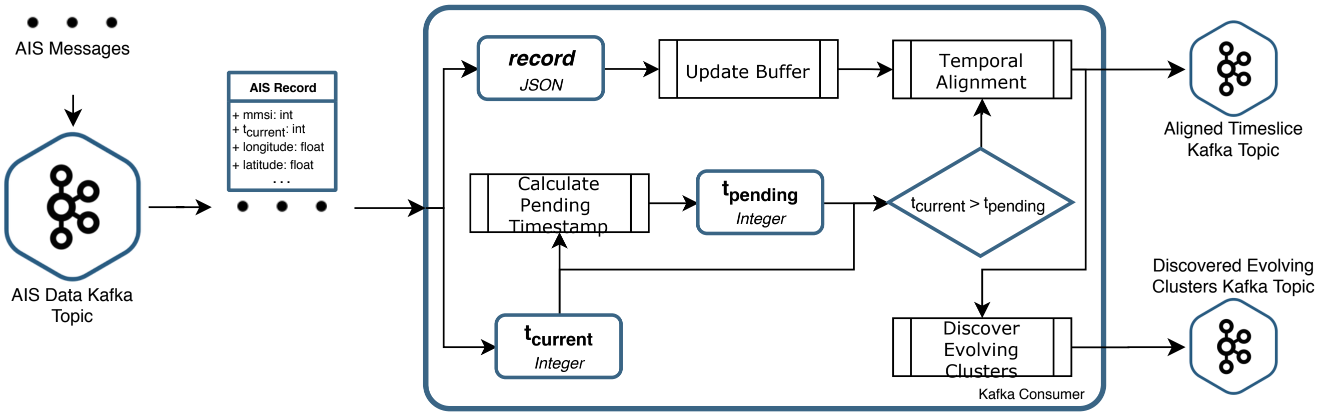

Figure 1: Evolving Cluster Discovery Online – Block Diagram.

3 Discovering Evolving Clusters Online using Apache Kafka

Algorithm 1: KConsumer. The core structure of the Kafka Consumer algorithm.

Input: Apache Kafka Topic $producer$, Alignment Rate $rate$, Alignment Mode $mode$

$t_{pending} \leftarrow None$
$timeslice \leftarrow \emptyset$
$ObjectPool \leftarrow \emptyset$
$patterns_{active} \leftarrow \emptyset$
$patterns_{closed} \leftarrow \emptyset$

foreach ***incoming message*** $msg \in producer$ **do**
    $record \leftarrow msg.value$
    $t_{current} \leftarrow msg.timestamp$
    $t_{round} \leftarrow RoundTimestamp(t_{current}, mode)$

    if $t_{pending} = None$ **then**
        $t_{pending} \leftarrow t_{round}$

    if $t_{pending} < t_{round}$ **then**
        $timeslice \leftarrow TemporalAlignment(ObjectPool, t_{pending}, mode)$
        $\{patterns_{active}, patterns_{closed}\} \leftarrow EvolvingClusters(timeslice, patterns_{active}, patterns_{closed})$
        output $patterns_{active}, patterns_{closed}$
        $\{ObjectPool, t_{pending}, timeslice\} \leftarrow AdjustBuffer(t_{round}, t_{pending}, ObjectPool)$

    $UpdateBuffer(ObjectPool, record, rate, mode)$
end

Algorithm 2: UPDATEBUFFER. Update the Objects' Buffer, given an incoming Data-Stream Record.

Input: Objects' Buffer $ObjectPool$, DataStream Record $record$, Alignment Rate $rate$, Alignment Mode $mode$
Output: (Updated) Objects' Buffer $ObjectPool$

1 $ObjectBuffer \leftarrow \{message \in ObjectPool : message.o_{id} = record.o_{id}\}$
2 **if** $ObjectBuffer = \emptyset$ **then**
3     $ObjectPool \leftarrow ObjectPool \cup \{record\}$
4 **else**
5     $LatestRecord \leftarrow ObjectBuffer_{|ObjectBuffer|}$
6     $dt \leftarrow record.t - LatestRecord.t$
7   **if** $dt > 2 \times rate$ **then**
8         $ObjectPool \leftarrow ObjectPool - ObjectBuffer$
9         $ObjectPool \leftarrow ObjectPool \cup \{record\}$
10   **else**
11         $distance \leftarrow HaversineDistance(record.location, LatestRecord.location)$
12         $speed \leftarrow \frac{distance}{dt}$
13   **if** $speed < SpeedThreshold$ **then**
14             $ObjectPool \leftarrow ObjectPool \cup \{record\}$
15 **return** $ObjectPool$

Algorithm 3: ADJUSTBUFFER. Adjust the Objects' Buffer w.r.t the Pending Timestamp

Input: Objects' Buffer $ObjectPool$, Rounded Timestamp t_{round}, Pending Timestamp $t_{pending}$, Alignment Mode $mode$, Alignment Rate $rate$
Output: (Updated) Objects' Buffer $ObjectPool$, Pending Timestamp $t_{pending}$, Aligned Timeslice $timeslice$

$ObjectPool \leftarrow \{message \in ObjectPool : t_{pending} - rate \leq message.t \leq t_{pending} + rate\}$
$t_{pending} \leftarrow t_{round}$

$timeslice \leftarrow \emptyset$
return $ObjectPool, t_{pending}, timeslice$

Algorithm 1, which is in charge of reading and managing the incoming transmissions of an AIS DataStream, uses the results of Algorithm 2 (i.e. the vessels' AIS messages buffer) and creates the aligned timeslice at the pending timestamp $t_{pending}$, which is used by EvolvingClusters in order to discover the evolving clusters at that timestamp. More specifically, for each incoming AIS transmission:

- Its mobility information is decoded (line 8);
- The pending timestamp is calculated (lines 9–13); and
- It is added to the objects' buffer ($ObjectPool$) – if it satisfies the conditions set at Algorithm 2 (line 21)

When the time is right (i.e., when the – rounded – timestamp of the AIS transmission t_{round} is greater than the pending timestamp $t_{pending}$):

- $ObjectPool$ is used to temporally align the objects at $t_{pending}$ (line 16);
- The aligned timeslice $timeslice$ is used as input to the EvolvingClusters algorithm, along with the respective methods of the latter (line 17); and
- Both active and closed patterns ($patterns_{active}$ and $patterns_{closed}$, respectively) are output to another Kafka Topic, called "ecdresults" (line 18).

Algorithm 2, which is in charge of creating and maintaining the objects' buffer, works as follows:

- Every message related to the object of interest $ObjectBuffer$ (i.e., the one that the transmitted message belongs to), is fetched (line 1);
- If $ObjectBuffer$ is empty, add the newly transmitted record $record$ to $ObjectPool$ (lines 2–3);

- If the temporal gap is at least twice the alignment rate, every record related to the object of interest, is replaced by the recieved message (lines 5–9); otherwise,
- If the calculated distance is less than a given threshold (i.e., the newly transmitted message is not an outlier), it is added to $ObjectPool$ (lines 11–14).

In order to reduce the memory footprint of Algorithm 1, the buffer needs to be adjusted, so as to keep *only* the necessary AIS messages for temporal alignment, and discard the rest. Algorithm 3, which is in charge of that, given $ObjectPool$ as well $t_{pending}$ and t_{round}, centers the $ObjectPool$ around $t_{pending}$, and finally replaces the value of the latter with the value of t_{round}. In this way, we interpolate (with a constant delay – equal to the alignment rate) at most once per timestamp and extrapolate (virtually instantly) at most twice.

4 Evolving Clusters Discovery in Action

In this section we conduct a (baseline) set of experiments and measure the performance of EvolvingClusters using real-life mobility datasets.

4.1 Datasets and Preprocessing

The Datasets that we use to demonstrate the performance of Algorithm 1 in action, lie within the maritime domain. More specifically, we use two datasets, namely:

- The 'Brest' Dataset[1] [5]; it contains AIS information regarding maritime movement (mainly) in Brest Bay, France.
- The 'Saronikos' Dataset[2] []; it contains AIS information regarding maritime movement (mainly) in Saronikos Gulf, Greece.

Both datasets are open-source and the preprocessing methods used in order to be utilized properly by our Algorithm are described at [4].

4.2 Experimental Setup

Having presented the datasets at Section 4.1, we experimentally evaluate the real-life performance of Algorithm 1 on both datasets, with the parameters of the aforementioned algorithm set as follows:

- Cardinality Threshold (c): 5 objects;
- Temporal Threshold (t): 15 minutes;
- Distance Threshold (θ): 1000 meters.

All algorithms were implemented in Python3. The experiments were conducted in a single node with 8 CPU cores, 16 GB of RAM and 256 GB of HDD, provided by okeanos-knossos[3], an IAAS service for the Greek Research and Academic Community.

4.3 Real-Life Performance

While the algorithm produces interesting patterns with some of them potentially leading to insightful conclusions, (justifiable) concerns may arise because of the exploitation of graphs, since their search algorithms (e.g., Cliques, Maximal Connected Subgraphs, etc.) can escalate up to $\mathcal{O}(3^{|V|/3})$, where V, E are the graph's vertices and edges, respectively.

Dataset	#Records	Total Consumption Time	Avg. Consumption Time (per Message)
Brest (1 day)	142,227	30.52 min	12.87 ($\pm$12.66) ms
Saronikos (4 hrs.)	45,333	10.34 min	13.68 ($\pm$32.30) ms

Table 1: Performance Statistics of Algorithm 1

[1] The dataset is publicly available at zenodo.org
[2] Paper under construction.
[3] `https://okeanos-knossos.grnet.gr/home/`

However, despite the theoretical complexity, the algorithm runs in online mode in both real-life scenarios, as Table 1 illustrates. In a nutshell, focusing on a crowded day from the Brest dataset (resp. three days from the Saronikos dataset), we get an average of 15.12 ms (resp. 10.52 ms) for each message consumed.

At this point it's important to clarify that by 'online mode', we mean that the response time of the Algorithm is well-within the real-time threshold with respect to its step. For example, for hundreds of objects we tolerate a step in seconds, whereas for thousands of objects, tolerate a step in minutes. In other words, EvolvingClusters runs in 'online mode', if its processing time is less than its step.

5 Conclusion

In this report we describe the technical aspects regarding the integration of EvolvingClusters [4] to a real-time datastream. In the near future, we plan to integrate this algorithm to the i4Sea pipeline in two modes; the online mode, i.e. the methodology described at Section 2, and the offline mode as well, i.e., focusing on *historical data* from a certain spatial area S and temporal period T.

References

[1] Fosca Giannotti and Dino Pedreschi, editors. 2008. *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery*. Springer.

[2] Gennady L. Andrienko, Natalia V. Andrienko, Peter Bak, Daniel A. Keim, and Stefan Wrobel. 2013. *Visual Analytics of Movement*. Springer.

[3] Nikos Pelekis and Yannis Theodoridis. 2014. *Mobility Data Management and Exploration*. Springer.

[4] George S. Theodoropoulos, Andreas Tritsarolis, and Yannis Theodoridis. 2019. Evolvingclusters: online discovery of group patterns in enriched maritime data. In *MASTER@PKDD/ECML* (Lecture Notes in Computer Science). Volume 11889. Springer, 50–65.

[5] Cyril Ray, Richard Dreo, Elena Camossi, Anne-Laure Jousselme, and Clément Iphar. 2019. Heterogeneous integrated dataset for maritime intelligence, surveillance, and reconnaissance. *Data in Brief*.